FTL REPORT R87-6

PAPER AIRPLANE USER'S MANUAL: VERSION FOUR

Ronnie M. Lajoie

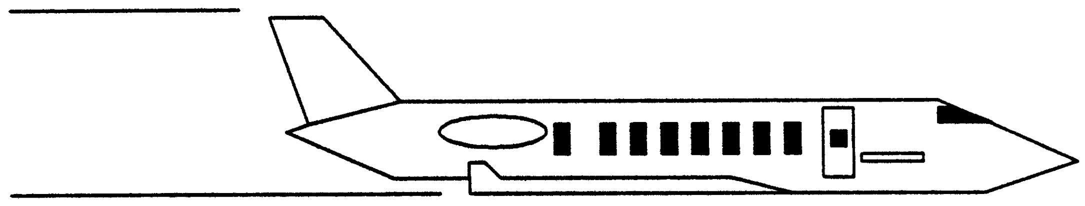
March 1987

MIT

DEPARTMENT
OF
AERONAUTICS
&
ASTRONAUTICS

FLIGHT TRANSPORTATION
LABORATORY
Cambridge, Mass. 02139

FLIGHT TRANSPORTATION LABORATORY R87-6



PAPER AIRPLANE USER'S MANUAL

Version IV

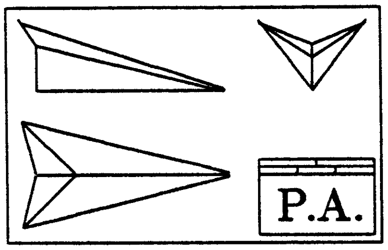
March 26, 1987

Ronnie M. Lajoie

Paper Airplane Project

Flight Transportation Laboratory

Massachusetts Institute of Technology



Preface

(Authors' note: This is an ***unfinished draft copy*** of the *Paper Airplane* User's Manual. As such, I would appreciate that you report any problems with language, spelling, format, etc, to the address at the end of this Preface.)

This document provides the user with a step-by-step guide to using *Paper Airplane*, Version IV. Although this program can be used to design any system (once given the proper set of design equations), this manual will use as a running example the design of an executive transport jet, the ***AM410-Laser***. This design was originally done by the author using manual calculations; where appropriate, comparisons will be made between the two methods.

This manual contains the following conventions for displaying examples:

- **Bold sans-serif text such as this** will represent user input requirements or options that you should enter ***exactly as shown*** — including upper and lower case.
- *Italic sans-serif text such as this* will represent user input requirements or options that you need to specify.
- ***Thin*** curly-brackets surrounding a list of user input separated by commas: {...,...} will represent a choice of requirements.
- ***Thin*** square-brackets surrounding user input: [...] will represent optional input.
- ***Thin*** square-brackets surrounding a list of user input separated by commas: [...,...] will represent a choice of options.
- `Typewriter-style text such as this` will represent the response from *Paper Airplane* or the system.

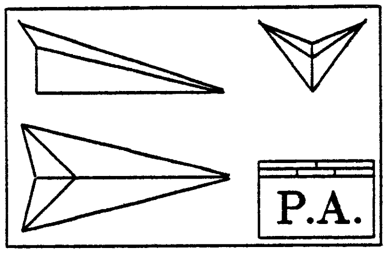
Paper Airplane Version IV runs under NIL release 329 or above on the DEC VAX family of processors under the VMS operating system. The NIL Lisp language system is available from M.I.T.'s Laboratory for Computer Science, 545 Technology Square, Cambridge MA 02139. *Paper Airplane* also runs under Zeta LISP on the Texas Instruments *Explorer*. *Paper Airplane* requires a VT100-compatible terminal, equipped with a keyboard containing arrow keys and a programmable keypad with PF-keys as well as numbers.

Version IV of *Paper Airplane* contains a substantial number of improvements over the last documented release. In particular:

1. Much faster convergence on design point solutions.
2. Capability to use multiple-input multiple-output design functions along with multiple-input single-output design functions.
3. Capability to use non-LISP based external computer programs as design functions (on VAX only).
4. An enhanced Design Function Exerciser.
5. Pre-defined table-lookup auxiliary functions.
6. Addition of a Library to store design variables and design functions from which design sets can be made.

Paper Airplane CAPD Project
Flight Transportation Laboratory, Room 33-412
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA 02139

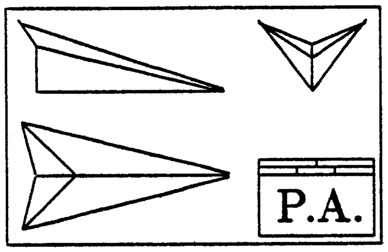
Please address any questions and comments to Mark Kolb at (617) 253-6883.



Contents

List of Figures

List of Tables



Chapter 1

Introduction

This chapter is an introduction to both the manual and the project. The first section explains the need for *Paper Airplane*-like systems and the history behind the project. The second section outlines the rest of the manual.

1.1 History of the Project

In the manufacturing environment, after a product is built, it is run through a series of tests: structural tests, acoustic tests, performance tests, thermal tests, safety tests, and many others; this is called "Product Testing." In the engineering environment, *before* a product is built, it is run through the same tests to decide whether or not the product *should* be built; this is called "Preliminary Design." Preliminary design is what takes an idea and possibly turns it into a blueprint for a product.

Representation of the Idea

Since the product is only an idea during preliminary design, an alternative representation of it must be found. This representation comes in the form of a mathematical model. A mathematical model of a simple metal screw, for example, must contain information on its geometric properties, its structural properties, its thermal properties and its electrical properties. Attach this screw to a metal plate and the mathematical model must not only include the aforementioned properties of both the screw and plate, but also the interaction of those properties between the two. Attach this plate to an avionics box and the mathematical model becomes very complex. Attach this avionics box to the cockpit of a commercial jetliner and the mathematical model becomes extremely complex.

To simplify the mathematical model, it is separated into many groups of components, or sub-systems, using a hierarchy similar to the one followed above. To simplify the mathematical model even more, each component model is further separated into groups according to its properties. Instead of one large and extremely complex mathematical model, preliminary design thus deals with many small and simpler sub-models. These sub-models are commonly referred to as engineering models, since these are the types of mathematical models an engineer usually deals with. Dividing the mathematical model into many engineering models also has the advantage that some engineering models of the idea may be common to other ideas already transformed into products.

The engineering models, depending upon on their own level of complexity, are physically represented by single equations, by sets of equations, and by computer programs. They are stored on magnetic tape and hard disk, in textbooks and notebooks, and on scraps of paper and piles of computer print-out.

The Problems with Engineering Models

In an ideal engineering environment,

1. The engineering models of an idea would be available in several different layers of complexity, ranging from a conceptual level to an advanced level of design.
2. At each level of design, there would be engineering models of that level's complexity to account for all parts of the proposed product and all of their properties. (Even though a structural model of an aircraft wing at the preliminary design level rarely includes the rivets and joint connections, the model should nevertheless account for them, even if it means merely adding some structural efficiency factor.)
3. The information on the proposed product would be stored in one secure central location and referenced by all the engineering models involved. This would insure that, for example, all engineering models requiring geometry information would acquire the *same* geometry information.

In the real engineering environment, however,

1. The engineering models of an idea are not always available in several different layers of complexity. For example, thermal models at the conceptual design level usually do not exist and their properties are usually ignored until the idea enters advanced preliminary design.
2. At each level of design, there are not always engineering models of that level's complexity to account for all parts of the proposed product and all of their properties. Instead, the missing engineering models and the information they contain are ignored (as mentioned above) or engineering models from more complex levels of design are substituted for the nonexistent simpler ones. This could be worse than ignorance since it brings unnecessary detail into the design at that level. It also can lock the design prematurely before all the degrees of freedom that a simple level of design has to offer are analyzed.
3. The information on the proposed product is scattered all over a company. The information resides in the company's main computer, in engineers' personal computer files, on notepads on engineers' desks, and on blueprints on drafters' tables. The time delay in acquiring needed information often results in an engineering model using assumed, and often conflicting, information. Wrong information can propagate throughout the design before it is finally detected and, expensively, corrected.

Part of the problem has been the enormity of the task. To correct Problems 1 and 2, more engineering models would have to be created; however, this would add to Problem 3. Making sure that many engineering models, scattered throughout a large company, never have conflicting information is an impossible job for a human being. Adding more human beings to the job requires one more human being above them to make sure that *they* communicate with each other. Most companies cannot afford either the manpower, money, or time to do this, thus business continues as usual.

A Computerized Solution

With the advent of smart computers and even smarter computer programmers, there may finally be a cost-effective means to monitor and handle the information engineering models require and produce and to insure that the design never has conflicting information.

Recent developments in computer technology have created engineering workstations, powerful cousins of personal computers. These new computers give the engineer the power of a large computer on a desktop. This, of course, means nothing if the engineer still has to write down the results or make a hard-copy to pass along design information. Other recent developments in computer technology and in communications technology have created very high speed networks that, once linked to several computers, provide instant communication between them. Even now, networks linking computers speed data across the country in a matter of minutes when it used to take days by mail and even by person.

Linking together engineering workstations is one thing, linking together engineers and their engineering models is another. The former is a matter of computer hardware; the latter, of computer software. The purpose of computer software is to do the same thing a human could do only faster, and repeatedly without adding mistakes. High-level computer programming languages, especially the object-oriented ones, now have the capability to monitor and handle design information between engineering models quickly and accurately. Computer databases now have the capability to house all the design information in one secure location plus allow for fast information storage and retrieval. A computer-based engineering model information sharing system (**CEMISS**) is now a cost-effective prospect to the engineering community.

The Paper Airplane Project

The *Paper Airplane* Project is the first one of its kind to apply this CEMISS idea to aerospace engineering. Although other computer programs have been developed that can share information between similar engineering models, they have been limited to certain types of products, such as general aviation aircraft [3] and naval airships [10], or to certain types of properties, such as NASTRAN[1] and MATRIX$_X$.[2] Neither group could

[1]NASTRAN is a finite-element modeling and analysis program for dealing with the structural and thermal properties of a component.

[2]MATRIX$_X$ is a mathematical modeling and analysis program for dealing with the dynamics and control

pass information on to any random engineering model. The goal of the *Paper Airplane* Project is to do just that.

The project was begun by Dr. Antonio Elias [2], former professor of Aeronautics and Astronautics at the Massachusetts Institute of Technology, in 1981 with a LISP-based code that could solve a simple system of design equations. Prof. Elias, working together with Mark Kolb ([4] and [5]), then a Master's candidate, later made *Paper Airplane* a user-friendly interactive test-bed for general systems of linear and non-linear design equations. In 1985, Ronnie Lajoie ([7] and [8]), another Master's candidate, joined the project to increase *Paper Airplane*'s domain to include multiple-output design functions and externally-based computer programs.

Paper Airplane itself is a Computer-Aided Preliminary Design (CAPD) tool; that is, a computer program designed to aid an engineer in the conceptual and preliminary design of an aircraft or any other system capable of being described by a set of scalar parameters. Superficially, *Paper Airplane* is a "simultaneous calculator"; that is, a calculator capable of determining the values of a set of parameters satisfying a set of linear and non-linear simultaneous functions. In this sense, *Paper Airplane* might be viewed as an "engineer's spreadsheet" program, similar to TK!Solver [13]. But while TK!Solver can only handle simple algebraic expressions, *Paper Airplane* can also handle complex multiple-input single-output (MISO) functions such as numerical integrators, and complex multiple-input multiple-output (MIMO) functions such as computer programs.

Paper Airplane now has the capabilities to perform automatically many of the computer tasks now performed manually by an engineer, such as setting up input files, executing and monitoring codes, and converting output information to data required by another code or by another engineer. *Paper Airplane* also has a user-friendly interface so that the engineer with little programming knowledge can work it as easily as one with expert knowledge. In these two regards, *Paper Airplane* has earned the status of an early prototype of a CEMISS.

Future Research Requirements

To make *Paper Airplane* reach the full status of a CEMISS, however, several more capabilities still need to be added.

1. *Paper Airplane* will require the ability to perform automatic trade studies and optimizations of design variables if it is to be of any value to the modern engineer. *Paper Airplane* already has the ability to compute a "performance function" based upon the weights applied to design variables. What it lacks is a general method to minimize or maximize that performance function and to do it as efficiently as possible.

2. *Paper Airplane* will require a "Librarian" to accompany the *Paper Airplane* Library. Much time can be saved by having computerized help in sorting through the collection of design functions and design variables to assemble the proper design set to solve the problem at hand.

properties of a component.

3. *Paper Airplane* will require a database to house all the design information if it is to properly integrate real engineering models. *Paper Airplane* just doesn't have the memory to store all the information of a design (such as the mathematical points defining a surface mesh). It needs a link to a database that can store and retrieve design information quickly and efficiently, so that an engineer can acquire any information he or she requires when it is required.

4. *Paper Airplane* will require the ability to handle non-scalar design variables if it is to handle real engineering models. Geometry information is sometimes best handled in a drawing; and tabular information, in a graph. *Paper Airplane* will require the ability to accept design information in this form as well as to give it out.

5. *Paper Airplane* will require the ability to communicate with the computers best suited for handling the engineering models. This will require a network interface capability in addition to a modified external code interface.

6. Finally, *Paper Airplane* will require the ability to perform parallel processing of tasks that can be parallel processed. Such an ability would greatly reduced the processing time of numerical searching for the solution well beyond the reductions already made.

With all these new abilities, *Paper Airplane* would finally be considered a true CEMISS. Would this mean that engineers would lose their jobs? No way. Even with all these capabilities, *Paper Airplane* would still be only a computational tool to the modern engineer. The key to finding a good numerical solution quickly is to start with a good intelligent guess at it; such information would still only come from an engineer. An expert system to teach *Paper Airplane* engineering is still a long, long way down the road.

1.2 Outline to the Manual

The following chapters and appendices guide the user through all the essential, and nonessential, features of *Paper Airplane*.

Chapter 2 defines the terminology of *Paper Airplane* and takes the reader through a simple example.

Chapter 3 details the first, and usually more difficult, phase of the use of *Paper Airplane*; that of assembling the proper design set to solve your problem, and creating its corresponding source file.

Chapter 4 details the second phase of the use of *Paper Airplane*; exercising the design set to find the solution to your problem.

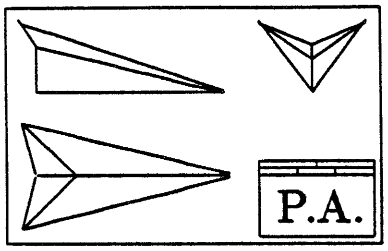
Appendix A instructs the user in the basics of LISP required to write the LISP code to a mathematical design equation or function.

Appendix B instructs the user in the basics of LaTeX required to produce "fancy" output. (It is ***not*** a necessity.)

Appendix C instructs the user in the mechanics of using *Paper Airplane*'s pre-defined auxiliary functions, namely the table-lookup functions and the external code interface function.

Appendix D gives the user a complete listing of the units pre-defined under the *Paper Airplane* Dimensions and Units Package (PDUP).

Appendix E gives the user a complete listing of the menu functions available in *Paper Airplane* and their results.



Chapter 2

A Paper Airplane Primer

This chapter is a primer to *Paper Airplane*. The first section explains the terminology of *Paper Airplane*. The second section explains the usage and basic concepts. The third section takes the reader through a simple, but complete, example.

2.1 The Terminology of Paper Airplane

This section explains the terminology of *Paper Airplane*. New terms appear in **bold sans serif** where they are defined. These terms also appear in the glossary at the end of this manual.

design variable: is a scalar parameter, such as Vehicle Length or Vehicle Weight, whose value uniquely determine part of the configuration of an aircraft, spacecraft, or any other system. A design variable has a number of attributes associated with it, such as its value, its dimensions, its order of magnitude, and the limits of its value.

design function: is a relationship between design variables. A design function can range in complexity from a simple algebraic equation to a very large and complex computer program.

design set: is a set of certain design functions and the design variables those functions relate towards the goal of solving a particular design problem.

auxiliary function: is a COMMON LISP function that can be called by any design function to perform a generic task (such as a table lookup).

source file: is a computer file containing the information on all of the design variables and design functions to be loaded internally into a *Paper Airplane* design set.

loading: is a COMMON LISP term for reading and evaluating LISP code from a file into main memory.

variable tableau: is a spreadsheet of information on the design set arranged on a computer screen. This information includes a list of design variables and their current values, units, and states.

variable state: is the condition of the value of a design variable. Variable states come in the following three varieties, which are assigned to design variables according to their initial letter.

> **Initialized-value state:** This indicates a design variable that has been given a known value by the user. A design variable obtains state I whenever the user changes its value, or when the user **freezes** it. I-state design variables, officially designated as **base variables**, will be referred to simply as **knowns**.
>
> **Guessed-value state:** This indicates a design variable that has been given a trial value by the user. A design variable obtains state G whenever the user **floats** it. G-state design variables, officially designated as **derived variables**, will be referred to simply as **unknowns**.
>
> **Computed-value state:** This indicates a design variable that *had* been given a trial value by the user, and was later given a known value by *Paper Airplane*. A design variable obtains state C only when the user **processes** the design set; and then only if *Paper Airplane* can find a solution which satisfies all the design functions in the user's design set.

design point: is the values and states of all the design variables in a design set at any stage in the design process.

design path: is the selection of certain design variables as knowns and the rest as unknowns; thereby setting up some implied path, or sequence of design functions, for *Paper Airplane* to follow once values are provided for the design variables.

computational agenda: is the actual path, or sequence of design functions, to be evaluated to find the values of the unknowns once given the initialized values for the knowns and the guess values for the unknowns. The computational agenda is also called the **computational path**. The computational agenda consists of a forced path and loops.

forced path: is a sequence of perfectly constrained design functions, each of which can be solved individually, although sequentially. The path is called "forced" since there is no alternative but to solve the design functions in this sequence in order to compute the values of their unknowns.

loop: is a sequence of perfectly constrained design functions, each of which computes values required by other design functions in a closed loop. Loops are solved by guessing the value of a **forcing variable** to compute two independent values of a **loop variable**. When the two values converge, the values of all the unknowns involved can be found.

2.2 The Usage of Paper Airplane

A user of *Paper Airplane* must first gather the engineering knowledge he or she will need and represent it as equations, functions, and/or computer programs. (A **function** is an internal piece of code written in COMMON LISP; whereas a **computer program** is an external piece of code usually not written in COMMON LISP.) Next, that information must be coded as design variables and design functions into a source file; the format to be followed is shown in Figure 2.1. The user then starts up *Paper Airplane* and loads the source file into it.

Figure 2.2 shows the spreadsheet-like tableau a user may see once the source file is loaded into a design set. The columns are for the design variable names, states, current values, and current units. The name of the tableau appears at the top since a design set can have more than one tableau (to better organize design set information).

The user selects the design path by changing design variable states to the best of his or her knowledge. As long as the number of unknowns equals the total number of values computed by all the design functions, *Paper Airplane* should then be able to build a computational agenda to solve for all the unknowns. If the number of unknowns are less than this total, as in an **overconstrained** problem, *Paper Airplane* would eventually come across an overconstrained design function it could not solve for; and if the number of unknowns are greater than this total, as in an **underconstrained** problem, *Paper Airplane* would eventually come across an underconstrained design function it could not solve for or a loop it could not close, and thus not solve either.

Once the design path is selected, the user then provides initialized values for the knowns and guess values for the unknowns to define the **initial design point** of the design set. The user then instructs *Paper Airplane* to process the design set to find the true values for the unknowns.

Paper Airplane processes the design set in combinations of three techniques:

1. It may evaluate a design function *directly*, if all output values of the design function are unknown, and all input values are known. This one-time single-function evaluation is called **forward computation.**

2. It may *invert* a design function, if a number of output values of the design function are known, and the *same* number of input values are unknown. *Paper Airplane* will attempt to *numerically* invert that design function by repeatedly evaluating it in order to find the values of the unknowns. Almost always, this will be successful, and the values of the unknowns will be obtained. This iterative single-function evaluation is called **reverse computation.**

3. It may *iterate* a set of design functions in a loop, if those design functions form a closed loop containing several interdependent unknowns. *Paper Airplane* will repeatedly guess values for a chosen forcing variable until the two computed values of a loop variable converge; thereby stabilizing the values of all the unknowns of the design functions in the loop. This iterative multiple-function evaluation is called **loop computation.**

After *Paper Airplane* builds the computational agenda to solve the design path (see following section for details of this process), it then uses numerical methods to find a numerical solution to the initial design point. If a solution is found, *Paper Airplane* will update the values of the unknowns and make them computed knowns, as shown in Figure 2.3. (Note that *Paper Airplane* also informs the user when values obtained in the solution fall outside the recommended limits.)

Now that the user has one solution, design variable values can be changed to form new initial design points to find more solutions as part a trade study or design optimization. (Automatic trade study and optimization features have yet to be incorporated into *Paper Airplane*; however, they are matters under research.) Design variable states can also be changed to form new design paths to find new computational agendas and lead to new types of solutions, trade studies, and optimizations.

Agenda Building

Given a design path of design variables selected as knowns and unknowns, *Paper Airplane* assembles its methodology for solution into a computational agenda. This process is commonly referred to as **agenda building**. This section briefly describes the process of agenda building, since it is this process that makes *Paper Airplane* a unique (so far) and powerful computer-aided preliminary design tool. (For complete details on agenda building, see [5].)

The key to understanding agenda building is that it only involves the knowns, the unknowns, and the design functions that use them. No design function is evaluated and no numbers are produced. Each design function is merely examined to find out what kind of design variables (knowns or unknowns) go in, and what kind come out. In this manner, an order of function evaluation can be set up without the need to resort to more difficult and time-consuming artificial intelligence methods.

An iterative search is performed to find the design function with the least amount of unknowns, whether they are going in or coming out. At any time, if the number of unknowns of the design function is less than the number of values it computes, that overconstrained design function is discarded and the unknowns involved are labeled "inconsistent."

If the number of unknowns of the design function equals the number of values it computes, however, the design function can be solved for using the forward computation method, the reverse computation method, or (for MIMO design functions) a combination of both. The design function is then labeled "used" and is placed as an agenda entry into the forced path of the computational agenda. An **agenda entry** is merely the design function and the unknowns to be solved for by it. The states of the unknowns are then changed to C and the design variables are then treated as knowns. In this manner, a design function that was initially underconstrained can become perfectly constrained because of the solution of another design function.

When the search returns a design function whose number of unknowns is greater than the number of values it computes (i.e., an underconstrained design function), the forced path construction is ended and the loop construction begins. The choice of forcing variable for a possible loop is the unknown most common to the remaining unused design functions. The state of the forcing variable is temporarily set such that the search will treat it is a known. As long as the search keeps returning perfectly constrained design functions, a forced path of **preliminary entries** will be constructed. This construction stops when an overconstrained or an underconstrained design function is returned.

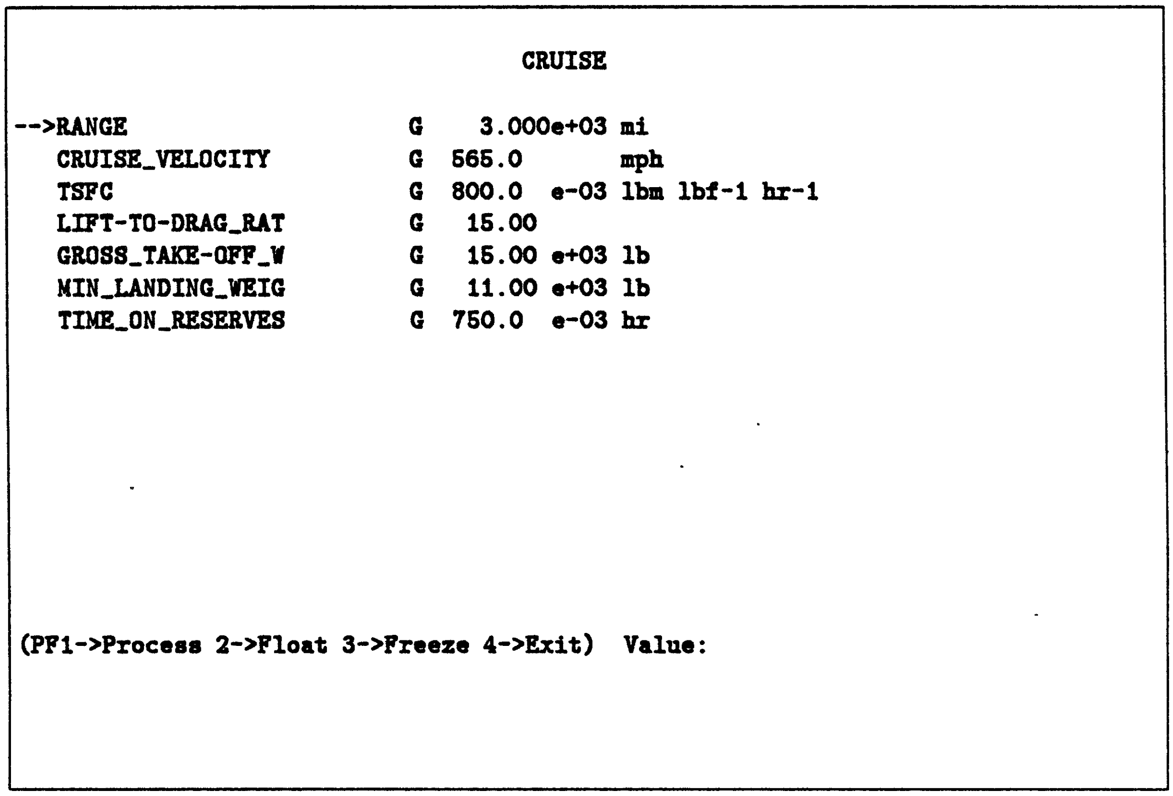
If an overconstrained design function is returned, it is checked to see if it has a computed unknown common to one of the design functions in the preliminary entries. If it does, then there are two design functions that can independently compute the value of the same unknown, then called the loop variable; thus the loop can be closed. If it doesn't however, then the overconstrained design function is discarded and its unknowns are labeled "inconsistent." On the other hand, if an underconstrained design function is returned, the loop can never be closed; thus a new forcing variable must be chosen.

The preliminary entries of any closed loop are organized into an initial path, two branches, and a final path. The **initial path** is a sequence of perfectly constrained design functions whose computed unknowns are required by both branches. The **branches** are two independent sequences of perfectly constrained design functions for computing the value of the loop variable. Lastly, the **final path** is a sequence of perfectly constrained design functions whose unknowns can be solved for once the loop has converged.

Agenda building continues until all the design functions are used or discarded, or until a loop construction failure occurs, when all possible forcing variables have been tried to construct a loop and have failed. If all the design functions are used, *Paper Airplane* should be then able to compute a unique numerical solution for any initial design point obeying the design path. If design functions have been discarded, however, any numerical solution found will have inconsistencies. On the other hand, if a loop construction failure occurs, the computational agenda will be incomplete, and *Paper Airplane* will only be able to find a partial numerical solution to any initial design path.

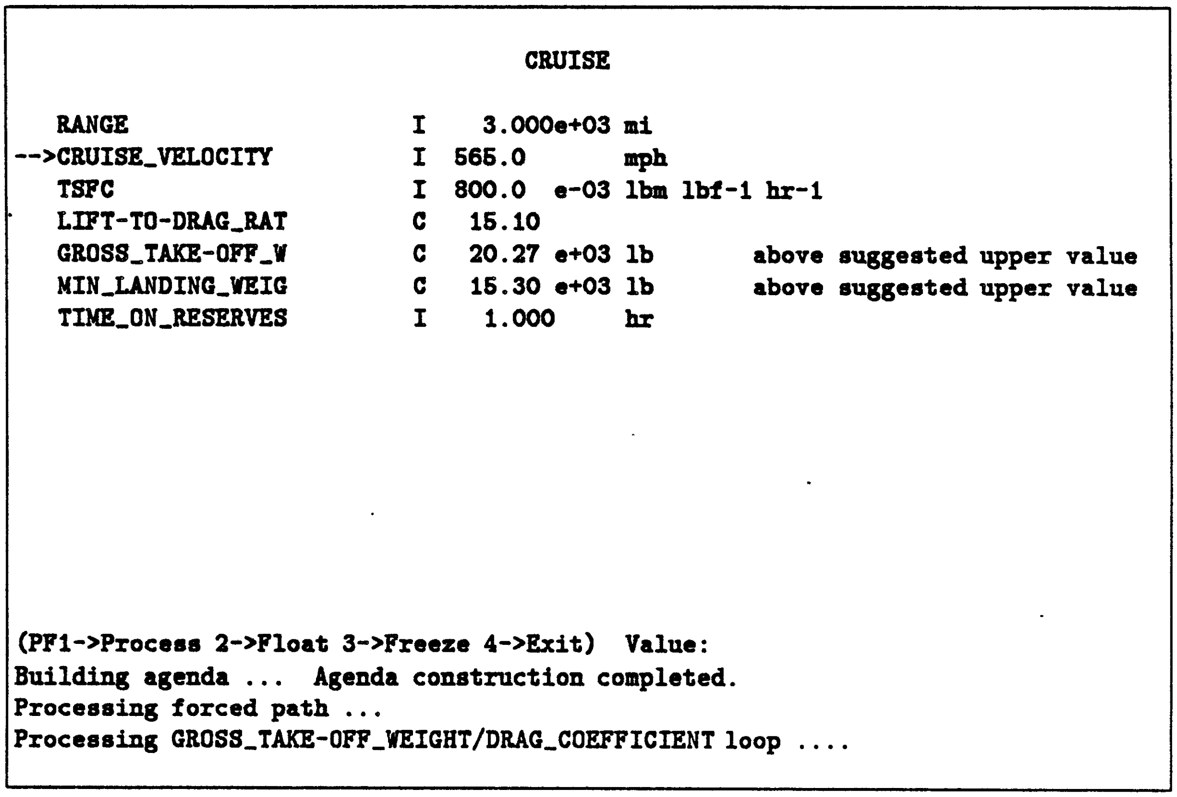
```
(pa-defvar WING_REFERENCE_AREA
           :category (geometry wing)
           :documentation "The Reference Area of the Wing."
           :TeX-name "$S_{ref}$"
           :order-of-magnitude 261.0
           :lower-value 220.0
           :upper-value 300.0
           :dimensions "12"
           :default-units "ft2")


(pa-defun DF-1
          :category weights
          :computed-variables (GROSS_TAKE-OFF_WEIGHT "lbf")
          :input-variables ((PAYLOAD_WEIGHT "lbf")
                            (FUEL_WEIGHT "lbf")
                            (EMPTY_WEIGHT_FRACTION ""))
          :function-body (/ (+ PAYLOAD_WEIGHT FUEL_WEIGHT)
                            (- 1 EMPTY_WEIGHT_FRACTION))
          :TeX-name "$W_{gto} {} = {} {{W_p + W_f} \\over {1 - f_e}}$"
          :documentation "Gross Take-off Weight Equation.")
```

Figure 2.1: Example design variable and design function declarations.



```
                                   CRUISE

-->RANGE                    G    3.000e+03 mi
   CRUISE_VELOCITY          G  565.0         mph
   TSFC                     G  800.0  e-03 lbm lbf-1 hr-1
   LIFT-TO-DRAG_RAT         G   15.00
   GROSS_TAKE-OFF_W         G   15.00 e+03 lb
   MIN_LANDING_WEIG         G   11.00 e+03 lb
   TIME_ON_RESERVES         G  750.0  e-03 hr








(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

Figure 2.2: Example tableau as it first appears.



```
                              CRUISE

   RANGE                      I    3.000e+03 mi
-->CRUISE_VELOCITY            I  565.0        mph
   TSFC                       I  800.0   e-03 lbm lbf-1 hr-1
   LIFT-TO-DRAG_RAT           C   15.10
   GROSS_TAKE-OFF_W           C   20.27  e+03 lb        above suggested upper value
   MIN_LANDING_WEIG           C   15.30  e+03 lb        above suggested upper value
   TIME_ON_RESERVES           I    1.000      hr







(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
Building agenda ...  Agenda construction completed.
Processing forced path ...
Processing GROSS_TAKE-OFF_WEIGHT/DRAG_COEFFICIENT loop ....
```

Figure 2.3: Example tableau after design set processing.

Symbol	*Paper Airplane* Name	Definition
AR	`ASPECT_RATIO`	Aspect Ratio of the Wing.
C_D	`DRAG_COEFFICIENT`	The Drag Coefficient of the aircraft at Cruise.
C_{D_0}	`ZERO-LIFT_DRAG_COEFF`	The Zero-lift Drag Coefficient of the aircraft.
C_L	`LIFT_COEFFICIENT`	The Lift Coefficient of the aircraft at Cruise.
ϵ	`OSWALD_EFFICIENCY`	The Oswald Efficiency of the Wing.
f_e	`EMPTY_WEIGHT_FRACTION`	The Empty (structural) Weight Fraction of the aircraft.
$^L/_D$	`LIFT-TO-DRAG_RATIO`	The Lift-to-Drag Ratio of the aircraft at Cruise.
R	`RANGE`	The Range of the aircraft.
S_{ref}	`WING_REFERENCE_AREA`	The Reference Area of the Wing.
TSFC	`TSFC`	The Thrust Specific Fuel Consumption of the Engines.
T_{res}	`TIME_ON_RESERVES`	The Time Available using the aircraft's Fuel Reserves.
V_{Cr}	`CRUISE_VELOCITY`	The Velocity of the aircraft at Cruise.
W_{Cr}	`CRUISE_WEIGHT`	The Weight of the aircraft at Cruise.
W_f	`FUEL_WEIGHT`	The Weight of the aircraft's Fuel Supply.
W_{gto}	`GROSS_TAKE-OFF_WEIGHT`	The Gross Weight of the aircraft at Take-off.
$W_{l_{min}}$	`MIN_LANDING_WEIGHT`	The Minimum Weight of the aircraft at Landing.
W_p	`PAYLOAD_WEIGHT`	The Weight of the aircraft's Payload.

Table 2.1: Design Variables for Tutorial Number 1.

2.3 Tutorial Number 1

This section presents a tutorial using the example of the *AM410-Laser* design. The design set for this example consists of 17 design variables and 7 design functions and illustrates the problem of determining the basic design characteristics of the aircraft (while minimizing its weight) so that it will obtain its required range. This example illustrates all of the basic concepts, features, and problems with *Paper Airplane*.

The tutorial begins by presenting the design set, that is, the 17 design variables and 7 design functions (which at this stage are just simple equations). How this information is coded into the source file will be explained in detail in a later chapter. The tutorial will begin at the point of starting up *Paper Airplane* on the computer and loading the source file. Then it will take you through the world of *Paper Airplane* and allow you to try out firsthand the early design of the *AM410-Laser*.

2.3.1 The LASER Design Set

Table 2.1 summarizes the 17 design variables, including their traditional mathematical symbols, their names inside of *Paper Airplane*, and their definitions — information which comes straight from the source file "`PA$DISK:[EXAMPLES]EXAMPLE1.SOU`".

The 7 design functions, which have been labeled simply `DF-1` through `DF-7`, are classical simple relationships which can be found in any book on aircraft design, such as the ones by Nicolai [12] and Torenbeek [15].

`DF-1` This is an equation for Gross Take-off Weight as a function of Payload Weight, Fuel Weight, and the Empty Weight Fraction.

$$W_{gto} = \frac{W_p + W_f}{1 - f_e}$$

DF-2 This is a simple equation for Cruise Weight which assumes that it is equal to the Gross Take-off Weight minus two-thirds of its Fuel Weight.

$$W_{Cr} = W_{gto} - \frac{2}{3}W_f$$

DF-3 This is an equation for Minimum Landing Weight in terms of its Gross Take-off Weight and a fraction of its Fuel Weight (like the equation above). In this case the fraction is the aircraft's endurance (equal to the Range divided by the Cruise Velocity) over the sum of the endurance plus the Time Available on Fuel Reserves.

$$W_{l_{min}} = W_{gto} - \left[\frac{R/V_{Cr}}{(R/V_{Cr}) + T_{res}}\right] W_f$$

DF-4 This is a form of the famous Bréguet Range Equation, relating Range to Cruise Velocity, Thrust Specific Fuel Consumption, Lift-to-Drag Ratio, and the ratio of Gross Take-off Weight to Minimum Landing Weight.

$$R = \frac{V_{Cr}}{\text{TSFC}} \frac{L}{D} \log\left(\frac{W_{gto}}{W_{l_{min}}}\right)$$

DF-5 This is the definition of Lift-to-Drag Ratio, which is simply Lift Coefficient divided by Drag Coefficient.

$$^{L}/_{D} = \frac{C_L}{C_D}$$

DF-6 This is an equation for Lift Coefficient at Cruise (when the aircraft's lift just balances its Cruise Weight). The variables in the denominator are the dynamic pressure (one-half the air density times the Cruise Velocity squared) and the Wing Reference Area.

$$C_L = \frac{W_{Cr}}{1/2\rho V_{Cr}^2 S_{ref}}$$

DF-7 This is an equation for Drag Coefficient at Cruise including only the Zero-Lift Drag Coefficient and the "wave drag" coefficient, which is equal to the Lift Coefficient squared divided by the product of pi, the Aspect Ratio, and the Oswald Efficiency.

$$C_D = C_{D_0} + \frac{C_L^2}{\pi AR\epsilon}$$

By the time the design variables and functions are selected, the user should have a good idea which design variables will be the knowns and which ones will be the unknowns. Keep in mind that *Paper Airplane* allows *any* design variable to be a known or unknown; the selection process is governed by the rules of engineering, not the limitations of *Paper Airplane*.

For this first tutorial, the engineering reasoning goes as follows. Since there are only 7 design functions to 17 design variables, 10 design variables will have to be initialized. Design requirements usually specify Range, Cruise Velocity, and Payload Weight, so those three will obviously be knowns. Likewise, obvious unknowns will be Gross Take-off Weight, Fuel Weight, Cruise Weight, and Minimum Landing Weight. Another common unknown is Lift-to-Drag Ratio, which suggests that Lift Coefficient and Drag Coefficient are unknowns as well. Those three added to the four unknown weights yield the seven unknowns the seven design functions can be used to solve for; the rest must be knowns. FAA regulations will supply the value for Time Available on Fuel Reserves; however, there are no regulations to supply the values for the last six design variables: Aspect Ratio, Wing Reference Area, Zero-Lift Drag Coefficient, Oswald Efficiency, Empty Weight Fraction, and Thrust Specific Fuel Consumption. The values of these six variables are arbitrary, although technology does impose limits on them. These are the true design variables that the design engineer can manipulate.

2.3.2 Starting up Paper Airplane

Once the design set has been coded into the source file, *Paper Airplane* should be started and the source file loaded into it. At this point the reader should be logged on to a *Paper Airplane* equipped computer, so that you can follow along as you read on. If you are not logged on to a VAX/VMS NIL-equipped computer, the following steps may not apply to you. If they do not, please consult your local systems operator to find out how to start up *Paper Airplane* on your computer. Then move on to the next section to continue with the tutorial.

Before you can start up *Paper Airplane*, you must first enable the NIL environment. Once enabled, it can be permanently left on the user's session (since one can leave and re-enter NIL at will). If you are a regular NIL user, then you probably have a "`NIL.COM`" and a "`NIL.INI`" file, and the following may be different from what you are used to. New NIL users should do the following from the VAX DCL ($-prompt) environment:

```
$ nil Return
This is NIL, MIT Common Lisp
      :
Type your terminal type, without quotes, then <return>: vt100 Return

This is NIL, MIT Common Lisp, running on WILBUR-WRIGHT.
 Experimental Steve   17.8
 Experimental Lisp   329.0
 VM                 1177
;Reading WILBUR::NIL$DISK:[NIL.SITE]DEFAULT.INI;3 into package USER.
```

```
;WILBUR::NIL$DISK:[NIL.SITE]DEFAULT.INI loaded.  Runtime = 2.17 seconds;
; 496 pagefaults, thrash coefficient = 228.57 pgs/sec.
"Experimental Steve 17.8, Experimental Lisp 329.0, VM 1177"
*
(setq si:*print-gc-messages* nil)
NIL
```

(The actual response may differ slightly, but it should have this basic format.) Once you are in NIL, you must *load Paper Airplane* into it:

```
*
(load "pa$disk:[utils]fastload")
;Reading WILBUR::PA$DISK:[UTILS]FASTLOAD.LSP;1 into package USER.
      ⋮
;WILBUR::PA$DISK:[UTILS]FASTLOAD.LSP; loaded.  Runtime = 77.82 seconds;
; 29364 pagefaults, thrash coefficient = 377.33 pgs/sec.
#<LOCAL-VMS-PATHNAME "WILBUR::PA$DISK:[UTILS]FASTLOAD.LSP;1" 1B7454>
```

You will notice that NIL begins to process the loading function immediately after you type the closing right parenthesis, without waiting for a Return or other *entering* keystroke. This is because NIL has a *hot* reader, similar to those of Lisp Machines, which detects when your input is logically complete. After *Paper Airplane* has successfully loaded, you start it by doing the following:

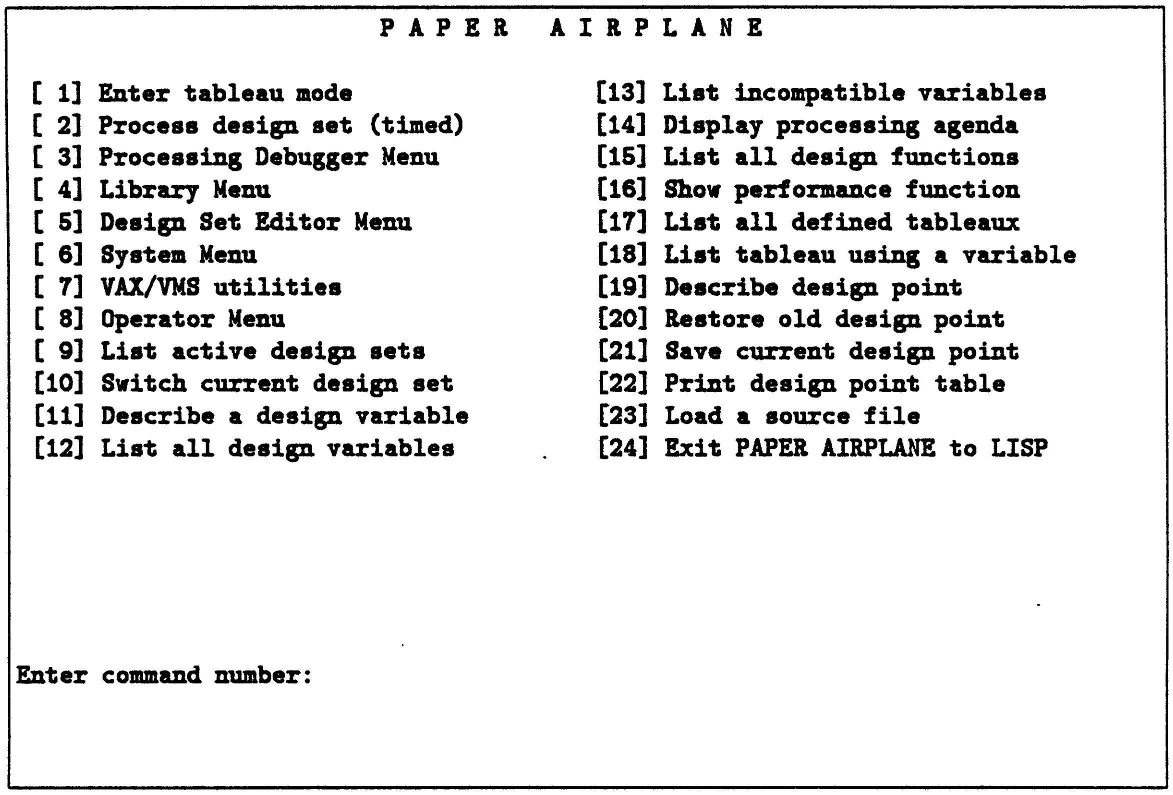
```
#<LOCAL-VMS-PATHNAME "WILBUR::PA$DISK:[UTILS]FASTLOAD.LSP;1" 1B7454>
(pkg-goto 'pa)
#<PACKAGE PA>
(pa-directory "pa$disk:[examples]")
#<LOCAL-VMS-PATHNAME "WILBUR::PA$DISK:[EXAMPLES]POINT.DAT" 1DC4D4>
(pa)
```

The first line you type instructs NIL to enter a special environment, or *package*, where the *Paper Airplane* LISP functions are defined. The second line you type defines the default directory where your source files are stored. For all tutorials, the source files are stored in the directory "`PA$DISK: [EXAMPLES]`." And finally, the last line you type instructs NIL to execute *Paper Airplane*. The screen will clear and be replaced by a welcoming message. Press any key, and the screen will be replaced by Figure 2.4, the main menu of *Paper Airplane*.

2.3.3 Loading the Source File

The main menu of *Paper Airplane*, Figure 2.4, contains the first 24 commands available to the user. The user selects one of them by entering the number associated with that command (again, both NIL and LISP have hot readers, so *any* non-alphanumeric keystroke after the number, such as a space, will terminate the input).

Since *Paper Airplane*, at this point in the tutorial, is devoid of design sets, you must "`Load a source file`" into it. To load the first tutorial source file, follow these steps:



```
                         P A P E R   A I R P L A N E

 [ 1] Enter tableau mode                  [13] List incompatible variables
 [ 2] Process design set (timed)          [14] Display processing agenda
 [ 3] Processing Debugger Menu            [15] List all design functions
 [ 4] Library Menu                        [16] Show performance function
 [ 5] Design Set Editor Menu              [17] List all defined tableaux
 [ 6] System Menu                         [18] List tableau using a variable
 [ 7] VAX/VMS utilities                   [19] Describe design point
 [ 8] Operator Menu                       [20] Restore old design point
 [ 9] List active design sets             [21] Save current design point
 [10] Switch current design set           [22] Print design point table
 [11] Describe a design variable          [23] Load a source file
 [12] List all design variables           [24] Exit PAPER AIRPLANE to LISP




Enter command number:
```

Figure 2.4: *Paper Airplane* Main Menu

```
Enter command number: 23 [Return]
Load file (WILBUR::PA$DISK:[EXAMPLES]SOURCE.SOU): example1 [Return]
File WILBUR::PA$DISK:[EXAMPLES]EXAMPLE1.SOU;1 loaded.
Enter command number:
```

The first thing you enter is the file specification of the source file. Since you told *Paper Airplane* what the default directory would be (as indicated by the name in parentheses), you do not need to do so again. All you need to supply is the information not identical to the name in parentheses, which in this case is the file name **example1**. You will have to press a carriage return because NIL's hot reader is turned off during this reading.[1] After you enter the source file name, *Paper Airplane* loads the source file into a design set, a process which may take 10–20 seconds for the LASER design set and longer for larger design sets. Once the source file has been loaded (as denoted by the reappearance of the "`Enter command number:`" prompt), you are ready to exercise the design set.

If you ever need to return to the NIL environment from *Paper Airplane*, you can do so easily by selecting the "`Exit PAPER AIRPLANE to LISP`" command, [24] in the main menu. This command allows you to leave *Paper Airplane* ***without terminating it***.[2] To re-enter *Paper Airplane*, just type (pa) and you will be brought back to *Paper Airplane*'s main menu. Similarly, you can non-destructively return to VMS from *Paper Airplane* by entering (valret) instead of a command number, and re-enter *Paper Airplane* by typing nil:

```
Enter command number: 24 [Return]
       ⋮
THANK YOU FOR USING PAPER AIRPLANE
       ⋮
T
(pa)
Enter command number: (valret)
$ nil [Return]
Enter command number:
```

Paper Airplane, being a research tool, is *far* from error-free, and it actually takes very little to upset it to the point where it will dive into debug mode (where you will see a prompt such as "1>xdbg>"). Fortunately, it also takes very little to recover back to the point where the error occurred. Merely press [Ctrl-G] to abort debug, then enter (pa) again:

```
Enter command number: (crash)
       ⋮
```

[1]A carriage return is always acceptable as a command terminator; however, it is *mandatory* when entering file pathnames and other text-type data.

[2]Actually, there is no way to *really* terminate *Paper Airplane*. To do so, you would also have to terminate NIL using the NIL (quit) function.

```
1>xdbg> Ctrl-G
1>xdbg>[Abort]
*
(pa)
Enter command number:
```

You will be brought back to the main menu of *Paper Airplane*, but all the changes you made to the design set will still be present. Likewise, if you need to stop *Paper Airplane* during a process, press Ctrl-C and you will be brought into the interrupt mode (where you will see an "`Interrupt>`" prompt). Again, press Ctrl-G to abort the interrupt, then enter **(pa)**:

```
Enter command number: Ctrl-C
Interrupt> Ctrl-G
Interrupt> ;Quit!
*
(pa)
Enter command number:
```

2.3.4 Exercising the LASER Design Set

In this section, you will learn how to exercise the LASER design set in order to perform conceptual design of the *AM410-Laser*. The interfaces to the design set are the tableaux defined in the source file. On a VT100 terminal, each tableau can contain 15 design variables, while on a Lisp Machine 48 design variables can be displayed at one time. In practice, it is better to group the design variables in smaller sets, according to the particular aspect of the design one wishes to concentrate on. On a large design, one may have a dozen or so tableaux, labeled "fuselage", "aerodynamics", "propulsion", "mission profile" and so on. The tutorial design set contains three tableaux. Command `[17]` will "`List all defined tableaux`" in alphabetical order:

```
Enter command number: 17 Return
AERODYNAMICS CRUISE WEIGHTS
Enter command number:
```

To display a particular tableau, use command `[ 1]`, "`Enter tableau mode`". Since this is by far the most often-used *Paper Airplane* command, it appears on both the main menu and most sub-menus, and it usually is command `[ 1]`:

```
Enter command number: 1 Return
Enter tableau name: ?
AERODYNAMICS CRUISE WEIGHTS
Enter tableau name: CRUISE Return
        ⋮
```

Notice that if you type a question mark, *Paper Airplane* will produce the same list of tableaux as it produced for command [17]. Also, you do not need to type the full name of the tableau you want, for *Paper Airplane*, like VMS, recognizes the name as long as it is given enough letters for it to distinguish your choice from the rest. For example, you could have typed merely C rather than CRUISE and *Paper Airplane* would still have selected the CRUISE tableau shown in Figure 2.5.

Each tableau consists of its name, which appears at the top of Figure 2.5, and five columns of design variable information. The first column contains the names of the design variables; the second, their current states (which are initialized to G); the third, their current values (which are initialized to their orders of magnitude — default values); and the fourth, their current units. The fifth column is reserved for comments or warnings about each design variable. When everything is going smoothly, the fifth column should be blank.[3]

The arrow (-->) pointing to "RANGE" marks the Range design variable as current, which means that this design variable can be manipulated. For example, you can change its state, its value, and/or its units; though, of course, you cannot change the units of a design variable that is dimensionless, such as Lift-to-Drag Ratio.

You will need to use the keypad keys and the arrow keys to operate on a tableau. If your keyboard lacks either of these, then stop what you are doing, and find a keyboard that has them. If your keyboard has these, then sorry about the interruption.

The major operations you can perform on a design variable are described below:

Getting Information on a Design Variable

To get information on a design variable, such as the limits to its value, press the $\boxed{4}$ key on the keypad:

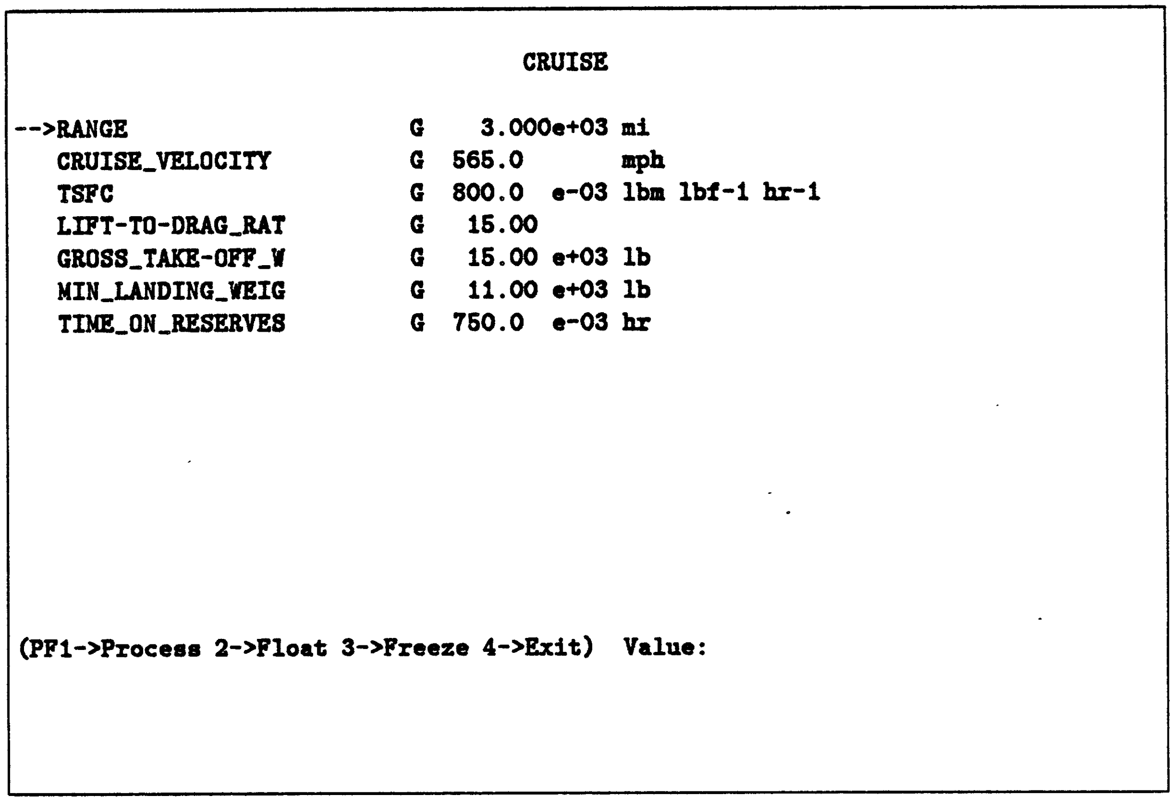
```
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:  [4]
Display RANGE brief or full? [Return]
```

Since, at this stage, both brief and full descriptions are the same, a $\boxed{\text{Return}}$ key is the easiest reply. The tableau screen will be replaced by the screen shown in Figure 2.6. This description screen contains the name of the design variable, its LaTeX name, its documentation (or help information), its dimensions (""l"" for "length"), its performance function weight (not currently available), its current state, its current value and units, its order of magnitude (default value) and limits, the tableaux it appears in, and the functions that can compute it. To return to the CRUISE tableau, simply press any key.

Changing a Design Variable's State

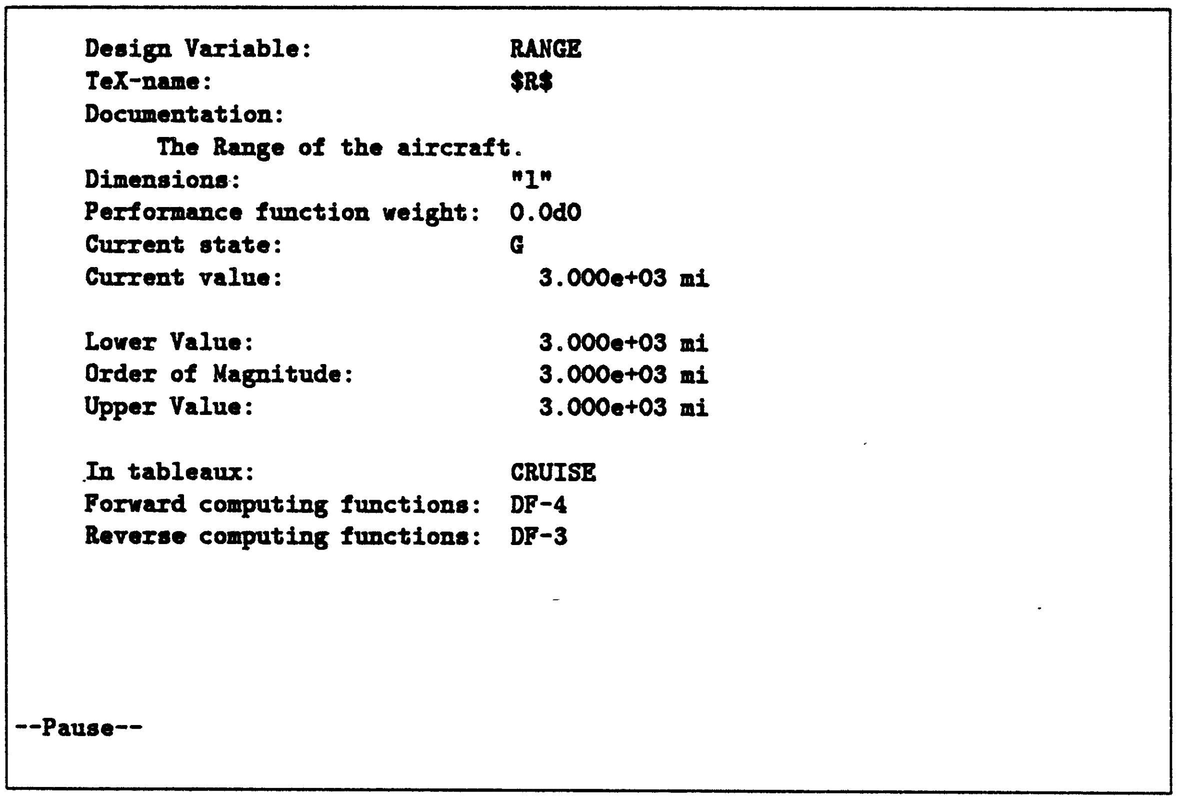
To change a design variable's state from G to I, simply press the $\boxed{\text{PF3}}$ key; this is referred to as freezing a design variable. To change its state back to G, simply press the $\boxed{\text{PF2}}$ key; this is referred to as floating a design variable.

[3]Chapter 4 discusses what to do when everything is *not* going smoothly.



```
                              CRUISE

-->RANGE                    G    3.000e+03 mi
   CRUISE_VELOCITY          G  565.0        mph
   TSFC                     G  800.0  e-03 lbm lbf-1 hr-1
   LIFT-TO-DRAG_RAT         G   15.00
   GROSS_TAKE-OFF_W         G   15.00 e+03 lb
   MIN_LANDING_WEIG         G   11.00 e+03 lb
   TIME_ON_RESERVES         G  750.0  e-03 hr







 (PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

Figure 2.5: Tableau CRUISE.



```
     Design Variable:              RANGE
     TeX-name:                     $R$
     Documentation:
          The Range of the aircraft.
     Dimensions:                   "1"
     Performance function weight:  0.0d0
     Current state:                G
     Current value:                  3.000e+03 mi

     Lower Value:                    3.000e+03 mi
     Order of Magnitude:             3.000e+03 mi
     Upper Value:                    3.000e+03 mi

     In tableaux:                  CRUISE
     Forward computing functions:  DF-4
     Reverse computing functions:  DF-3




--Pause--
```

Figure 2.6: Information on RANGE.

Changing a Design Variable's Value

To change a design variable's value, type the new value, then press the $\boxed{\text{Return}}$ key. You will notice that the design variable's state will change as well, to I, since *Paper Airplane* assumes that all user-entered values are for knowns. If you want to provide *Paper Airplane* with a better trial value for an unknown, just press the $\boxed{\text{PF2}}$ key after entering the value.

Changing a Design Variable's Units

To change a design variable's units, press the $\boxed{-}$ (minus-sign) key on the keypad. *Paper Airplane* will then ask you for the new units, which you then supply. For example, to change the units of Range from standard miles to nautical miles (a more common unit of Range) you would do the following:

```
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:  [-]
Enter new units (or RETURN to abort):  kg
Units "kg" do not have dimensions "l".
Check spelling of your units.
Enter new units (or RETURN to abort):  NM
```

The value and units of "`RANGE`" will change from "`3.000e+03 mi`" to "`2.607e+03 NM`". *Paper Airplane* automatically updates the value to reflect the change in units. Note, that *Paper Airplane* also checks to make sure that the new units have the proper dimensions. Appendix D lists all the dimensions and units pre-defined by *Paper Airplane*. It also explains to the user how to define new units and new dimensions.

Moving on to Another Design Variable

There are three ways to move on to another design variable. Two are to use the arrow keys: the $\boxed{\Downarrow}$ key to step down the list of design variables, and the $\boxed{\Uparrow}$ key to step up. Both keys will cause a wrap-around if you pass an end of the list. The third way is to move directly to the design variable you want by pressing the $\boxed{1}$ key on the keyboard:

```
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:  [1]
Variable to visit:  TSFC [Return]
```

The tableau will be redrawn with the arrow marker positioned at "`TSFC`". This method also works for moving directly to a design variable contained in a different tableau.

Moving on to Another Tableau

There are four ways to move on to another tableau. Two are to use the arrow keys: the $\boxed{\Rightarrow}$ key to step down the alphabetized list of design tableaux, and the $\boxed{\Leftarrow}$ key to step up. Again, both keys will cause a wrap-around if you pass an end of the list. The third way is when you move directly to a design variable contained in a different tableau (see previous description); and the fourth way is to move directly to the tableau you want by pressing the $\boxed{2}$ key on the keyboard:

```
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   [2]
Tableau to visit:  AERO [Return]
```

The AERODYNAMICS tableau will be drawn with the arrow marker positioned at the first design variable. This tableau is shown in Figure 2.7.

Reviewing the Keypad Options

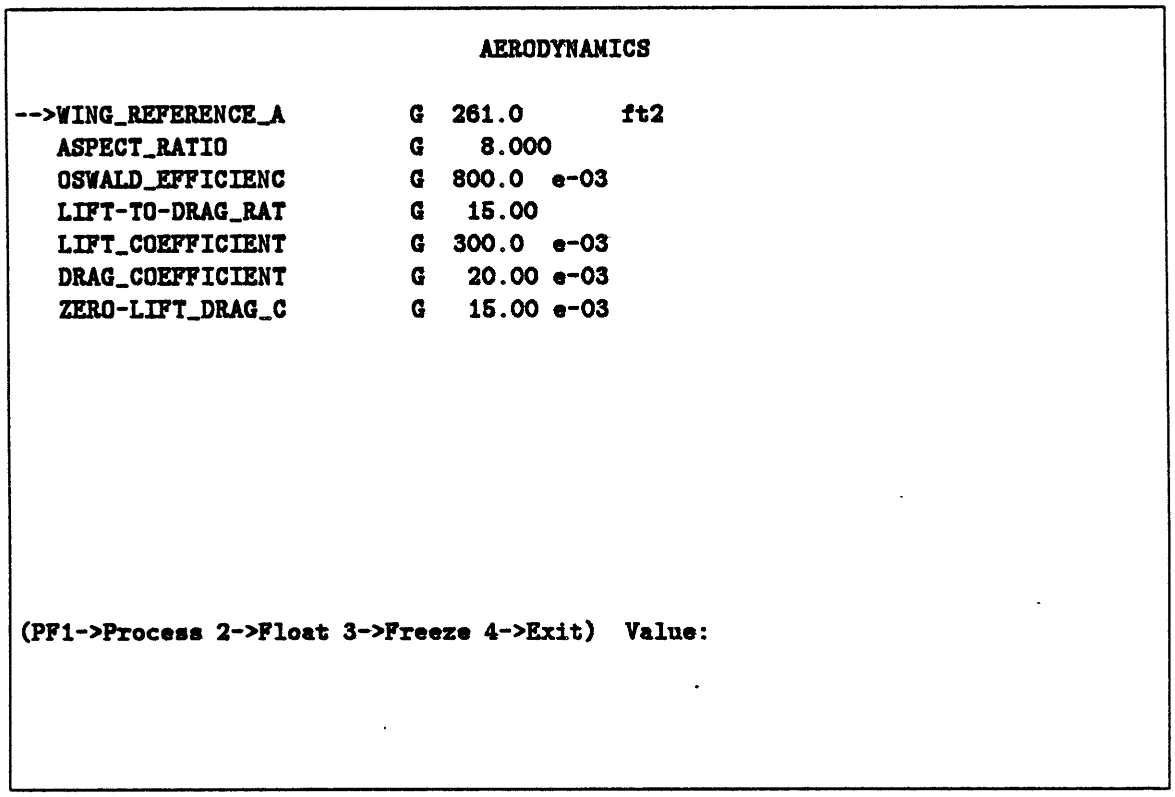
To review these and other keypad options, press the $\boxed{0}$ key on the keypad. The screen will be replaced with Figure 2.8. For now, please ignore the keys that were not described above.

You now know enough to make the changes to the design set necessary before you can proceed to the next step, processing it. Since the unknowns and knowns have already been selected (page **2-11**), you can now assign them values. The default values that appear in the tableaux were taken from an early paper on the *AM410-Laser*. The upper and lower limits were based upon the range of such values for existing executive transport jets from Nicolai [12]. Design requirements include a Range of 3000 standard miles, a Cruise Velocity of 565 mph, and a Payload Weight of 2200 pounds. Of the rest, the more specific values will be rounded to better starting guesses; the less specific ones will be left as is.

The next three figures, 2.9, 2.10, and 2.11, show the three tableaux after all the necessary changes have been made — after the design set has been initialized. You should be able to make those changes just by comparing the figures with what you see on the screen; however, if you have trouble, you can reference the text below. It is a list of the author's keystrokes and *Paper Airplane*'s responses. Although, in normal practice, many commands are used much more than others, the author's changes were designed to give the user a feel for *all* of the commands.

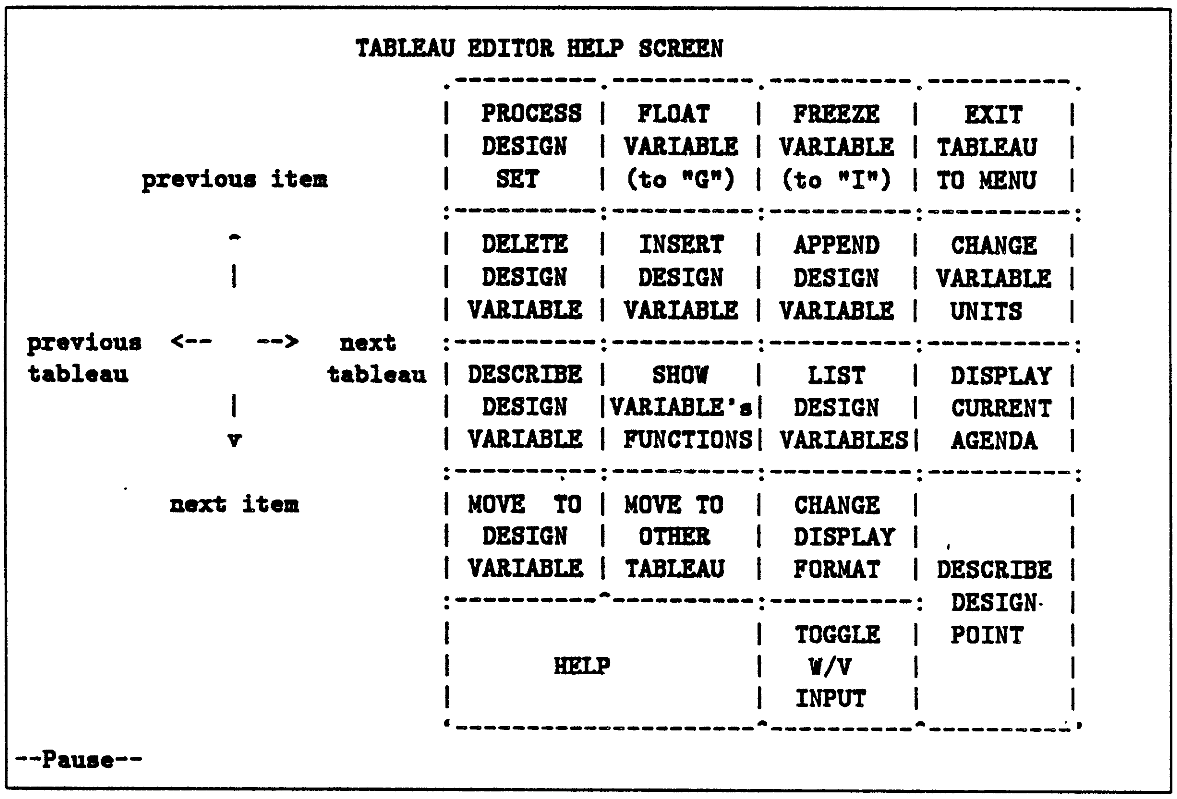
The changes made to the AERODYNAMICS tableau are as follows:

```
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   250 [Return]
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   [⇓]
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   [PF3]
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   [⇓]
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   [PF3]
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   [⇓]
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   12 [Return]
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   [PF2]
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   [1]
Variable to visit:  ZERO [Return]
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   [PF3]
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```



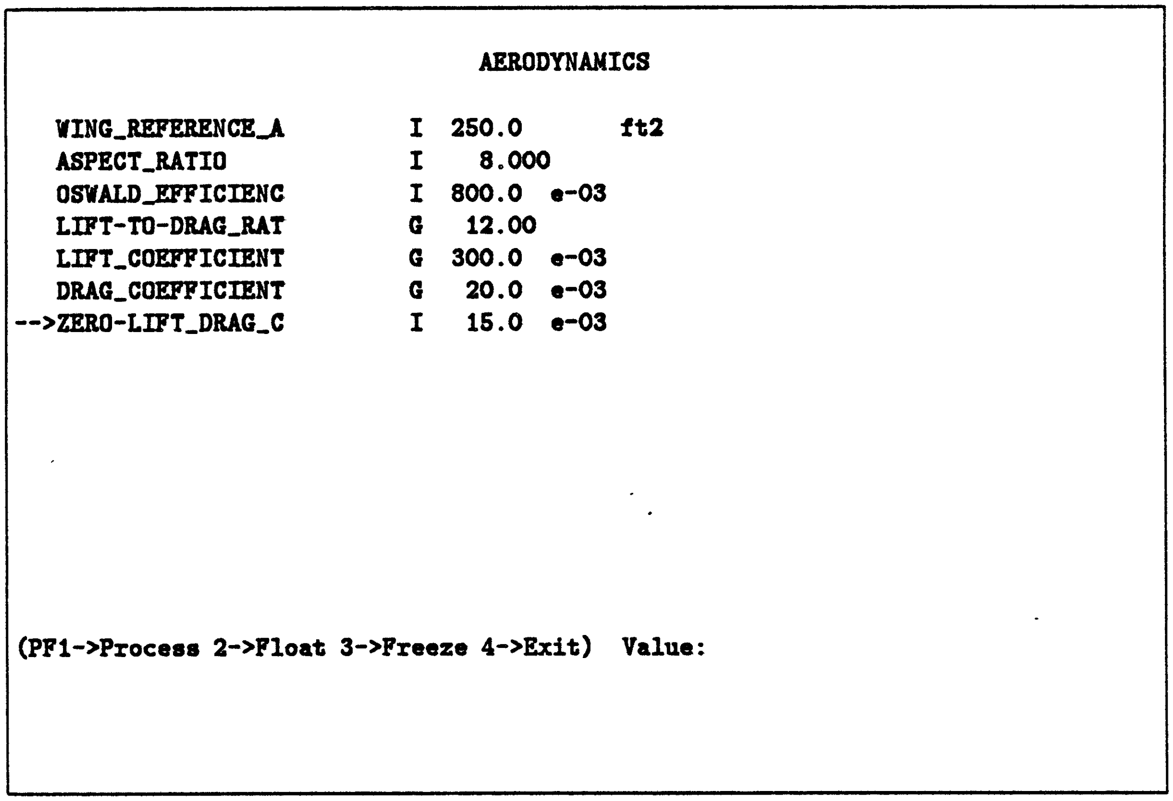
```
                               AERODYNAMICS

-->WING_REFERENCE_A         G  261.0       ft2
   ASPECT_RATIO             G    8.000
   OSWALD_EFFICIENC         G  800.0  e-03
   LIFT-TO-DRAG_RAT         G   15.00
   LIFT_COEFFICIENT         G  300.0  e-03
   DRAG_COEFFICIENT         G   20.00 e-03
   ZERO-LIFT_DRAG_C         G   15.00 e-03









(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

Figure 2.7: Tableau AERODYNAMICS.



```
                      TABLEAU EDITOR HELP SCREEN
                                .----------.----------.----------.----------.
                                | PROCESS  |  FLOAT   |  FREEZE  |   EXIT   |
                                |  DESIGN  | VARIABLE | VARIABLE | TABLEAU  |
        previous item           |   SET    | (to "G") | (to "I") | TO MENU  |
                                :----------:----------:----------:----------:
                 ^              |  DELETE  |  INSERT  |  APPEND  |  CHANGE  |
                 |              |  DESIGN  |  DESIGN  |  DESIGN  | VARIABLE |
                 |              | VARIABLE | VARIABLE | VARIABLE |  UNITS   |
previous  <--       -->  next   :----------:----------:----------:----------:
tableau               tableau   | DESCRIBE |   SHOW   |   LIST   | DISPLAY  |
                 |              |  DESIGN  |VARIABLE's|  DESIGN  | CURRENT  |
                 v              | VARIABLE | FUNCTIONS| VARIABLES|  AGENDA  |
                                :----------:----------:----------:----------:
          next item             | MOVE  TO | MOVE TO  |  CHANGE  |          |
                                |  DESIGN  |  OTHER   | DISPLAY  |          |
                                | VARIABLE | TABLEAU  |  FORMAT  | DESCRIBE |
                                :----------^----------:----------:  DESIGN  |
                                |                     |  TOGGLE  |  POINT   |
                                |        HELP         |   W/V    |          |
                                |                     |  INPUT   |          |
                                '---------------------^----------^----------'
--Pause--
```

Figure 2.8: Screen review of the keypad choices.



```
                              AERODYNAMICS

   WING_REFERENCE_A        I  250.0        ft2
   ASPECT_RATIO            I    8.000
   OSWALD_EFFICIENC        I  800.0  e-03
   LIFT-TO-DRAG_RAT        G   12.00
   LIFT_COEFFICIENT        G  300.0  e-03
   DRAG_COEFFICIENT        G   20.0  e-03
-->ZERO-LIFT_DRAG_C        I   15.0  e-03





(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

Figure 2.9: Tableau AERODYNAMICS after initialization.

The changes made to the WEIGHTS tableau are as follows:

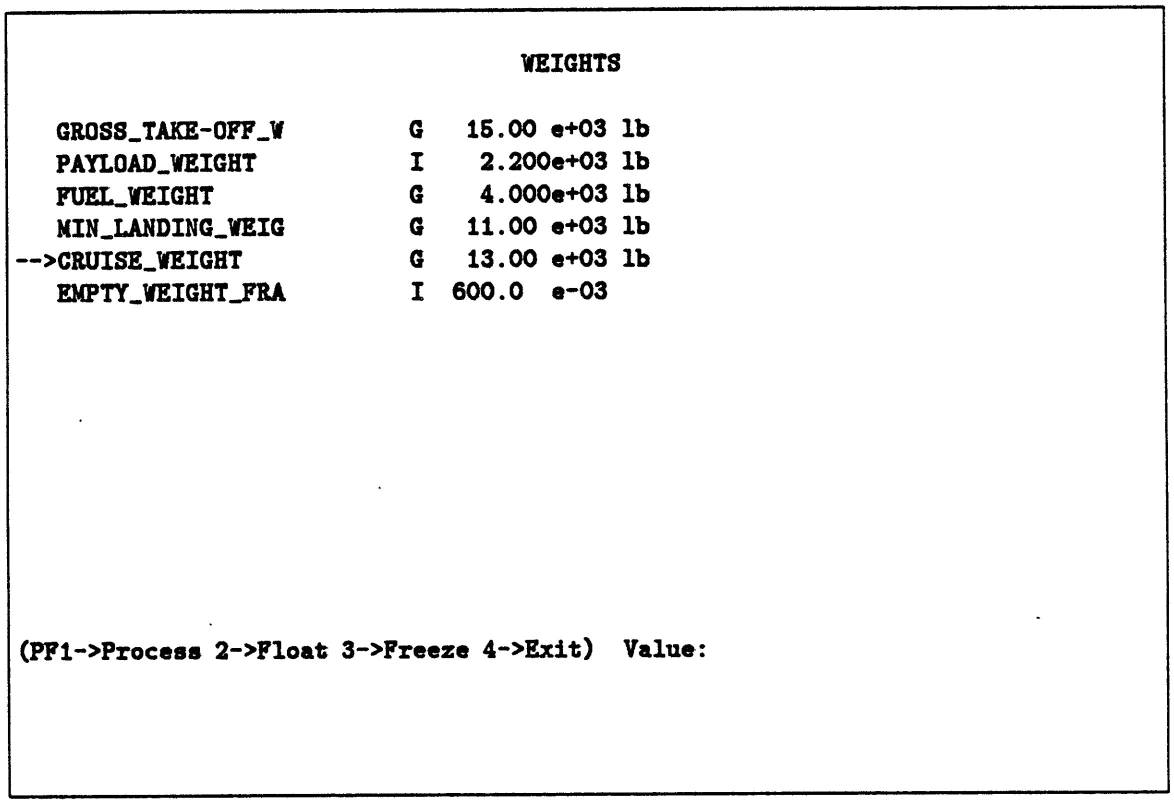
```
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   1
Variable to visit:  GROSS Return
Variable to visit:   GROSS_TAKE-OFF_WEIGHT
Visit GROSS_TAKE-OFF_WEIGHT in which tableau? ?
CRUISE  WEIGHTS
Visit GROSS_TAKE-OFF_WEIGHT in which tableau? W Return
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   ⇓
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   PF3
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   ⇑
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   ⇑
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   PF3
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   ⇑
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   13000 Return
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   PF2
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

The changes made to the CRUISE tableau are as follows:

```
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   ⇐
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   -
Enter display units for RANGE (or PF-4) to abort): mi Return
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   PF3
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   ⇓
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   ⇓
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   PF3
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   1
Variable to visit:  TIME Return
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   1 Return
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

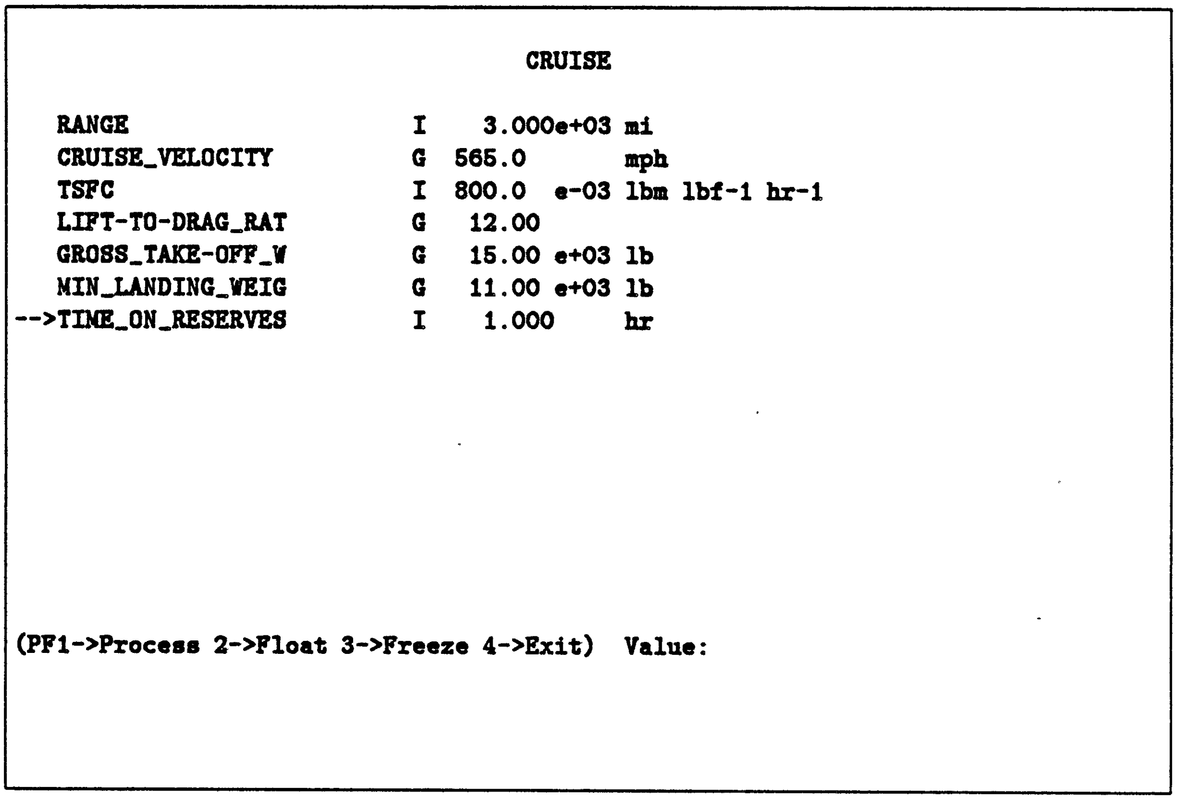
Before you press the $\boxed{\text{PF1}}$ key to tell *Paper Airplane* to process the design set, it is a good idea to press the $\boxed{\text{Enter}}$ key on the keypad first. This asks *Paper Airplane* to tell you if your design set is properly constrained; that is, if the number of unknowns matches the number of values computed by all the design functions:

```
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:   Enter
Design LASER has 9 base variables, 8 derived variables,
and 7 functions computing 7 variables.  The design path
is underconstrained by 1 derived variable.
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```



```
                              WEIGHTS

   GROSS_TAKE-OFF_W        G   15.00 e+03 lb
   PAYLOAD_WEIGHT          I    2.200e+03 lb
   FUEL_WEIGHT             G    4.000e+03 lb
   MIN_LANDING_WEIG        G   11.00 e+03 lb
-->CRUISE_WEIGHT           G   13.00 e+03 lb
   EMPTY_WEIGHT_FRA        I  600.0  e-03





(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

Figure 2.10: Tableau WEIGHTS after initialization.



```
                                     CRUISE

   RANGE                      I    3.000e+03 mi
   CRUISE_VELOCITY            G  565.0        mph
   TSFC                       I  800.0  e-03 lbm lbf-1 hr-1
   LIFT-TO-DRAG_RAT           G   12.00
   GROSS_TAKE-OFF_W           G   15.00 e+03 lb
   MIN_LANDING_WEIG           G   11.00 e+03 lb
-->TIME_ON_RESERVES           I    1.000      hr









(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

Figure 2.11: Tableau CRUISE after initialization.

Now you know why it is a good idea. *Paper Airplane* has discovered that the design set is underconstrained; that there is 1 more unknown ("derived" variable) than values computed by all the design functions. Since *Paper Airplane* cannot solve a design problem that is underconstrained, you will have to change one more unknown into a known. Quick analysis reveals that the Cruise Velocity design variable is the obvious choice. It is a design requirement, but was accidentally skipped over. You merely have to change its state to I and, then, you can see if your design is properly constrained:

```
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:  ⇓
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:  ⇓
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:  PF3
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:  Enter
Design LASER has 10 base variables, 7 derived variables,
and 7 functions computing 7 variables.  The design path
is perfectly constrained.
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

The design is now properly constrained. Here's where the fun begins. All you have to do now is to press the $\boxed{\text{PF1}}$ key and let *Paper Airplane* go to work to solve for the unknowns:

```
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:  PF1
Building agenda ...
      ⋮
Building agenda ...  Agenda construction completed.
Processing forced path ...
Processing FUEL_WEIGHT/DRAG_COEFFICIENT loop .
      ⋮
Processing FUEL_WEIGHT/DRAG_COEFFICIENT loop ....
(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

If you examine all three tableaux, they will appear as shown in Figures 2.12, 2.13, and 2.14. You will notice that all the unknowns are now knowns (computed-values) since their states have been changed from G to C. Their values have been changed as well, some by too much (as reflected in the warning messages for Gross Take-off Weight and Minimum Landing Weight). Do not be alarmed if your computed values differ slightly from those shown in the figures. *Paper Airplane* convergence criteria defaults its system epsilon to 0.1%, so it is not your fault if your values differ from the figures' beyond the third significant digit.

If you want to examine all of your design variables at the same time, you can do so by pressing the $\boxed{6}$ key on the keypad in any tableau. This will give you an alphabetized table of all design variables in the design set, such as the one shown in Figure 2.15. The +'s (plus signs) in the first column mark those design variables which are the computed variables of design functions, while the T's in the second column mark those design

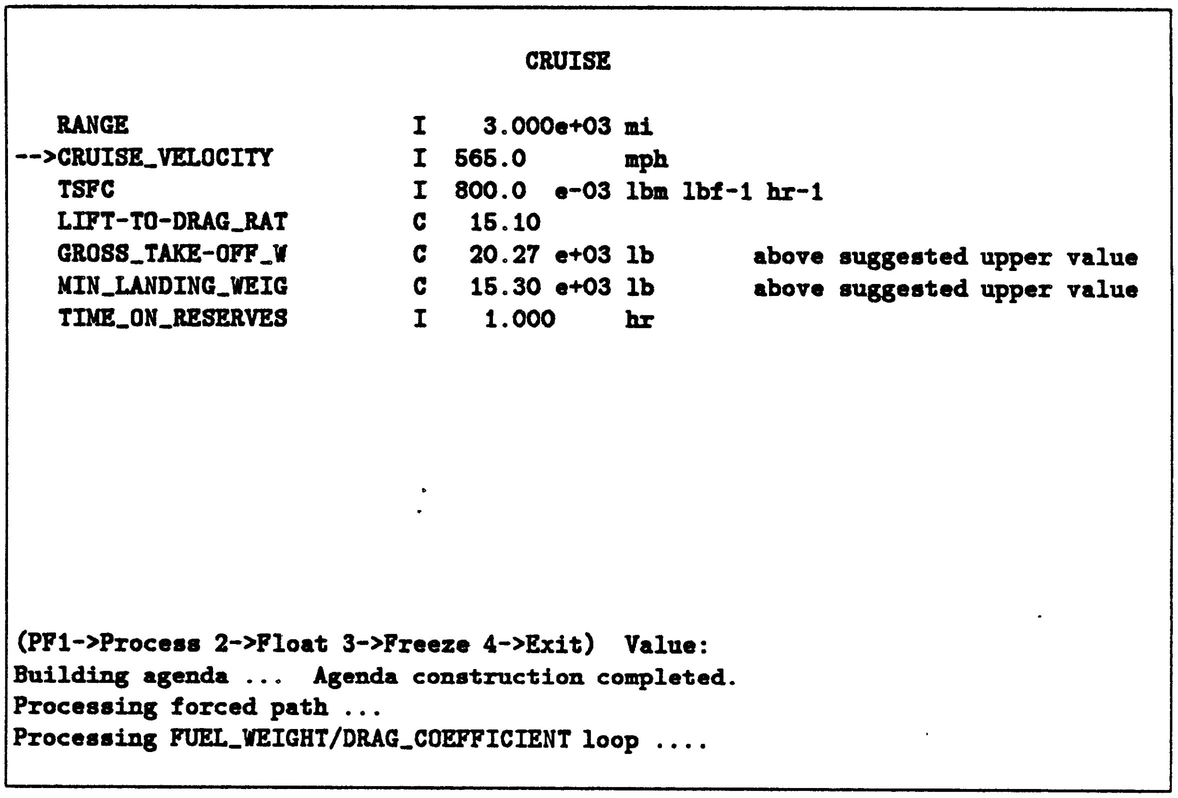
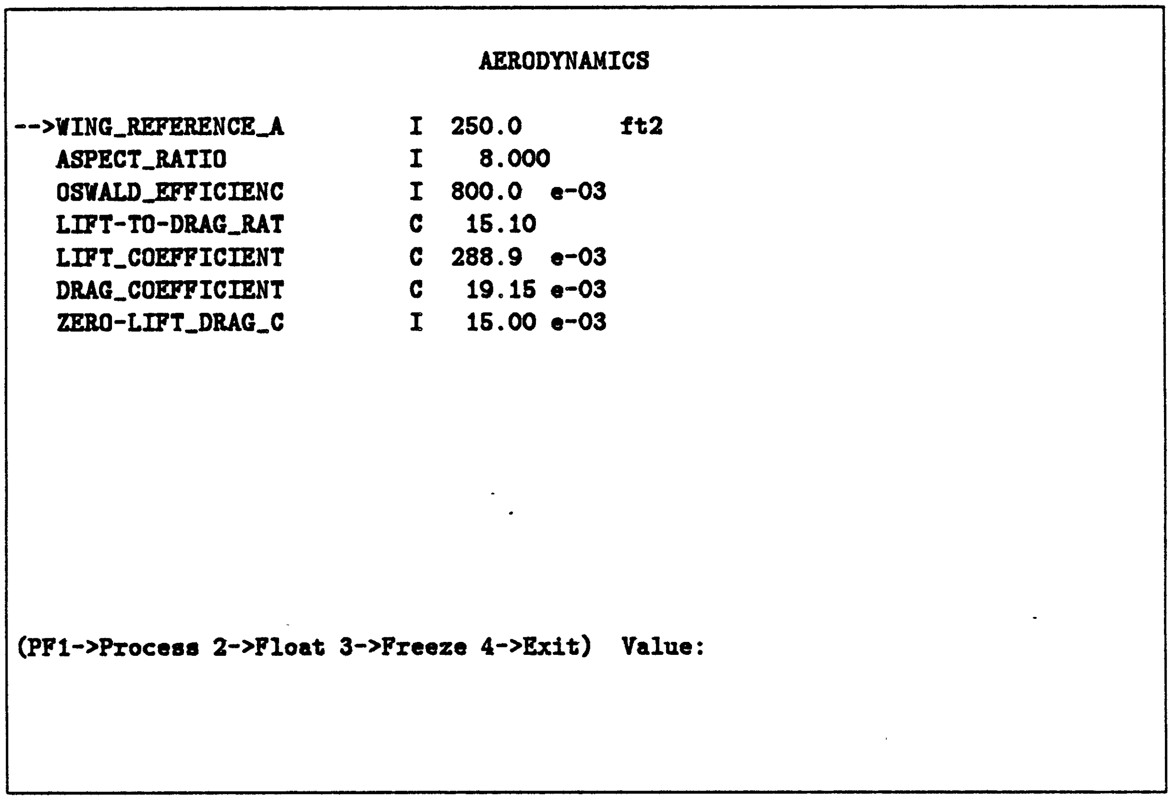
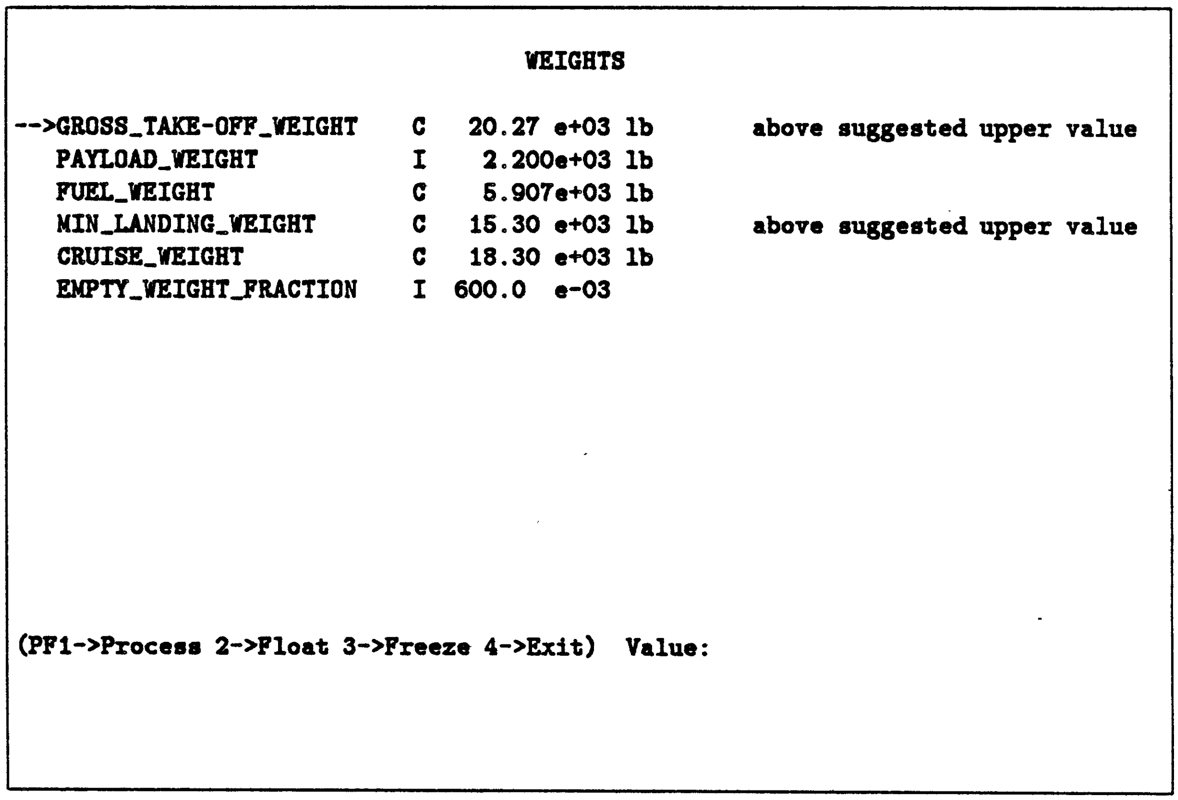

```
                                  CRUISE

   RANGE                      I    3.000e+03 mi
-->CRUISE_VELOCITY            I  565.0       mph
   TSFC                       I  800.0  e-03 lbm lbf-1 hr-1
   LIFT-TO-DRAG_RAT           C   15.10
   GROSS_TAKE-OFF_W           C   20.27 e+03 lb       above suggested upper value
   MIN_LANDING_WEIG           C   15.30 e+03 lb       above suggested upper value
   TIME_ON_RESERVES           I    1.000     hr






(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
Building agenda ...  Agenda construction completed.
Processing forced path ...
Processing FUEL_WEIGHT/DRAG_COEFFICIENT loop ....
```

Figure 2.12: Tableau CRUISE after processing.



```
                              AERODYNAMICS

-->WING_REFERENCE_A          I  250.0        ft2
   ASPECT_RATIO              I    8.000
   OSWALD_EFFICIENC          I  800.0  e-03
   LIFT-TO-DRAG_RAT          C   15.10
   LIFT_COEFFICIENT          C  288.9  e-03
   DRAG_COEFFICIENT          C   19.15 e-03
   ZERO-LIFT_DRAG_C          I   15.00 e-03








(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

Figure 2.13: Tableau AERODYNAMICS after processing.



```
                                    WEIGHTS

-->GROSS_TAKE-OFF_WEIGHT    C   20.27 e+03 lb       above suggested upper value
   PAYLOAD_WEIGHT           I    2.200e+03 lb
   FUEL_WEIGHT              C    5.907e+03 lb
   MIN_LANDING_WEIGHT       C   15.30 e+03 lb       above suggested upper value
   CRUISE_WEIGHT            C   18.30 e+03 lb
   EMPTY_WEIGHT_FRACTION    I  600.0  e-03






(PF1->Process 2->Float 3->Freeze 4->Exit)  Value:
```

Figure 2.14: Tableau WEIGHTS after processing.

variables which are displayed in one or more tableaux. The fourth column, which contains the weights on the values of the design variables, should be blank as this space is merely being reserved for the future. The last column, which contains the verbal number of incompatibilities each design variable has, should be blank as well; however, it would not be blank if *Paper Airplane* had problems processing the design set. Like the design variable information screen, you return to the tableau by pressing any key.

By now, you should be able to load a source file into a design set, prepare it for processing, and have *Paper Airplane* process it. If you wish to continue exercising the LASER design set, then keep reading; if you do not, then skip to the next section to learn what you may do with your design set now that it has been processed.

2.3.5 Further Exercising of the Design Set

In this section, you will continue exercising the LASER design set in an effort to bring the weight of the aircraft down. Since one of an aerospace engineer's major goals is to reduce weight, the design set will have to be reprocessed. The next attempt will be to change the value of a known; this way *Paper Airplane* will not have to recompute the computational path (agenda).[4] For the rest of this exercise, you will not be shown the author's keystrokes (since you should be able to make the changes on your own); nor will you be shown any more tableau figures, just table figures, like Figure 2.15.

One way to reduce aircraft weight is to reduce fuel weight (since it is such a large percentage of the total weight), and one way to reduce fuel weight is to reduce the amount of time needed on fuel reserves. Since FAA regulations only require 45 minutes of reserve fuel, you can lower "`TIME_ON_RESERVES`" back to 0.75 hours, and then press $\boxed{\text{PF1}}$ again. Notice that this time, *Paper Airplane* did not print out "`Building agenda`" since it already knew the computational path. After processing has completed, you will notice that the warning messages on the weights have gone, that their values have dropped back within design limits — "`GROSS_TAKE-OFF_WEIGHT`" is down to about 19,340 pounds — however, that should not stop you from trying to reduce weight further.

Another way to lower fuel weight is to use a more fuel efficient engine, which you can get by lowering its Thrust Specific Fuel Consumption. Lower "`TSFC`" from 0.8 to 0.6 $\mathrm{lb\,lbf^{-1}\,hr^{-1}}$, then press the $\boxed{\text{PF1}}$ key. Indeed, "`GROSS_TAKE-OFF_WEIGHT`" is back down to its initial guess; however, this was done using technology that does not yet exist, as shown by the warning on "`TSFC`". If you press the $\boxed{\ 4\ }$ key and look at the information on "`TSFC`", you will find that 0.8 $\mathrm{lb\,lbf^{-1}\,hr^{-1}}$ is not only the order of magnitude, but the limits as well. This is because a certain type of engine was specified in the requirements; thus, you will have to change "`TSFC`" back to 0.8 $\mathrm{lb\,lbf^{-1}\,hr^{-1}}$.

Another way to lower fuel weight is to raise the Lift-to-Drag Ratio, which is closely related to the ratio of aircraft weight and engine thrust (which you now know is fixed). You *could* change the value of "`LIFT-TO-DRAG_RATIO`" directly, but that would cause problems; you would be ignoring the values of "`LIFT_COEFFICIENT`" and "`DRAG_COEFFICIENT`", and thus you would be *also* ignoring the values of the other four design variables in the

[4]Indeed, *Paper Airplane* remembers *every* computational path it creates for each design set during a session.

```
                            LIST OF DESIGN VARIABLES

  F T   Design Variable Name   Weight   State & Value     Units      Incomp's
  - -  ---------------------- ------  --------------- ----------- --------
    T  ASPECT_RATIO                    I      8.000
    T  CRUISE_VELOCITY                 I    565.0        mph
  + T  CRUISE_WEIGHT                   C     18.30 e+03 lb
  + T  DRAG_COEFFICIENT                C     19.15 e-03
    T  EMPTY_WEIGHT_FRACTION           I    600.0  e-03
    T  FUEL_WEIGHT                     C      5.907e+03 lb
  + T  GROSS_TAKE-OFF_WEIGHT           C     20.27 e+03 lb
  + T  LIFT-TO-DRAG_RATIO              C     15.10
  + T  LIFT_COEFFICIENT                C    288.9  e-03
  + T  MIN_LANDING_WEIGHT              C     15.30 e+03 lb
    T  OSWALD_EFFICIENCY               I    800.0  e-03
    T  PAYLOAD_WEIGHT                  I      2.200e+03 lb
  + T  RANGE                           I      3.000e+03 mi
    T  TIME_ON_RESERVES                I      1.000      hr
    T  TSFC                            I    800.0  e-03 lbm lbf-1 hr-1
    T  WING_REFERENCE_AREA             I    250.0        ft2
    T  ZERO-LIFT_DRAG_COEFF            I     15.00 e-03

--Pause--
```

Figure 2.15: Table of all design variables in design set LASER.

`AERODYNAMICS` tableau, the ones that are supposed to be the ***knowns***. A better way would be to change the value of one of the knowns.

Longer wings create more lift and usually have less drag. That is why gliders have them. Aspect Ratio is the design variable to change here. Change "`ASPECT_RATIO`" to 10, then process the design set. "`GROSS_TAKE-OFF_WEIGHT`" has been reduced another 800 pounds. Unfortunately, at the speeds executive transport aircraft want to fly, longer wings cause other problems not reflected in this design set (which is why fighter aircraft have short wings); therefore, set "`ASPECT_RATIO`" back to 8.

One last way to reduce weight is actually the simplest from the design standpoint, but the hardest from the technological one. That is to lower the empty (structural) weight of the aircraft. The Empty Weight Fraction design variable used to be considered a fixed quantity, but with the introduction of composite materials, it has become a key design variable. As a last attempt to lower the aircraft weight, change "`EMPTY_WEIGHT_FRACTION`" to 0.55, then process the design set again. "`GROSS_TAKE-OFF_WEIGHT`" will be greatly reduced from 19,340 pounds down to 16,410 pounds. Now you know why this design variable can be so important. A lighter empty aircraft requires less fuel to make the same journey, and thus makes for a much lighter fully-loaded one.

Figure 2.16 summarizes this section's changes to the design set. It is the table of design variables (the one you get by pressing the keypad $\boxed{6}$ key) as it should now appear. This marks the end of the exercise portion of the tutorial. The next section tells you what you can do with your design set now that it has been processed.

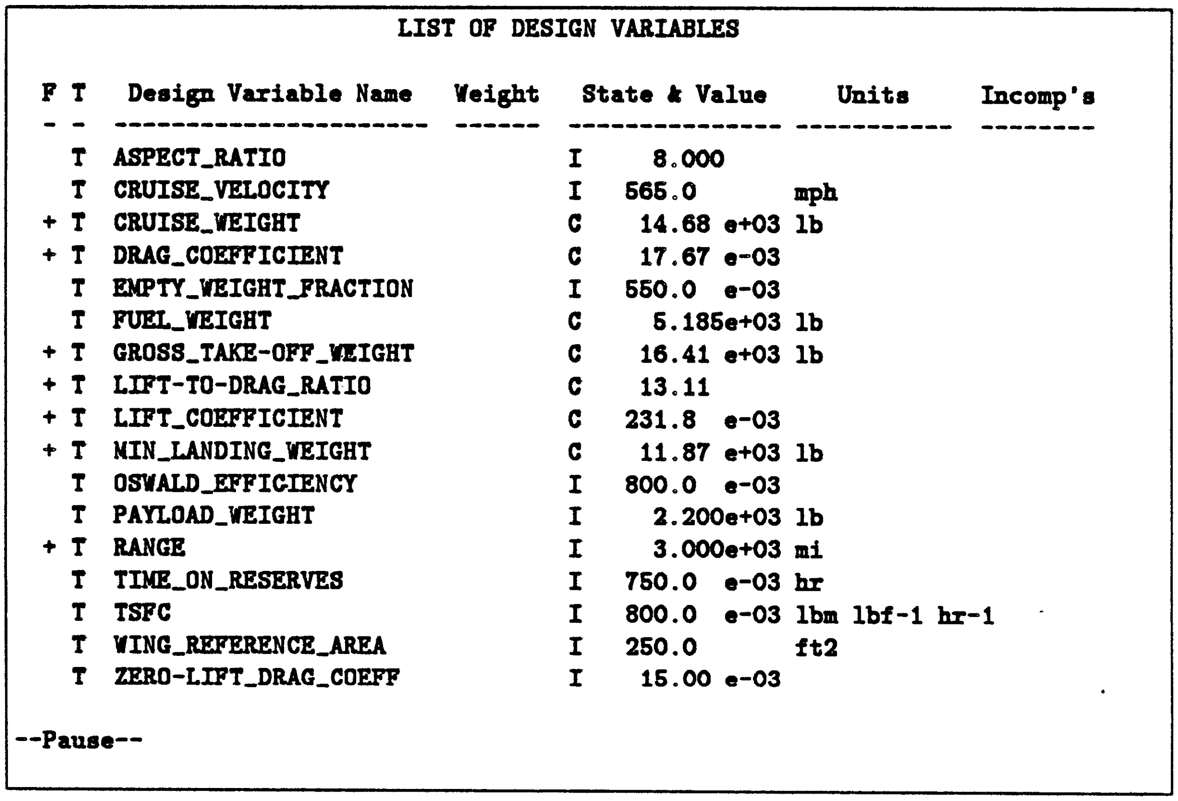
2.3.6 Finishing Up

This section tells you what you can do with your design set now that it has been processed. To begin, you will need to return to the main menu, which you can do easily by pressing the $\boxed{\text{PF4}}$ key.

If you examine the main menu, you will find that some commands produce information similar to that produced by some keypad keystrokes inside the tableaux environment. Command `[ 2]` will process the design set and return the time (in CPU seconds) *Paper Airplane* took to do it. Command `[11]` will "`Describe a design variable`" in the same manner that pressing the keypad $\boxed{4}$ key did. Likewise, command `[12]` will "`List all design variables`" in the same manner that pressing the keypad $\boxed{6}$ key did. Command `[13]` does the same thing as command `[11]` except that it only lists design variables with incompatible values (between design functions). Finally, command `[19]` will "`Describe design point`" in the same manner that pressing the keypad $\boxed{\text{Enter}}$ key did. See Appendix E for a complete description of these and other menu functions.

Now that you have a processed design set, you may want to keep something for your records. There are two ways to make hardcopy, both use the same command, `[22]`, "`Print design point table`". If your computer supports LaTeX, you can get high quality output by doing the following:

```
Enter command number: 22 [Return]
Do you wish to produce a TeX file or a standard data file? tex [Return]
```



```
                          LIST OF DESIGN VARIABLES

 F T  Design Variable Name  Weight  State & Value     Units      Incomp's
 - -  --------------------  ------  -------------  -----------  --------
   T  ASPECT_RATIO                  I     8.000
   T  CRUISE_VELOCITY               I   565.0       mph
 + T  CRUISE_WEIGHT                 C    14.68 e+03 lb
 + T  DRAG_COEFFICIENT              C    17.67 e-03
   T  EMPTY_WEIGHT_FRACTION         I   550.0  e-03
   T  FUEL_WEIGHT                   C     5.185e+03 lb
 + T  GROSS_TAKE-OFF_WEIGHT         C    16.41 e+03 lb
 + T  LIFT-TO-DRAG_RATIO            C    13.11
 + T  LIFT_COEFFICIENT              C   231.8  e-03
 + T  MIN_LANDING_WEIGHT            C    11.87 e+03 lb
   T  OSWALD_EFFICIENCY             I   800.0  e-03
   T  PAYLOAD_WEIGHT                I     2.200e+03 lb
 + T  RANGE                         I     3.000e+03 mi
   T  TIME_ON_RESERVES              I   750.0  e-03 hr
   T  TSFC                          I   800.0  e-03 lbm lbf-1 hr-1
   T  WING_REFERENCE_AREA           I   250.0       ft2
   T  ZERO-LIFT_DRAG_COEFF          I    15.00 e-03

--Pause--
```

Figure 2.16: LASER design variables at end of exercise.

```
Print file (WILBUR::PA$DISK:[EXAMPLES]POINT.TEX): sys$login:laser Return
Design point for "LASER" written to "WILBUR::SYS$LOGIN:LASER.TEX;" at
12:41:43.
Enter command number:
```

Normally you would just enter laser, but for the tutorial you should put any created files in your own directory ("SYS$LOGIN" is your home directory). You process this file just as you would any other LATEX file, and you will get results similar to Table 2.2. If your computer does not support LATEX, however, you can still get a normal text file version of the table (shown in Table 2.3) by doing the following:

```
Enter command number: 22 Return
Do you wish to produce a TeX file or a standard data file? dat Return
Print file (WILBUR::PA$DISK:[EXAMPLES]POINT.DAT): sys$login:laser Return
Design point for LASER written for "RML" at 12:45:51.
Enter command number:
```

Once you are finished with your design set, you should save it in a file as a design point. This allows you to pick up where you left off when you restart *Paper Airplane* some other time; otherwise, you would have to make the same changes all over again. Command [21] will "Save current design point" as follows:

```
Enter command number: 21 Return
Save file (WILBUR::PA$DISK:[EXAMPLES]POINT.SAV): sys$login:laser Return
File WILBUR::SYS$LOGIN:LASER.SAV;1 written out.
Variable values saved.
Enter command number:
```

To restore a design point, *you must load the original source file first.* Then use command [20] to "Restore old design point":

```
Enter command number: 20 Return
Read file (WILBUR::PA$DISK:[EXAMPLES]POINT.SAV): sys$login:laser Return
File WILBUR::SYS$LOGIN:LASER.SAV;1 read in.
Variable values restored.
Enter command number:
```

Finally, once you have finished using *Paper Airplane*, you exit it using command [24], "Exit PAPER AIRPLANE to LISP" and to exit NIL using the (quit) function:

```
Enter command number: 24 Return
      ⋮
THANK YOU FOR USING PAPER AIRPLANE
      ⋮
T
(quit)
```

oint for LASER, written for RML on Monday the twenty-third of March, 1987; 2:58:27 pm.

O.O.M.		Weight	State	Variable Name	Current Value		Units	Incomp's	Responsible Function
.000			I	AR	8.000				
65.0			I	V_{Cr}	565.0		mph		
3.30	$\times10^{3}$		C	W_{Cr}	14.68	$\times10^{3}$	lb		$W_{Cr} = f(W_{gto}, W_f)$
0.00	$\times10^{-3}$		C	C_D	17.67	$\times10^{-3}$			DF-5
3000			I	f_e	0.5500				
000.			C	W_f	5185.		lb		$C_D = C_{D_0} + \frac{C_L^2}{\pi AR\epsilon}$
5.00	$\times10^{3}$		C	W_{gto}	16.41	$\times10^{3}$	lb		$W_{gto} = \frac{W_p+W_f}{1-f_e}$
5.00			C	$^L/_D$	13.11				$R = \frac{V_{Cr}}{\mathrm{TSFC}} \frac{L}{D} \log\left(\frac{W_{gto}}{W_{l_{min}}}\right)$
3000			C	C_L	0.2318				$C_L = \frac{W_{Cr}}{1/2\rho V_{Cr}^2 S_{ref}}$
1.00	$\times10^{3}$		C	$W_{l_{min}}$	11.87	$\times10^{3}$	lb		$W_{l_{min}} = W_{gto} - W_f\left[\frac{R/V_{Cr}}{(R/V_{Cr})+T_{res}}\right]$
3000			I	ϵ	0.8000				
200.			I	W_p	2200.		lb		
000.			I	R	3000.		mi		
7500			I	T_{res}	0.7500		hr		
3000			I	TSFC	0.8000		$\mathrm{lb_m\ lb_f^{-1}\ hr^{-1}}$		
61.0			I	S_{ref}	250.0		ft^2		
5.00	$\times10^{-3}$		I	C_{D_0}	15.00	$\times10^{-3}$			

Table 2.2: Table of design variables produced by LATEX.

```
›oint for LASER printed for RML on Monday the twenty-third of March, 1987; 2:58:54 pm. ***

4.        Wt. State Variable Name            Value            Inc's    Resp. Function
-------------------------------------------------------------------------------------------------
)               I  ASPECT_RATIO          8.000
                I  CRUISE_VELOCITY      565.0       mph
 e+03           C  CRUISE_WEIGHT         14.68 e+03 lb                   DF-2
 e-03           C  DRAG_COEFFICIENT      17.67 e-03                      DF-5
 e-03           I  EMPTY_WEIGHT_FRACT   550.0  e-03
)e+03           C  FUEL_WEIGHT            5.185e+03 lb                   DF-7
 e+03           C  GROSS_TAKE-OFF_WEI    16.41 e+03 lb                   DF-1
                C  LIFT-TO-DRAG_RATIO    13.11                           DF-4
 e-03           C  LIFT_COEFFICIENT     231.8  e-03                      DF-6
 e+03           C  MIN_LANDING_WEIGHT    11.87 e+03 lb                   DF-3
 e-03           I  OSWALD_EFFICIENCY    800.0  e-03
)e+03           I  PAYLOAD_WEIGHT         2.200e+03 lb
)e+03           I  RANGE                  3.000e+03 mi
 e-03           I  TIME_ON_RESERVES     750.0  e-03 hr
 e-03           I  TSFC                 800.0  e-03 lbm lbf-1 hr-1
                I  WING_REFERENCE_ARE   250.0       ft2
 e-03           I  ZERO-LIFT_DRAG_COE    15.00 e-03
```

Table 2.3: Table of design variables in text file.

```
NIL Terminated

$
```

You have now completed your first lesson in *Paper Airplane*. You should now know the terminology and should be able to do the following:

- Start up *Paper Airplane*.
- Recover from dives into debug mode.
- Load a source file into a design set.
- Prepare a design set for processing, including the ability to:
  - Get information on a design variable.
  - Change a design variable's state.
  - Change a design variable's value.
  - Change a design variable's units.
  - Move on to another design variable or tableau.
  - Check to see if the design set is properly constrained.
- Process the design set.
- Examine the results.
- Make hardcopy of the results.
- Save the design point.
- Exit *Paper Airplane*.

You should also have a feel for what *Paper Airplane* can and cannot do at this time. The rest of the chapters will go into more detail on many of the subjects discussed in this chapter, especially Chapter 3, which will explain how to create source files.

Please remember that the *AM410-Laser* design was just *one* example of *Paper Airplane*'s applications. As long as you can define the functions (equations and/or codes) that describe a system, you can model and manipulate that system using *Paper Airplane*.

Chapter 3

Getting Started

In the last chapter, you learned how to use *Paper Airplane* once you had a source file. In this chapter, you will learn how a source file is made, a process discussed in the first three sections: Choosing the Design Set, Preparing the Design Set, and Creating the Source File. The last section will then ask you to edit the first tutorial source file to add some more features to the LASER design set.

3.1 Choosing the Design Set

Choosing the proper set of design variables and design functions is crucial to successfully finding the solution to a design problem. This process is often the most difficult and time-consuming, for it involves a lot of reasoning — on the engineering level, on the scientific level, and even on the mathematical level. This section details the rules and steps to follow for choosing a proper design set. It then closes with an examination of the author's reasoning for choosing the first LASER design set.

3.1.1 Rules for Choosing the Design Set

Rule Number 1

The most important rule for choosing the design set is to *keep the design set simple.* The design functions themselves can be complex, but you should keep the number of those design functions down to a manageable level — on the order of 10. Manufacturers are aware of this rule, that is why they separate the design process into a number of **design levels** ranging from "conceptual design" to "final design". When advancing levels, simpler design functions are replaced with more detailed ones, thus keeping the number of design functions small. Design variables that were once unknowns, meanwhile, have their values frozen (into knowns) or at least tightly constrained, and thus make room for new unknowns. Whether you are designing an aircraft, microchip, or sailboat, the rule is the same: *keep the design set simple.*

Rule Number 2

Another important rule for choosing the design set is to ***keep the design set focused.*** The design functions can solve for many unknowns, but there should only be ***one*** design goal. For example, in conceptual aircraft design, the goal is to minimize the aircraft's weight while meeting its performance requirements. Although many unknowns are solved for, such as the aircraft's geometric and aerodynamic characteristics, the chief design goal is to find the aircraft's total weight. By choosing a design set with one clear design goal in mind, you are forced to assemble only those design variables and design functions appropriate for the task at hand; thereby, creating a clear, concise package of interdependent design functions and the design variables those functions relate. With such a design set, you can be insured of not only solving your design problem, but solving for all the associated unknowns as well. This type of approach is designed to keep you from accidentally leaving out an important design principle by forcing you to think the design problem out thoroughly. Whether you are designing a diesel engine, wind mill, or communications satellite, the rule is the same: ***keep the design set focused.***

A fringe benefit of choosing a design set to solve only one design problem is, ironically, the ability to solve other design problems as well — using the same design set. For example, a conceptual spacecraft design set that is designed to minimize launch weight and solves for material costs along the way also could be used to minimize material costs and solve for launch weight along the way. In fact, once the proper design set is chosen, almost any design variable value can be optimized, or any appropriately weighted combination of them. This ability is one of the prime benefits that *Paper Airplane* gives the user over manual design.[1]

Summary of Rules

The two major rules to remember when choosing a design set are:

Rule 1: Keep the design set simple.

Rule 2: Keep the design set focused.

3.1.2 Steps for Choosing the Design Set

Although there are only two major rules, there are seven major steps to follow for choosing the design set. These steps are designed to help you think out the design problem thoroughly enough so that you can choose the proper design set.

Step Number 1

The first step is to decide what your level of expertise is in the field of the design problem you want to solve. In this regard, you can be either an expert, an intermediate, or a novice.

[1]This should not be taken that *Paper Airplane* can perform optimization. That ability is, unfortunately, still to be incorporated.

If you are an expert, choosing the proper design set would be a matter of probing your own mind. If you are an intermediate, choosing the proper design set would be a matter of consulting those experts in the field and/or consulting texts and technical papers written by them. If you are a novice, however, choosing the proper design set would be a matter of getting an expert or an intermediate in the field to choose the proper design set for you, or of taking a course on the subject to raise your level of expertise. (A current research project is the creation of a *Paper Airplane* "Librarian" to help users — even novices — choose the proper design set out of the *Paper Airplane* Library of design variables and design functions, which now exists in a limited capacity.)

Step Number 2

Assuming that you are either an expert or an intermediate, the second step is to clearly state the design goal in terms of key design parameters (both requirements and unknowns). These design parameters will become your initial set of design variables.

Step Number 3

The third step is to decide how much precision and detail you want in the solution to your design problem. Usually, the higher the precision, the more complex the design functions have to be provide it, and the longer it will take to find the solution.

Following the conventions of engineering procedure, it is usually best to first find a "quick and dirty" solution to put you in the vicinity of the precise one you are looking for, and then to close in on the precise one in a series of steps. In design terms, the first process is called "conceptual design," and the others are called "preliminary design," "advanced preliminary design," and "final design." It is recommended to the user that your first design set be at a conceptual design level so that you do not accidentally leave out an important design principle.

Step Number 4

The fourth step is to find the key functional relationships among the initial set of design variables. These relationships will become your initial set of design functions and can consist of engineering, scientific, and mathematical relationships as simple as the equation $F = ma$ to as complex as a computational fluid dynamics program. This all depends on the level of precision and detail you chose in Step 3.

Be careful, however, for *Paper Airplane* is very naive about engineering, scientific, and complex mathematical principles, so do not leave out fundamental relationships just because they seem obvious to you. To *Paper Airplane*, a design function is merely a black box with values going in and values coming out, and what happens inside the black box is a mystery.

Once you have completed Step 4, you will then have formed an initial design set. The next procedure requires you to iterate on Steps 5 and 6 until you decide that your design set will yield the degree of precision and detail you desire.

Step Numbers 5 and 6

The fifth step is to add to the design set new design variables which were introduced by being associated with design functions previously added, while the sixth step is to add to the design set new design functions to further relate the increased set of design variables.

If you get trapped in this cycle, it will probably be because you will add a design function to define a new design variable, then be forced to add the new design variables associated with this new design function, and then be forced to add new design functions to define *these* new design variables. The only way to get out of such a trap is to stop defining design variables and just plan on making some of them *assumed* constants. Making assumptions is a standard engineering and scientific practice. It is the assumptions that define the degree of precision and thereby the level of design.

For a conceptual design set, therefore, many assumptions will have to be made to keep the design set simple and focused. Once you ended the cycle of Steps 5 and 6, you will then have a complete design set — but is it a proper one?

Step Number 7

The seventh step is to review the entire design set in light of the two major rules. If your design set breaks Rule 1, you can simplify the design set by merging design functions that belong together. For example, the lift and drag coefficient equations "`DF-6`" and "`DF-7`" could have been merged into one aerodynamics function. (This will be done in Tutorial Number 2.) If your design set breaks Rule 2, you can correct this by throwing out all design functions and design variables that do not keep the design set focused.

After you have completed Step 7, you should then have a proper design set that is both simple and focused. You are then ready to write this information into a source file.

Summary of Steps

The seven major steps to follow when choosing a design set are:

Step 1: Decide your level of expertise in the field.

Step 2: State the design problem in terms of key design variables.

Step 3: Decide the level of precision and detail you require.

Step 4: Find the key design functions relating the design variables.

Step 5: Add any necessary design variables to the design set.

Step 6: Add any necessary design functions to the design set.

Step 7: Review entire design set in light of the two major rules.

3.1.3 The Initial AM410-Laser Design Set

As an example of how a design engineer chooses a design set, this section will follow the author's reasoning for choosing the 7 design functions and 17 design variables making up the LASER design set.

In the beginning, there is the idea. In this case, The idea was for an executive transport aircraft that would be more spacious on the inside, yet still fly as far and as fast as other existing executive transport aircraft. The idea is then composed into a list of design requirements. In the business world, the list of design requirements for an aircraft is almost as thick as the final document on its design! Since the author's design problem existed in the academic world, the list of design requirements was *substantially* smaller — only one page. The following list of requirements is quoted from the final paper on the *AM410-Laser* [6]:

<u>Mission Description</u>

Despite the advances in computer technology, tele-conferencing is not going to replace person-to-person meetings. Long distance may be the "next best thing to being there", but "being there" is still [the] best. Therefore, when tele-conferencing just won't do, executives are going to need to get to a meeting quickly. [And] because executives like to travel in style, standard-size business jets with cramped compartments are becoming less wanted. Thus, a high-speed medium-size executive transport is now required to take executives in style across the miles. Specific requirements are listed below.

<u>Mission Requirements</u>

Payload: Crew of two, eight passengers, plus baggage
Speed: Cruise at Mach 0.85 at minimum [altitude] of 35000 feet
Range: 3000 miles (with full payload and 45 minute reserve fuel)
Take-off roll: 4000 feet
Landing roll: 2500 feet

Aircraft should use [composite technology] where currently applicable (no creating new technology for this aircraft) to decrease aircraft weight and increase fuel efficiency. Aircraft speed should not be lowered for the sake of better fuel economy, though cruise altitude may be raised. [The aircraft must use the Garrett Airesearch TFE731-3B-100 turbofan engine.][2]

Immediately from this list of requirements come several design variables. The design set knowns include Payload Weight, Cruise Mach Number, Range, Take-off Roll, Landing Roll, and everything about the engines, such as Engine Weight and Thrust Specific Fuel Consumption. The design set unknowns include Aspect Ratio, Wing Reference Area, and

[2]This design requirement was left out by accident, mainly because it was the only turbofan engine the class had information on.

other geometric characteristics; Lift-to-Drag Ratio, Oswald Efficiency, and other aerodynamics characteristics; Gross Take-off Weight, Empty Weight Fraction, and other weight characteristics; Range, Landing Velocity, and other performance characteristics; Center of Gravity, Mean Aerodynamic Chord, and other stability and control characteristics; Chair Pitch, Window Height, and other interior characteristics; Fuselage Thickness, Wing Loading, and other structural characteristics; and many more characteristics, including avionics, electrical systems, flight service, landing systems, maintenance, and operation costs.

As you can plainly see from the different sizes of the lists above, there are far more unknowns than knowns, enough to overwhelm a *team* of design engineers, let alone *one*. This is where the expertise is needed to find the proper design set, to separate out a small group of knowns and unknowns from literally hundreds, and to find an equally small group of equations and/or codes to relate them. For the field of aircraft design, much of that expertise has been documented into two very useful books, Nicolai [12] and Torenbeek [15], and as well as others. Only *you* know if the same is true for the field of *your* design problem.

The 7 design functions and 17 design variables contained in the initial *AM410-Laser* design set were chosen to give the design engineer a general feeling for the weights and aerodynamics of the aircraft. The Bréguet Range Equation (BRE) is a relationship between the weight, engine, aerodynamic, and performance characteristics of the aircraft. Since the required performance and engine characteristics (Range, Cruise Velocity, and Thrust Specific Fuel Consumption) were given, the BRE was an obvious choice for a design function. This equation introduced some unknowns: Lift-to-Drag Ratio, Gross Take-off Weight, and Minimum Landing Weight. Each of these design variables could be evaluated by adding three more design functions, which are forms of their definitions rather than observed relationships, such as the BRE. These design functions added more unknowns to the design set: Lift and Drag Coefficients, Fuel Weight, Empty Weight Fraction, and Time Available on Fuel Reserves. The latter two values could be considered temporary knowns since there values are similar for all executive transport aircraft. The aerodynamic coefficients required design function definitions, which then added more unknowns to the design set: Aspect Ratio, Wing Reference Area, Oswald Efficiency, Zero-Lift Drag Coefficient, and Cruise Weight, all of which could be considered temporary knowns, with the exception of Cruise Weight, which required one more design function definition. Fuel Weight could be solved for by using the Gross Take-off Weight equation since the other design variables it involves, Payload Weight and now Empty Weight Fraction, are knowns; thus eliminating the need to add another design function for Fuel Weight. These design variables and design functions form the initial *AM410-Laser* design set.

Even in this small design set, important relationships exist between design variables, such as "the higher the Lift-to-Drag Ratio the lower the Gross Take-off Weight". It is this relationship and others that allow a few knowns to be seeded into a garden of unknowns, to take root and spread, and, finally, be harvested as a complete design. But just as a bad seed will not yield a good harvest, a poor initial design set will not yield a good design. Keep the design set simple and focused, especially if you are designing a revolutionary product or are using new technology. *Paper Airplane* will allow you to exercise design sets much faster than you could via a hand calculator or computer input/output files, thus you will have more time to think about the engineering aspects of your design so that you can produce a better product. And a better product is the goal of *every* design engineer.

3.2 Preparing the Design Set

Once you have chosen your design set, you need to prepare it for use with *Paper Airplane*. A *Paper Airplane* design set consists of a group of design functions which relate a group of design variables. It also consists of another group of functions called **auxiliary functions** which do *not* relate any design variables, but can be called by any design functions. This section discusses the rules and restrictions of these three groups according to the current version of *Paper Airplane*.

3.2.1 Design Variables

Design variables must be *scalar* quantities. Complex numbers, vectors, and arrays are not possible in their *original* forms. It is possible, however, to separate a complex number, vector, or array into its scalar elements and make these design variables. A design function could receive the individual elements, then would reassemble them back into their original form so it could perform its high-level math operations. This alternative is recommended for only those quantities that could be thought of as scalars, such as the components of a velocity vector, and not for those quantities that are best left tabularized, such as the points in a "Lift Coefficient versus Angle of Attack" plot. Tabular information is better served by auxiliary functions, which are explained later in this section.

3.2.2 Design Functions

Design functions must be written in COMMON LISP, whether they are single equations or complex functions. A way to avoid writing in COMMON LISP is to write all of your design functions, in the language of your own choosing, into separate external codes in, compile and link them, and call them from *Paper Airplane* using *Paper Airplane*'s External Code Interface (see Appendix C). This is recommended only for those design functions that are already written as external codes and for those that require frequent use of vector and array operations and/or complex mathematical functions, such as integration and differentiation. It is *not* recommended for those design functions that can be written as simple equations, groups of simple equations, or simple logical functions. These simpler design functions can easily be written in COMMON LISP using the help available in Appendix A.

All inputs to and outputs from a design function must be design variables, thus they must be scalar quantities. Input variable values should not be changed, for only the final values of the output variables will be returned to *Paper Airplane*, not the final values of the input variables. The only way an input variable value could be modified is to have the same input variable be also an output variable. This cannot be done, however, because *Paper Airplane* would have no idea how to solve such a design function (it would always be over-constrained or under-constrained).

In the case where an input variable value *must* be modified, a new design variable should be added to the design set to become the output variable, and a new design function should be added to the design set to equate the value of the output variable to the value of the input variable. The steps for such a procedure are as follows:

1. Add another design variable to the design set with the same name as the design variable you wish to copy — except add the prefix "OLD-" to it. For example, if your design variable's name was "VAR," the added design variable would be named "OLD-VAR."
2. Make "OLD-VAR" the input variable to the design function that is doing the modification and "VAR" the output variable.
3. Create a new design function with "VAR" as the input variable *and* as the body of the function, and "OLD-VAR" as the output variable.
4. All other design functions should reference "VAR" and not "OLD-VAR."
5. Both "OLD-VAR" and "VAR" must be unknowns, thus their states must be G. If you have an initial guess for the design variable, it should be given to "OLD-VAR."

Design functions that require extra variables to hold values of intermediate steps should declare them internally as **local variables**, while those that require extra variables to hold constants or tables of reference values should declare them externally as **global variables**. More will be said about local and global variables in the next section.

Design functions that return only tables of values and no scalar values should be written as auxiliary functions instead and called upon by other design functions that do return scalar values. An alternative to this approach is to add a design variable to act as a flag to signal the rest of the design set when a table has been produced. The value of the flag would be unimportant since *Paper Airplane* would only check its state; therefore you could assign the flag to any value you wished, such as the table size or dimensions.

3.2.3 Auxiliary Functions

This is not to belittle the importance of auxiliary functions, however. Auxiliary functions are very useful for performing generic sub-tasks that one or more design functions have need of. Such sub-tasks include table look-ups, external code executions, and high-level mathematics operations, which include complex number and matrix operations, numerical integration, and numerical differentiation. Most scalar mathematics functions, such as trigonometric functions, are already available in COMMON LISP.

Auxiliary functions cannot operate on design variables directly. If an auxiliary function needs the value of certain design variables, it must be passed those values from the design function into dummy arguments in an argument list. Like design functions, auxiliary functions must be written in COMMON LISP.

Table look-ups and external code executions are required so often in engineering that they have been incorporated into special *Paper Airplane*-internal auxiliary functions called **utility functions**. Utility functions can be called from any design function and even from any auxiliary function. (Any auxiliary function can be called from another auxiliary function.)

The main table look-up function is called simply "`table-lookup`" and the main External Code Interface function is called simply "`run-program`." "`table-lookup`" is an n-dimensional table look-up function that calls itself recursively. For slightly more efficient processing, there is also a one-dimensional table look-up function called "`1d-table-lookup`," and a two-dimensional table look-up function called "`2d-table-lookup`." Appendix C instructs the user how to use all of these utility functions.

The next section instructs the user how to write design variables, design functions, and auxiliary functions into a source file.

3.3 Creating the Source File

Once you have a proper design set, the next step is to encode this information into a source file. This section will describe the Basic Format of the source file, then describe each of its major components: Design Variables, Auxiliary Functions, Design Functions, and Design Sets.

3.3.1 Basic Format

Figure 3.1 shows the basic format of the source file. The solid boxes surround source file requirements to produce a *Paper Airplane* design set; the dashed boxes surround source file options. Like a PASCAL program, all variables must be declared before they can be used, thus global variables and design variables must precede auxiliary functions, design functions, and design sets. Auxiliary functions must precede the design functions that call them; however, it is easier to remember that all auxiliary functions should be declared before any design functions. Design sets must be declared last. Because of this grouping structure, a large source file can be easily separated into smaller modules, such as one for global and design variables, one for auxiliary and design functions, and one for design sets. You must load the modules into *Paper Airplane* in the stated order, however, for a design set to be assembled properly.

It is a good idea to create a header out of comments[3] to specify the name of the design product, the current design level, and the name of the design set. Figure 3.2 shows the author's convention for specifying the header in the source file. Also shown is the declaration of global variables.

A global variable declaration is the one piece of code written *exactly* the way it is written in COMMON LISP. The declaration format is as follows:

(defvar *global-name global-value*) ; *global-units*

[3]A comment is any line in the source file starting with a semicolon.

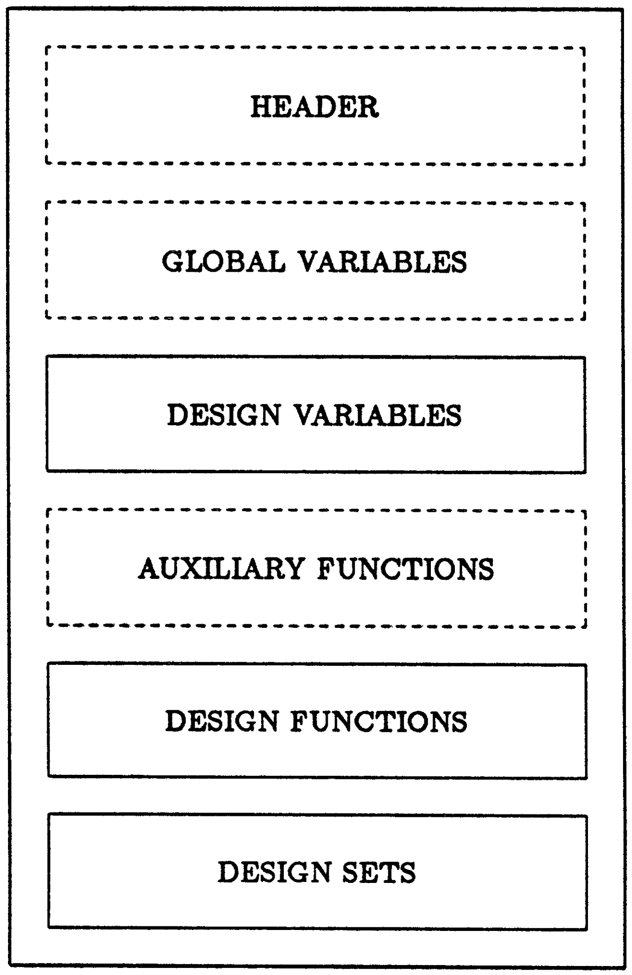


Figure 3.1: Basic format of the source file.

```
;;;; -*-  mode:pa ; package:pa; readtable:cl; base:10  -*- ;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                            ;;
;;         AM410-Laser Executive Transport Aircraft           ;;
;;                             a                              ;;
;;          design experiment using Paper Airplane            ;;
;;                          and a                             ;;
;;  running example for the Paper Airplane User's Manual     ;;
;;                                                            ;;
;;                   Conceptual Design Level                  ;;
;;               Tutorial Design Set -- Number 1              ;;
;;                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;; GLOBAL VARIABLES ;;;;;;;;;;;;;;;;;;;

(defvar *AIR_DENSITY* 0.000738)      ; slugs ft-3
(defvar *PI-D*        3.14159265)    ;


;;;;;;;;;;;;;;;;;;;;;; DESIGN VARIABLES ;;;;;;;;;;;;;;;;;;;
```

Figure 3.2: Header and constant declaration in the source file.

where

global-name is the name of the global variable, such as "`*AIR_DENSITY*`". The asterisks around the name are COMMON LISP requirements to identify any global variable.

global-value is the value of the global variable, such as "`0.000738`".

global-units are the units of the global variable corresponding to the value, such as "`slugs ft-3`". If the global variable is dimensionless, just leave a blank.

3.3.2 Design Variable Declaration

Following the declaration of global variables should come the declaration of your design variables (as shown in Figure 3.3. The declaration format is as follows:

```
(pa-defvar dv-name
          :category dv-cat-list
[         :TeX-name dv-tex-symbol ]
          :order-of-magnitude dv-def-value
          :lower-value dv-lower-value
          :upper-value dv-upper-value
          :dimensions dv-dimensions
          :units dv-def-units
[         :documentation dv-description ] )
```

where

dv-name is the name of the design variable seen in tableaux and tables, such as "`RANGE`".

dv-cat-list is the list of the names of the library category and optional sub-categories you wish to place the design variable under, surrounded by parentheses.

dv-tex-symbol is the LATEX representation of the design variable name seen in hardcopy output like Table 2.2, such as "`"$R$"`". The LATEX name is a string, thus it must be surrounded by double quotes. If your system does not support LATEX, do not worry, the LATEX name is optional to the design variable declaration (note the thin square brackets). See Appendix B for instructions on how to represent design variables using LATEX.

dv-def-value is the default value (order of magnitude) of the design variable seen when you first load your source file into a design set. The numerical value is only required, not the units.

dv-lower-value is the suggested lower limit to the value of the design variable seen in information screens like Figure 2.6. The numerical value is only required, not the units.

```
;;;;;;;;;;;;;;;;;;;;;; DESIGN VARIABLES ;;;;;;;;;;;;;;;;;;;;

(pa-defvar LIFT-TO-DRAG_RATIO
           :category (AERODYNAMICS)
           :TeX-name "$^L/_D$"
           :order-of-magnitude 15
           :lower-value 10
           :upper-value 20
           :dimensions ""
           :documentation "The Lift-to-Drag Ratio of the aircraft at Cruise.")

(pa-defvar RANGE
           :category (PERFORMANCE CRUISE)
           :TeX-name "$R$"
           :order-of-magnitude 3000
           :lower-value 3000
           :upper-value 3000
           :dimensions "l"
           :units "mi"
           :documentation "The Range of the aircraft.")

(pa-defvar WING_REFERENCE_AREA
           :category (AERODYNAMICS)
```

Figure 3.3: Design variable declaration in the source file.

dv-upper-value is the suggested upper limit to the value of the design variable seen in information screens like Figure 2.6. The numerical value is only required, not the units.

dv-dimensions is the dimensions of the design variable seen in information screens like Figure 2.6, such as "`"1"`". Notice that dimensionless design variables, such as Lift-to-Drag Ratio, have dimensions of "`""`", a null string. Higher-order dimensions are represented by number corresponding to their order, such as "`"12"`" for Wing Reference Area. The dimensions of Air Density would be "`"m 1-3"`". The dimensions is a string, thus it must be surrounded by double quotes.

dv-units are the units corresponding to the default and limiting values you provide, such as "`"sm"`" for Range. The units are always checked by *Paper Airplane* to make sure that they agree with the design variable's dimensions. The units are a string, thus they must be surrounded by double quotes.

dv-description is a description or definition of the design variable seen in information screens like Figure 2.6, such as "`"The Range of the aircraft"`". The description is a string, thus it must be surrounded by double quotes. The description is optional to the design variable declaration, but is recommended.

When choosing the default and limiting values of a design variable, you should consider how each of those values would affect your design. As you know from the tutorial, *Paper Airplane* does not stop processing if a value goes beyond those bounds; it merely prints a warning message — as it should, since the limits are design limits, not physical ones.[4] Design variables whose values are requirements, such as Range, should have their limiting values be the same value as its order of magnitude.

3.3.3 Auxiliary Function Declaration

Following the declaration of design variables should come the declaration of any auxiliary functions you may have. The declaration format, which is almost *exactly* the same for LISP defuns, is as follows:

```
(paux-defun af-name
            :category af-cat-list
            :lambda-list af-arg-list
            :function-body af-body
[           :documentation af-description ] )
```

where

af-name is the name of the auxiliary function.

[4]*Paper Airplane* does not *search* beyond those bounds when it is looking for a solution, however; during iteration those bounds are treated as physical.

af-cat-list is the list of the names of the library category and optional sub-categories you wish to place the auxiliary function under, surrounded by parentheses.

af-arg-list is the list of arguments passed to the auxiliary function, surrounded by parentheses. If no arguments are to be passed; type empty parentheses "()" instead.

af-body is the function body of the auxiliary function, which is written exactly as it would be in a COMMON LISP defun. Please remember that the body cannot reference design variables directly. See Appendix A for a discussion of LISP programming.

af-description is a description or definition of the auxiliary function. The description is a string, thus it must be surrounded by double quotes. The description is optional to the auxiliary function declaration, but is recommended.

3.3.4 Design Function Declaration

Following the declaration of auxiliary functions should come the declaration of your design functions (as shown in Figure 3.4). The declaration format is as follows:

```
(pa-defun df-name
          :category df-cat-list
          :computed-variables df-output-list
          :input-variables df-input-list
          :function-body df-body
[         :TeX-name df-tex-symbol ]
[         :documentation df-description ] )
```

where

df-name is the name of the design function seen in information screens like Figure 2.6, such as "`DF-5`". If you prefer to use a more meaningful design function name, such as "`Lift-to-Drag_Ratio_Equation`", you can use the description option for an explanation of what the function does.

df-cat-list is the list of the names of the library category and optional sub-categories you wish to place the design function under, surrounded by parentheses.

df-output-list is the list of names and units of all the design variables (the order is not important) whose values are computed by the design function, surrounded by parenthesis. Each pair of name and units is surrounded by parentheses as well. The units can be any units you need to write the design function the simplest, as long as they agree with the design variable's dimensions; *Paper Airplane* will take care of all units conversions.

df-input-list is the list of names and units of all the design variables (the order is not important) whose values are required by the design function, surrounded by parenthesis. Each pair of name and units is surrounded by parentheses as well. The

```
;;;;;;;;;;;;;;;;;;;;; DESIGN FUNCTIONS ;;;;;;;;;;;;;;;;;;;;;

(pa-defun DF-5
          :category (AERODYNAMICS)
          :computed-variables ((LIFT-TO-DRAG_RATIO ""))
          :input-variables ((LIFT_COEFFICIENT "")
                            (DRAG_COEFFICIENT ""))
          :function-body (/ LIFT_COEFFICIENT DRAG_COEFFICIENT)
          :TeX-name "DF-5"
          :documentation "Lift-to-Drag Ratio Equation.")

(pa-defun DF-6
          :category (PERFORMANCE CRUISE)
          :computed-variables ((LIFT_COEFFICIENT ""))
          :input-variables ((CRUISE_WEIGHT "lb")
                            (FUEL_WEIGHT "lb")
                            (CRUISE_VELOCITY "ft s-1")
                            (WING_REFERENCE_AREA "ft2"))
          :function-body (/ CRUISE_WEIGHT
                            (* 0.5 *RHO* CRUISE_VELOCITY CRUISE_VELOCITY
                               WING_REFERENCE_AREA))
          :TeX-name "$C_L {} = {} {W_{Cr}
                                 \\over {1/2 \\rho V_{Cr}^2 S_{ref}}}$"
          :documentation "Lift Coefficient Equation.")
```

Figure 3.4: Design function declaration in the source file.

units can be any units you need to write the design function the simplest, as long as they agree with the design variable's dimensions; *Paper Airplane* will take care of all units conversions.

df-body is the function body of the design function, which is written exactly as it would be in a COMMON LISP defun. The body should relate only the design variables that are contained in the input list, not the output list. See Appendix A for a discussion of LISP programming.

df-tex-symbol is the LATEX representation of the design function name seen in hardcopy output like Table 2.2, such as the equation for Lift Coefficient. The LATEX form is a string, thus it must be surrounded by double quotes. If your system does not support LATEX, do not worry, the LATEX form is optional to the design function declaration (note the thin square brackets). See Appendix B for instructions on how to represent design variables using LATEX.

df-description is a description or definition of the design function, such as "`"Lift-to-Drag Ratio Equation."`". The description is a string, thus it must be surrounded by double quotes. The description is optional to the design function declaration, but is recommended.

Design function declarations, as you can see, are much more complicated than auxiliary function declarations. For example, you need to specify the output of a design function; whereas you do not with an auxiliary function. Also, you need to specify the units of each input and output variable of the design function; whereas you do not with an auxiliary function. This is not bad, however; it is actually good, since anyone looking at the declaration knows immediately what kind of values are required and expected from the design function. And the format itself is not very different from the argument list — it is an argument list, and the arguments just happen to be pairs of design variable name and units.

3.3.5 Design Set Declaration

Following the declaration of design functions should come the last item, the declaration of your design sets (as shown in Figure 3.5). The declaration format is as follows:

```
(pa-defset ds-name
        :design-variables dv-list
        :auxiliary-functions af-list
        :design-functions df-list
        :tableaux tab-list )
```

where

ds-name is the name of the design set, such as "LASER".

dv-list is the list of the names of the design variables you wish to place in the design set, surrounded by parentheses.

```
;;;;;;;;;;;;;;;;;;;; DESIGN SETS ;;;;;;;;;;;;;;;;;;;;;

(pa-defset laser
           :design-variables
           (ASPECT_RATIO LIFT_COEFFICIENT DRAG_COEFFICIENT LIFT-TO-DRAG_RATIO
                         GROSS_TAKE-OFF_WEIGHT PAYLOAD_WEIGHT FUEL_WEIGHT
                         ZERO-LIFT_DRAG_COEFF CRUISE_WEIGHT CRUISE_VELOCITY
                         range TSFC OSWALD_EFFICIENCY MIN_LANDING_WEIGHT
                         EMPTY_WEIGHT_FRACTION WING_REFERENCE_AREA
                         TIME_ON_RESERVES)
           :design-functions (DF-1 DF-2 DF-3 DF-4 DF-5 DF-6 DF-7)
           :tableaux
           ((aerodynamics
              WING_REFERENCE_AREA ASPECT_RATIO OSWALD_EFFICIENCY
              LIFT-TO-DRAG_RATIO LIFT_COEFFICIENT DRAG_COEFFICIENT
              ZERO-LIFT_DRAG_COEFF)
            (cruise
              RANGE CRUISE_VELOCITY TSFC LIFT-TO-DRAG_RATIO
              GROSS_TAKE-OFF_WEIGHT MIN_LANDING_WEIGHT TIME_ON_RESERVES)
            (weights
              GROSS_TAKE-OFF_WEIGHT PAYLOAD_WEIGHT FUEL_WEIGHT
              MIN_LANDING_WEIGHT CRUISE_WEIGHT EMPTY_WEIGHT_FRACTION)))
```

Figure 3.5: Design set declaration in the source file.

af-list **is the list of the names of the auxiliary functions you wish to place in the design set, surrounded by parentheses.**

df-list **is the list of the names of the design functions you wish to place in the design set, surrounded by parentheses.**

tab-list **is the list of the tableaux declarations you wish to define, surrounded by parentheses. Each tableau declaration consist of a list with the name of the tableau followed by the names of the design variables to put in it, surrounded by parentheses.**

You know enough to create a source file, or at least to edit one, which is what you will do in the next section.

3.4 Tutorial Number 2

Now that you have been given the declaration formats for all the objects in the source file, you need to *learn* them; and the best way to do so is by editing a source file, and creating those objects yourself. In this section, you will edit the source files — note plural — belonging to the second tutorial design set LASER-2. You will be asked to complete the revision that was started by the author (purposely left unfinished) in order for the LASER-2 design set to be exercised in the next chapter.

In this tutorial, you will be required to use an EMACS editor. Because learning a new editor is always difficult, special *Paper Airplane*-defined editor functions have been added to help the user create design variable, auxiliary function, design function, and design set declarations easily and quickly.

3.4.1 Editing the Tutorial Source File

The actual "EXAMPLE2.SOU" source file is write-protected, so you will have to make your own copy of it and put it in your home directory. Then start up NIL and enter the EMACS editor using the following procedure:

```
$ copy pa$disk:[examples]example2.sou sys$login:*.* Return
$ nil Return
      ⋮
*
(ed sys$login:example2.sou)
      ⋮
```

You have now entered the EMACS editor and the screen should appear as shown in Figure 3.6. The color-inverted line near the bottom of the screen is the "mode line" and is used to identify the EMACS editor ("STEVE"), the editor mode ("(PA)"), the file editing buffer ("EXAMPLE2.SOU[=]SYS$LOGIN:WILBUR:: (1)"), and the status of the keypad functions ("(F---)"). Help on the keypad functions is available by pressing the 0 key on the keypad.

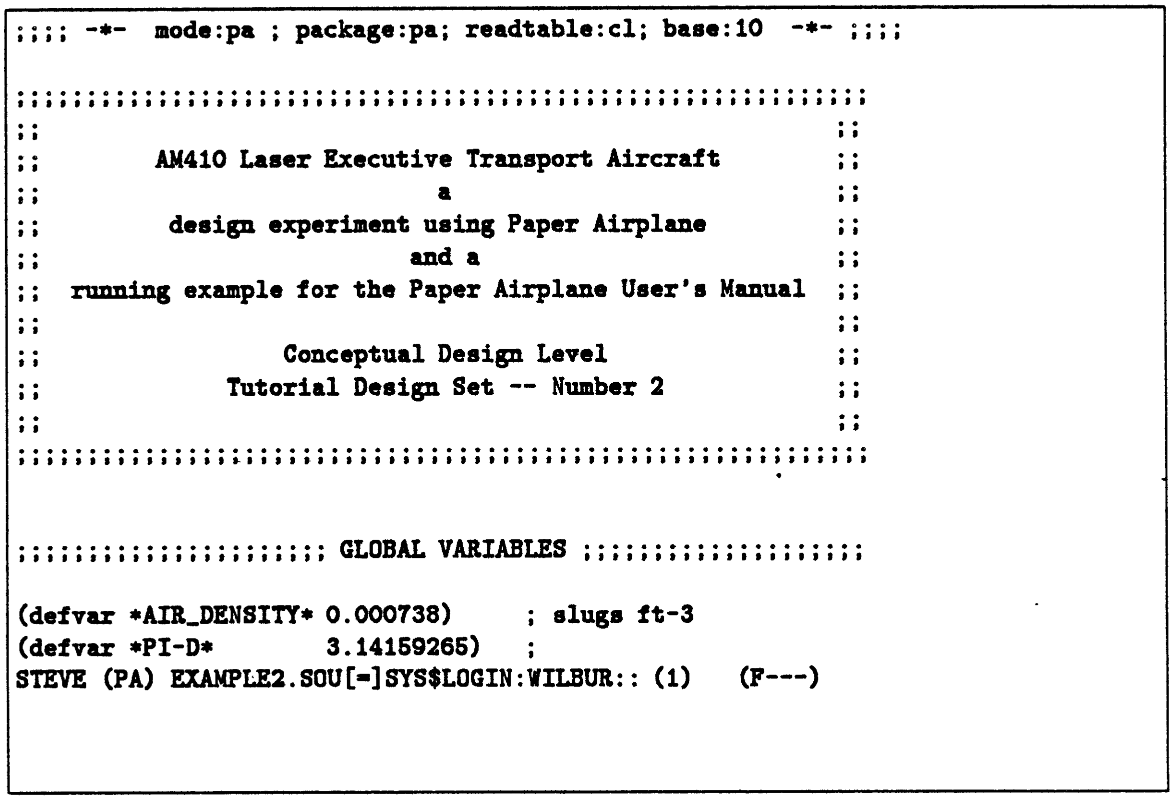
The special editor functions for building *Paper Airplane* declarations are chosen by using a Meta-P (ESC P) prefix, which becomes the prompt "verbPA-". Interactive help in building declarations is available by following the Meta-P prefix with a lowercase letter, or, if you prefer a more manual help, declaration skeletons are available by following the Meta-P prefix with a uppercase letter. The letters are as follows:

v — interactive design variable declaration function.

a — interactive auxiliary function declaration function.

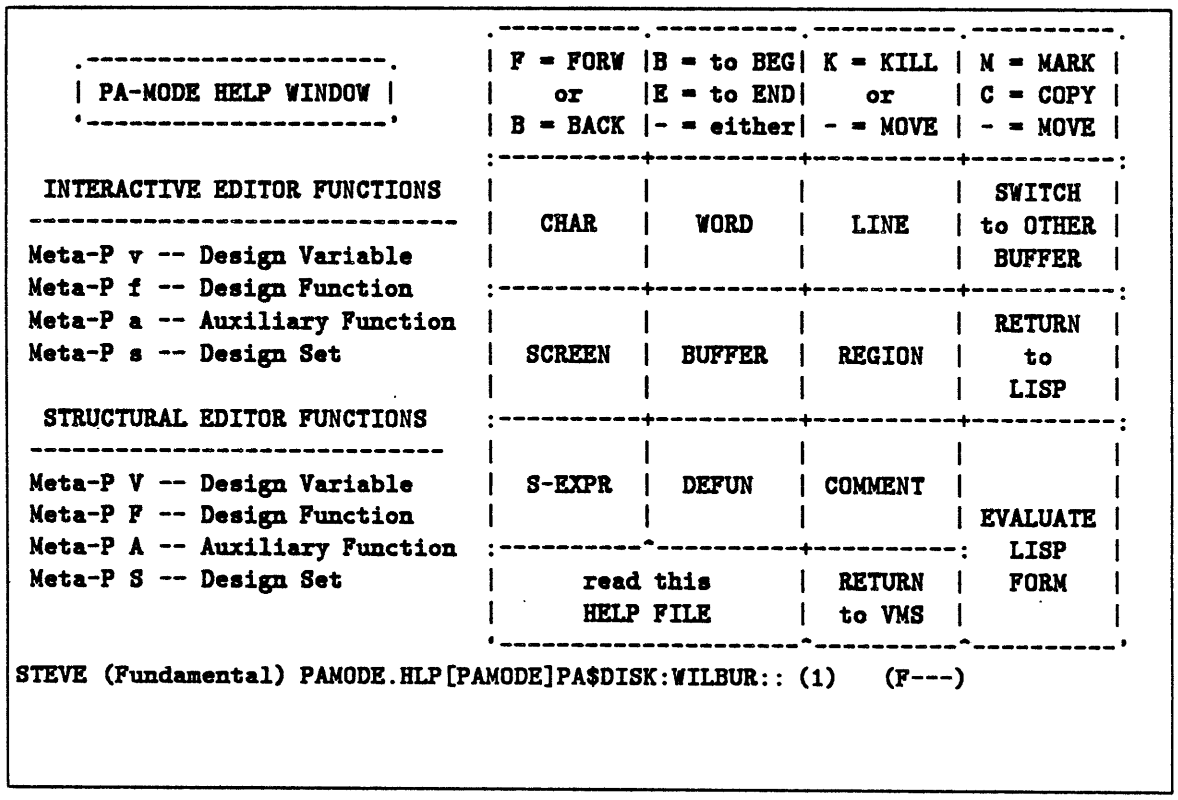
f — interactive design function declaration function.

s — interactive design set declaration function.



```
;;;; -*-  mode:pa ; package:pa; readtable:cl; base:10  -*- ;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                        ;;
;;        AM410 Laser Executive Transport Aircraft        ;;
;;                           a                            ;;
;;         design experiment using Paper Airplane         ;;
;;                         and a                          ;;
;;  running example for the Paper Airplane User's Manual  ;;
;;                                                        ;;
;;                 Conceptual Design Level                ;;
;;             Tutorial Design Set -- Number 2            ;;
;;                                                        ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;; GLOBAL VARIABLES ;;;;;;;;;;;;;;;;;;;;

(defvar *AIR_DENSITY* 0.000738)      ; slugs ft-3
(defvar *PI-D*        3.14159265)    ;
STEVE (PA) EXAMPLE2.SOU[-]SYS$LOGIN:WILBUR:: (1)    (F---)
```

Figure 3.6: The EMACS editor screen.



```
                                 .----------.----------.----------.----------.
  .---------------------.        | F = FORW |B = to BEG| K = KILL | M = MARK |
  | PA-MODE HELP WINDOW |        |    or    |E = to END|    or    | C = COPY |
  '---------------------'        | B = BACK |- = either| - = MOVE | - = MOVE |
                                 :----------+----------+----------+----------:
 INTERACTIVE EDITOR FUNCTIONS    |          |          |          |  SWITCH  |
-----------------------------    |   CHAR   |   WORD   |   LINE   | to OTHER |
Meta-P v -- Design Variable      |          |          |          |  BUFFER  |
Meta-P f -- Design Function      :----------+----------+----------+----------:
Meta-P a -- Auxiliary Function   |          |          |          |  RETURN  |
Meta-P s -- Design Set           |  SCREEN  |  BUFFER  |  REGION  |    to    |
                                 |          |          |          |   LISP   |
 STRUCTURAL EDITOR FUNCTIONS     :----------+----------+----------+----------:
----------------------------     |          |          |          |          |
Meta-P V -- Design Variable      |  S-EXPR  |  DEFUN   | COMMENT  |          |
Meta-P F -- Design Function      |          |          |          | EVALUATE |
Meta-P A -- Auxiliary Function   :----------^----------+----------:   LISP   |
Meta-P S -- Design Set           |      read this      |  RETURN  |   FORM   |
                                 |      HELP FILE      |  to VMS  |          |
                                 '---------------------^----------^----------'
STEVE (Fundamental) PAMODE.HLP[PAMODE]PA$DISK:WILBUR:: (1)   (F---)
```

Figure 3.7: EMACS keypad help screen.

V — skeletal design variable declaration function.

A — skeletal auxiliary function declaration function.

F — skeletal design function declaration function.

S — skeletal design set declaration function.

Chapter 4

Exercising the Design Set

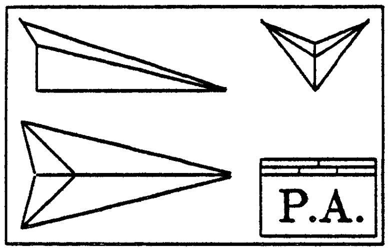
This chapter is not ready yet; however, you can go back to Chapter 1 for most of the information you will need to exercise any design set. For more exercise, you can load the source files EX2DVARS.SOU, EX2DFUNS.SOU, and EX2DSET.SOU — in that order — into a *clean* library and create the LASER-2 design set. This design set includes a design function that calls the external code RANGE and a design function that returns more than one output value.

Appendix A

The Basics of COMMON LISP

This appendix instructs the user in the basics of COMMON LISP programming required to form the bodies of design functions and auxiliary functions.

Until this appendix is ready, the author refers the user to Guy Steele's text on COMMON LISP [14].



Appendix B

The Basics of LATEX

This appendix instructs the user in the basics of LaTeX, a document preparation system that can be used to create professional-quality tables of mathematical equations and their symbols.

Until this appendix is ready, the author refers the user to Leslie Lamport's text on LaTeX [9].

Appendix C

Utility Functions

This appendix instructs the user how to use the *Paper Airplane* Utility Functions. The first section describes the usage of the table-lookup functions, and the second section describes the usage of the external code interface.

C.1 Table-lookup Functions

C.1.1 General Table Look-up Function

The format for using the general table look-up function is shown below:

```
(table-lookup
              :dimensions dim
              :indices indice-list
              :indice-form x-var-list
              :values yvalues
              :value-form y-var
              :lookup-indices index-list )
```

where

dim is the number of dimensions of the values.

indice-list is the list of indice-lists, one per dimension.

x-var-list is the list of variation formats of the indices.

yvalues are the values for each grouping of indices. A matrix form reduced to lists of lists.

y-var is the variation format of the values.

index-list is the list of indices, one per dimension.

Variation formats allow interpolation to be performed with greater precision by fitting the data to the curves approximating their true shapes. Variation formats can best be understood by rationalizing how the output variable varies with each input variable. For example, an output variable z could vary exponentially with an input variable x and geometrically with another input variable y. In this case the variation of z would be "LINEAR"; that of x would be "EXPONENTIAL"; and that of y would be "GEOMETRIC."

Although the variation of the output variable would normally be "LINEAR," there is possibility of strange relationships such as the square root of z could vary logarithmically with x. This would require that the variation of z be "GEOMETRIC" while the variation of x would be "LOGARITHMIC."

The choices for variable variation are **'LINEAR**, **'EXPONENTIAL**, **'GEOMETRIC**, and **'LOGARITHMIC**. The single quote is required when the names appear on there own. When they are part of a list, the list must be quoted.

An example follows:

```
(defun test (index)
  (table-lookup
    :dimensions 3
    :indices '((0 4 25) (2 3 4 5) (1 2 3 4 5))
    :xform '(linear geometric linear)
    :yform 'linear
    :values
'(((2 4 6 8 10) (3 6 9 12 15) (4 8 12 16 20) (5 10 15 20 25))
 ((6 8 10 12 14) (7 10 13 16 19) (8 12 16 20 24) (9 14 19 24 29))
 ((27 29 31 33 35) (28 31 34 37 40) (29 33 37 41 45) (30 35 40 45 50)))
    :lookup-indices index))
```

C.1.2 One-dimensional Table Look-up Function

The format for using the one dimensional table look-up function is shown below:

```
(1d-table-lookup
            :x-indices x-tab
            :x-variation x-var
            :y-values y-tab
            :y-variation y-var
            :x-index x-val )
```

where

x-tab is the list of indices.

x-var is the variation format of the indices.

y-tab is the list of values.

y-var is the variation format of the values.

x-val is the the value of the index.

An example follows:

```
(defvar *ALT* '(80000.0 82021.0 85000.0 90000.0 95000.0 100000.0
                        110000.0 120000.0 130000.0 140000.0 150000.0))
(defvar *RHO* '(0.04410 0.04002 0.03428 0.02655 0.02067 0.016170
                        0.010040 0.006344 0.004076 0.002658 0.0017591))

(pa-defun CRUISE-DENSITY-FUNCTION
          :category (aerodynamics)
          :computed-variable (CRUISE_AIR_DENSITY "kg m-3")
          :input-variables ((CRUISE_ALTITUDE "ft"))
          :function-body (1d-table-lookup :x-indices *ALT*
                                          :x-variation 'exponential
                                          :y-values *RHO*
                                          :x-index CRUISE_ALTITUDE)
          :TeX-name "")
```

C.1.3 Two-dimensional Table Look-up Function

The format for using the two-dimensional table look-up function is shown below:

```
(2d-table-lookup
          :s-indices s-tab
          :s-variation s-var
          :t-indices t-tab
          :t-variation t-var
          :z-map z-map
          :z-variation z-var
          :s-index s-val
          :t-index t-val )
```

where

s-tab is the list of indices in the first dimension.

s-var is the variation format of the first indices.

t-tab is the list of indices in the second dimension.

t-var is the variation format of the second indices.

z-map is the list of list of values.

z-var is the variation format of the values.

s-val is the the value of the first index.

t-val is the the value of the second index.

C.1.4 Family of Curves Look-up Function

The curves look-up function follows a different format for two-dimensional table look-up processing. The format is shown below:

```
(curves-lookup
              :plot-values p-tab
              :p-variation p-var
              :x-indices x-tablist
              :x-variation x-var
              :y-values y-tablist
              :y-variation y-var
              :p-index p-val
              :x-index x-val )
```

where

p-tab is the list of indices to the curves.

p-var is the variation format of the curves.

x-tablist is the list of lists of x-locations of each point, one list per curve. This way, each curve can have a different number of points.

x-var is the variation format of the x indices.

y-tablist is the list of lists of y-locations of each point, one list per curve. This way, each curve can have a different number of points.

y-var is the variation format of the y values.

p-val is the the value of the index between the curves.

x-val is the the value of the index between the points.

C.2 External Code Interface

The format for using the general table look-up function is shown below:

```
(run-program
             :program-name prog-name
             :program-directory prog-dir
             :file-directory file-dir
             :preprocessor preprocessor
             :postprocessor postprocessor
             :sys$input-file sys-infile
             :sys$output-file sys-outfile
             :sys$error-file sys-errfile
:monitor-file mon-file
:average-run-time runtime
:overtime-allowance overtime
:verbose verbose )
```

where

prog-name is the name of program as a symbol (i.e., quoted).

prog-dir is the directory location of the program executable. This information should be a string, and therefore surrounded by double quotes.

file-dir is the directory location for the input and output files, which defaults to the program directory if it is not specified. This information is also a string. For *Paper Airplane* applications, this should be a null string (i.e., " "" ").

preprocessor is the name of the COMMON LISP defun that will prepare the input files to be read by the external code. This name, like the program name, is a symbol. The default is the program name followed by "`-preprocessor`". For *Paper Airplane* use, the preprocessor should be **NIL** (off) since it expects to handle global variables rather than design variables. The preprocessor should be called before the call to run the program.

postprocessor is the name of the COMMON LISP defun that will process the output files to written by the external code. This name, like the program name, is a symbol. The default is the program name followed by "`-postprocessor`". For *Paper Airplane* use, the postprocessor should be **NIL** (off) since it expects to handle global variables rather than design variables. The postprocessor should be called after the call to run the program.

sys-infile is the name of the file to be used as SYS$INPUT to the program. The name is a string. The default is the program name followed by "`.in`".

sys-outfile is the name of the file to be used as SYS$OUTPUT from the program. The name is a string. The default is the program name followed by "`.out`".

sys-errfile is the name of the file to be used as SYS$ERROR from the program. The name is a string. The default is the program name followed by "`.err`".

mon-file is the name of the file to monitored to find out when the program has terminated. The name is a string. The default is whatever you provide for the SYS$OUTPUT file.

runtime is the average expected run time for the code, in seconds. The monitor "sleeps" for 80/the program has terminated. The default is 10 seconds.

overtime is the allowable extra time the monitor should continue to wait for the program to terminate past the given run time, in percent. The default is 20/

verbose is a flag to have the function report what it is doing or not. For *Paper Airplane* use, this flag should be NIL (off).

Typical forms for a preprocessor and a postprocessor are shown below.

```
(DEFUN program-preprocessor ()
  (LET ((INFILE (OPEN "program input file name" 'OUT)))
    (UNWIND-PROTECT
        (PROGN
          (FORMAT INFILE "data form" data-values)
;                     .
;                     .
;                     .
          (FORMAT INFILE "data form" data-values))
      (CLOSE INFILE))))

(DEFUN program-postprocessor ()
  (LET ((symbol1 value1) (symbol2 value2) ... (symbolN valueN))
    (WITH-OPEN-FILE (OUTFILE "program output file name")
      (SETQ symbol1 (readline OUTFILE))
      (SETQ symbol1 (readline OUTFILE))
;                     .
;                     .
;                     .
      (SETQ symbolN (readline OUTFILE)))
    (VALUES symbol1 symbol2 ... symbolN)))
```

An example follows:

```
(pa-defun PERFORMANCE-PROGRAM
          :category (performance overall)
          :computed-variables ((RANGE "sm")
                               (TIME_OF_FLIGHT "s"))
          :input-variables ((CRUISE_MACH "")
                            (CRUISE_ALTITUDE "m")
                            (CLIMB_ANGLE "deg")
                            (CLIMB_ACCELERATION "g")
                            (WING_REFERENCE_AREA "m2")
                            (INLET_CAPTURE_AREA "m2")
                            (MAXIMUM_LIFT_COEFF "")
                            (VEHICLE_GROSS_WEIGHT "kg")
                            (FUEL_WEIGHT "kg")
                            (FUEL_RESERVES "kg"))
          :function-body
          (progn
            (perf-preprocessor CRUISE_MACH CRUISE_ALTITUDE CLIMB_ANGLE
                          CLIMB_ACCELERATION WING_REFERENCE_AREA
                          INLET_CAPTURE_AREA MAXIMUM_LIFT_COEFF
                          VEHICLE_GROSS_WEIGHT FUEL_WEIGHT
                          FUEL_RESERVES *FUEL-TO-AIR_RATIO*)
  (run-program :program-name         'perf
               :program-directory    "sys$user:[ftl.rml.nasp.perf]"
               :file-directory       ""
               :preprocessor nil     :postprocessor nil
               :average-run-time 100 :overtime-allowance  50)
  (perf-postprocessor))
:TeX-name "")
```

Appendix D

Dimensions and Units

This appendix lists all the dimensions and units that are pre-defined inside of *Paper Airplane*. It also instructs the user how to define new units and even new dimensions.

D.1 Dimensions

Below is a list of all dimensions pre-defined inside of *Paper Airplane*.

Pre-defined Derived Dimensions

Name	Definition	Derivation
`f`	force	`t-2 m l`
`F`	force	`t-2 m l`
`p`	pressure	`t-2 m l-1`
`E`	energy	`t-2 m l+2`
`P`	power	`t-3 m l+2`
`Q`	charge	`I t`
`V`	voltage	`I-1 t-3 m l+2`
`R`	resistance	`I-2 t-3 m l+2`
`Z`	impedance	`I-2 t-3 m l+2`
`G`	conductance	`I+2 t+3 m-1 l-2`
`C`	capacitance	`I+2 t+4 m-1 l-2`
`L`	inductance	`I-2 t-2 m l+2`
`M`	magnetic flux	`I-1 t-2 m l+2`
`B`	magnetic inductance	`I-1 t-2 m`

Pre-defined Canonical Dimensions

Dimension	Definition	Base Unit
l	length	m
m	mass	kg
t	time	s
T	temperature	K
A	angle	rad
I	current	A
c	currency	$
a	amount of substance	mol
LI	luminous intensity	cd

D.2 Units

Below is a list of all units pre-defined inside of *Paper Airplane*.

Pre-defined Units of Length

Unit Name	Symbol	Plurul Definition	Dimensions
f	f	fermis	l
pm	pm	picometers	l
Ang	Å	angstroms	l
nm	nm	nanometers	l
um	μm	micrometers	l
mil	mil	mils	l
mm	mm	millimeters	l
cm	cm	centimeters	l
in	in	inches	l
ft	ft	feet	l
yd	yd	yards	l
m	m	meters	l
fath	fath	fathoms	l
fr	fr	furlongs	l
kft	kft	kilofeet	l
km	km	kilometers	l
mi	mi	miles	l
SM	SM	statute miles	l
NM	NM	nautical miles	l
Mm	Mm	megameters	l
Re	$R_\oplus$	earth radii	l
AU	AU	astronomical units	l
ly	ly	light years	l
par	par	parsecs	l
Mpar	Mpar	megaparsecs	l

Pre-defined Units of Mass

Unit Name	Symbol	Plurul Definition	Dimensions
me	m_e	electron-masses	m
AMU	AMU	atomic mass units	m
pg	pg	picograms	m
ng	ng	nanograms	m
ug	μg	micrograms	m
mg	mg	milligrams	m
g	g	grams	m
ozm	oz_m	ounces-mass	m
lbm	lb_m	pounds-mass	m
kg	kg	kilograms	m
sl	sl	slugs	m
t	t	metric tons	m
Me	$M_\oplus$	earth-masses	m
Ms	$M_\odot$	solar-masses	m

Pre-defined Units of Time and Frequency

Unit Name	Symbol	Plurul Definition	Dimensions
ps	ps	picoseconds	t
ns	ns	nanoseconds	t
us	μs	microseconds	t
ms	ms	milliseconds	t
s	s	seconds	t
min	min	minutes	t
hr	hr	hours	t
d	d	days	t
wk	wk	weeks	t
fn	fn	fortnights	t
yr	yr	years	t
dec	dec	decades	t
cen	cen	centuries	t
millen	millen	millennia	t
		— ⋆ —	
Hz	Hz	hertz	t-1
kHz	kHz	kilohertz	t-1
MHz	MHz	megahertz	t-1
GHz	GHz	gigahertz	t-1

Pre-defined Units of Angle and Angular Motion

Unit Name	Symbol	Plurul Definition	Dimensions
`mdeg`	mdeg	millidegrees	`A`
`mrad`	mrad	milliradians	`A`
`deg`	deg	degrees	`A`
`rad`	rad	radians	`A`
`rev`	rev	revolutions	`A`
		— ★ —	
`rpm`	rpm	revolutions-per-minute	`A t-1`

Pre-defined Units of Area and Volume

Unit Name	Symbol	Plurul Definition	Dimensions
`acre`	acre	acres	`l+2`
		— ★ —	
`ul`	μl	microliters	`l+3`
`ml`	ml	milliliters	`l+3`
`cc`	cc	cubic-centimeters	`l+3`
`pt`	pt	pints	`l+3`
`qt`	qt	quarts	`l+3`
`l`	l	liters	`l+3`
`gal`	gal	gallons	`l+3`

Pre-defined Units of Velocity and Acceleration

Unit Name	Symbol	Plurul Definition	Dimensions
`fps`	fps	feet-per-second	`t-1 l`
`mph`	mph	miles-per-hour	`t-1 l`
`kt`	kt	knots	`t-1 l`
`c`	c	light-speeds	`t-1 l`
		— ★ —	
`g's`	g's	g's	`t-2 l`

Pre-defined Units of Force and Weight

Unit Name	Symbol	Plurul Definition	Dimensions
dyn	dyn	dynes	t-2 m 1
oz	oz	ounces	t-2 m 1
ozf	oz_f	ounces-force	t-2 m 1
N	N	newtons	t-2 m 1
lb	lb	pounds	t-2 m 1
lbf	lb_f	pounds-force	t-2 m 1
kgf	kg_f	kilograms-force	t-2 m 1
kN	kN	kilonewtons	t-2 m 1
ton	ton	U.S. tons	t-2 m 1
kton	kton	kilotons	t-2 m 1
Mton	Mton	megatons	t-2 m 1

Pre-defined Units of Pressure

Unit Name	Symbol	Plurul Definition	Dimensions
ubar	μbar	microbars	t-2 m 1-1
Pa	Pa	pascals	t-2 m 1-1
psf	psf	pounds-per-square-foot	t-2 m 1-1
mbar	mbar	millibars	t-2 m 1-1
kPa	kPA	kilopascals	t-2 m 1-1
psi	psi	pounds-per-square-inch	t-2 m 1-1
bar	bar	bars	t-2 m 1-1
atm	atm	atmospheres	t-2 m 1-1
MPa	MPa	megapascals	t-2 m 1-1
ksi	ksi	kilopounds-per-square-inch	t-2 m 1-1
kbar	kbar	kilobars	t-2 m 1-1
Mbar	Mbar	megabars	t-2 m 1-1

Pre-defined Units of Energy, Heat, and Work

Unit Name	Symbol	Plurul Definition	Dimensions
eV	eV	electron volts	t-2 m l+2
keV	keV	kilo-electron volts	t-2 m l+2
MeV	MeV	mega-electron volts	t-2 m l+2
erg	erg	ergs	t-2 m l+2
mJ	mJ	millijoules	t-2 m l+2
J	J	joules	t-2 m l+2
cal	cal	calories	t-2 m l+2
kJ	kJ	kilojoules	t-2 m l+2
Btu	Btu	British thermal units	t-2 m l+2
kcal	kcal	kilocalories	t-2 m l+2
kwh	kwh	kilowatt-hours	t-2 m l+2
TNT	TNT	tons of TNT	t-2 m l+2
kTNT	kTNT	kilotons of TNT	t-2 m l+2
MTNT	MTNT	megatons of TNT	t-2 m l+2

Pre-defined Units of Power

Unit Name	Symbol	Plurul Definition	Dimensions
mW	mW	millawatts	t-3 m l+2
W	W	watts	t-3 m l+2
hp	hp	horsepower	t-3 m l+2
kW	kW	kilowatts	t-3 m l+2
MW	MW	megawatts	t-3 m l+2
GW	GW	gigawatts	t-3 m l+2

Pre-defined Units of Temperature

Unit Name	Symbol	Plurul Definition	Dimensions
K	°K	kelvin	T
R	°R	degrees-Rankine	T
deg-C	°C	degrees-Centigrade	T
deg-F	°F	degrees-Fahrenheit	T

Pre-defined Units of Charge, Current, and Voltage

Unit Name	Symbol	Plurul Definition	Dimensions
e	e	electron-charges	I t
mC	mC	millicoulombs	I t
C	C	coulombs	I t
		— ⋆ —	
uA	μA	microamperes	I
mA	mA	milliamperes	I
A	A	amperes	I
kA	kA	kiloamperes	I
		— ⋆ —	
mV	mV	millivolts	I-1 t-3 m 1+2
V	V	volts	I-1 t-3 m 1+2
kV	kV	kilovolts	I-1 t-3 m 1+2
MV	MV	megavolts	I-1 t-3 m 1+2

Pre-defined Units of Resistance and Conductance

Unit Name	Symbol	Plurul Definition	Dimensions
ohm	Ω	ohms	I-2 t-3 m 1+2
kohm	kΩ	kilohms	I-2 t-3 m 1+2
Mohm	MΩ	megohms	I-2 t-3 m 1+2
		— ⋆ —	
S	S	siemens	I+2 t+3 m-1 1-2
mho	℧	mhos	I+2 t+3 m-1 1-2

Pre-defined Units of Capacitance and Inductance

Unit Name	Symbol	Plurul Definition	Dimensions
pF	pF	picofarads	I+2 t+4 m-1 1-2
nF	nF	nanofarads	I+2 t+4 m-1 1-2
uF	μF	microfarads	I+2 t+4 m-1 1-2
mF	mF	millifarads	I+2 t+4 m-1 1-2
F	F	farads	I+2 t+4 m-1 1-2
		— ⋆ —	
uH	μH	microhenrys	I-2 t-2 m 1+2
mH	mH	millihenrys	I-2 t-2 m 1+2
H	H	henrys	I-2 t-2 m 1+2

Pre-defined Units of Magnetic Inductance and Flux

Unit Name	Symbol	Plurul Definition	Dimensions
uG	μG	microgauss	I-1 t-2 m
mG	mG	milligauss	I-1 t-2 m
G	G	gauss	I-1 t-2 m
kG	kG	kilogauss	I-1 t-2 m
T	T	teslas	I-1 t-2 m
		— ★ —	
Wb	Wb	webers	I-1 t-2 m 1+2

Miscellaneous Pre-defined Units

Unit Name	Symbol	Plurul Definition	Dimensions
ct	ct	cents	c
$	$	dollars	c
M$	M$	megabucks	c
		— ★ —	
cd	cd	candelas	LI
		— ★ —	
mol	mol	moles	a
gm-mol	gm-mol	gram-moles	a
lb-mol	lb-mol	pound-moles	a
kg-mol	kg-mol	kilogram-moles	a

Note: The user can be insured that all conversions are done with the highest precision possible. Indeed, most conversion factors are exact by derivation or "exact" by way of conventional and international definitions. Only units with physical derivations (such as electron-masses) are not exact, and this is only because there is scientific uncertainty in their measurement.

D.3 Defining New Dimensions and Units

Paper Airplane provides the user with an easy way to define new units and even new dimensions. It also provides the user with a way to define aliases for existing unit names.

Like design variable and design function definitions, units and dimensions definitions must be put into a source file; however, it is highly recommended that these definitions be put into a ***separate source file***. At the top of the source file should be put the following heading:

```
;;; -*- Mode:Common-Lisp; Package:PDUP; Readtable:CL; Base:10 -*- ;;;
```

If you really want to put all definitions in one source file, you can do so by proceeding the function call with "pdup:" (i.e., change "defunit" to "pdup:defunit").

D.3.1 Defining New Dimensions

Defining a new dimension is done using the following format:

(defdimension *dim-name dim-def dim-deriv*)

where

dim-name is the symbolic name of the dimension, such as “`"F"`”.

dim-def is the definition of the dimension, such as “`"force"`”.

dim-deriv is the derivation of the dimension, such as “`"m l t-2"`”.

Note that all arguments are strings because case is *very* important; therefore, they all must be surrounded by double quotes.

One exception to the above is the case where a new dimension cannot be derived from the others. In this case, *dim-deriv* should be exactly “**:canonical-dimension**” with no double quotes around it.

Some examples of the pre-defined dimensions are shown below:

```
(defdimension "t" "time"          :canonical-dimension)
(defdimension "I" "current"       :canonical-dimension)
(defdimension "Q" "charge"        "I t")
```

D.3.2 Defining New Units

Defining a new unit is done using the following format:

(defunit *unit-name unit-def-s unit-def-p unit-dims unit-derv*)

where

unit-name is the symbolic name of the unit, such as “`"km"`”.

unit-def-s is the singular definition of the unit, such as “`"kilometer"`”.

unit-def-p is the plural definition of the unit, such as “`"kilometers"`”.

unit-dims are the dimensions of the unit, such as “`"l"`”.

unit-deriv is the derivation of the dimension, such as “`(1000.0 "m")`”.

Note that, with the exception of the unit derivation, all arguments are strings because case is *very* important; therefore, they all must be surrounded by double quotes.

If the symbolic name of the unit usually involves Greek or other mathematical symbols, as is the case for micrometers, the *unit-name* can be separated into a list as follows:

(*unit-print-name unit-TeX-name*)

where

unit-print-name is the strictly ASCII name to appear on screen.

unit-TeX-name is the LATEX-formatted symbol to appear in design point tables.

For example, the ***unit-name*** of micrometers is "`("um" "{\\mu}m")`".

If the derivation of a unit does not require a numerical factor, as is the case for miles-per-hour, the derivation does not need to be a list. For example, the derivation of miles-per-hour is simply the string "`"mi hr-1"`".

When a new canonical dimension is defined, a new unit should accompany it as the "base unit" of that dimension. To accomplish this, the derivation of that unit should not be a string but exactly "**:internal-unit**".

Some examples of the pre-defined units are shown below:

```
(defunit "kg"  "kilogram" "kilograms" "m" :internal-unit)
(defunit "mg"  "milligram" "milligrams" "m" (1.0e-6 "kg"))
(defunit ("ug" "{\\mu}g")  "microgram" "micrograms" "m" (1.0e-9 "kg"))
(defunit "kt"  "knot" "knots" "l t-1" "NM hr-1")
(defunit "TNT"  "ton of TNT" "tons of TNT" "E" (4.2e+9 "J"))
(defunit ("deg-F" "{^\\circ\\!}F")
         "degree-Fahrenheit" "degrees-Fahrenheit"
         "T" (1.0 "R") (f(x)(- x 459.688)) (f(x)(+ x 459.688)))
```

The last unit, for degrees-Fahrenheit, was also listed to show how the derivation can also incorporate functions as well as factors. The derivation is "`(1.0 "R")`" but is followed by two COMMON-LISP functions: the first to convert degrees-Rankine to degrees-Fahrenheit, and the second to convert degrees-Fahrenheit to degrees-Rankine.

D.3.3 Aliasing Existing Unit Names

Aliasing an existing unit name is done using the following format:

(alias-unit *unit-name unit-alias*)

where

unit-name is the actual symbolic name of the existing unit, such as "`"km"`".

unit-alias is the alternative symbolic name for the existing unit, such as "`"KM"`".

Please note that aliasing a unit name does not change the name of that unit, but merely allows that unit to be referenced by the specified alias.

Some examples of the aliasing unit names are shown below.

```
(alias-unit "km"  "KM")
(alias-unit "t"   "MT")
(alias-unit "Btu" "BTU")
```

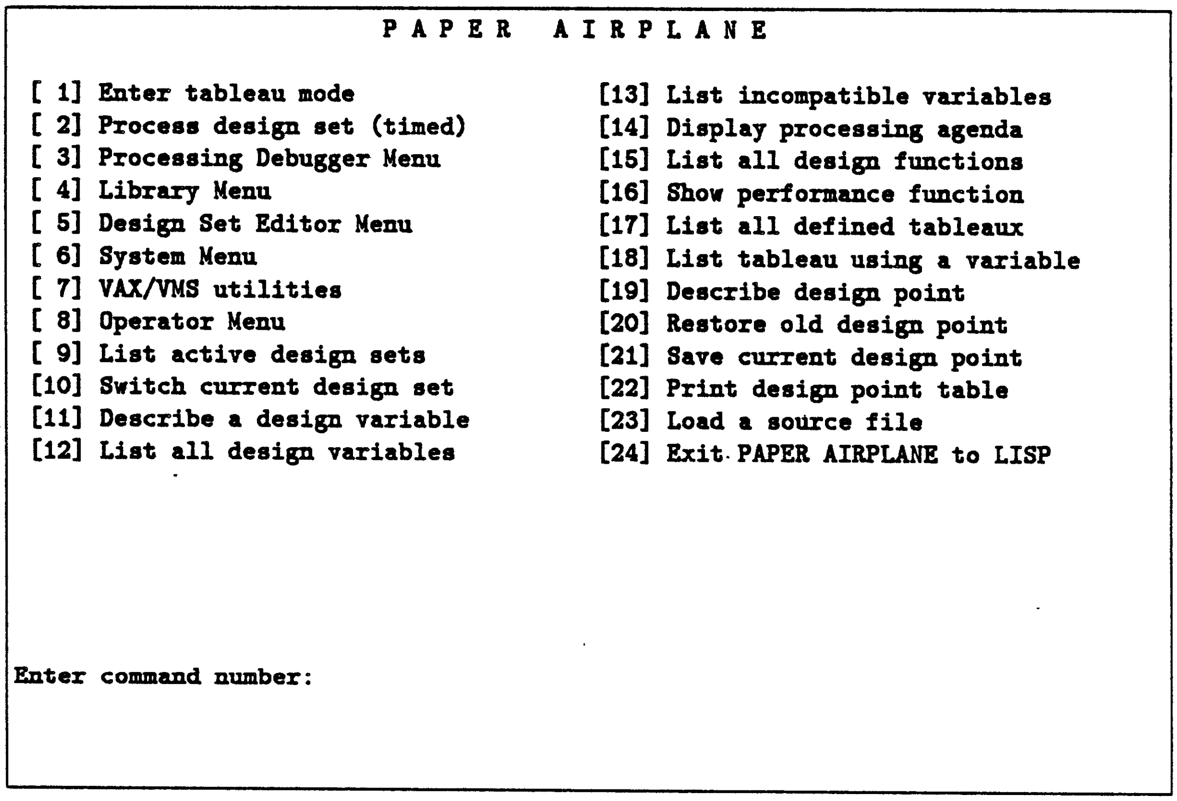
Appendix E

Guide to Menu System

This appendix serves as a guide to the *Paper Airplane* menu system. In addition to the main menu, *Paper Airplane* contains several sub-menus. The following sections describes each menu and each of their choices.

E.1 Main Menu

```
[ 1] Enter tableau mode
[ 2] Process design set (timed)
[ 3] Processing Debugger Menu
[ 4] Library Menu
[ 5] Design Set Editor Menu
[ 6] System Menu
[ 7] VAX/VMS utilities
[ 8] Operator Menu
[ 9] List active design sets
[10] Switch current design set
[11] Describe a design variable
[12] List all design variables
[13] List incompatible variables
[14] Display processing agenda
[15] List all design functions
[16] Show performance function
[17] List all defined tableaux
[18] List tableaux using a variable
[19] Describe design point
[20] Restore old design point
[21] Save current design point
[22] Print design point table
[23] Load a source file
```



```
                         P A P E R   A I R P L A N E

 [ 1] Enter tableau mode                  [13] List incompatible variables
 [ 2] Process design set (timed)          [14] Display processing agenda
 [ 3] Processing Debugger Menu            [15] List all design functions
 [ 4] Library Menu                        [16] Show performance function
 [ 5] Design Set Editor Menu              [17] List all defined tableaux
 [ 6] System Menu                         [18] List tableau using a variable
 [ 7] VAX/VMS utilities                   [19] Describe design point
 [ 8] Operator Menu                       [20] Restore old design point
 [ 9] List active design sets             [21] Save current design point
 [10] Switch current design set           [22] Print design point table
 [11] Describe a design variable          [23] Load a source file
 [12] List all design variables           [24] Exit PAPER AIRPLANE to LISP




Enter command number:
```

Figure E.1: *Paper Airplane* Main Menu.

```
[24] Exit PAPER AIRPLANE to LISP
```

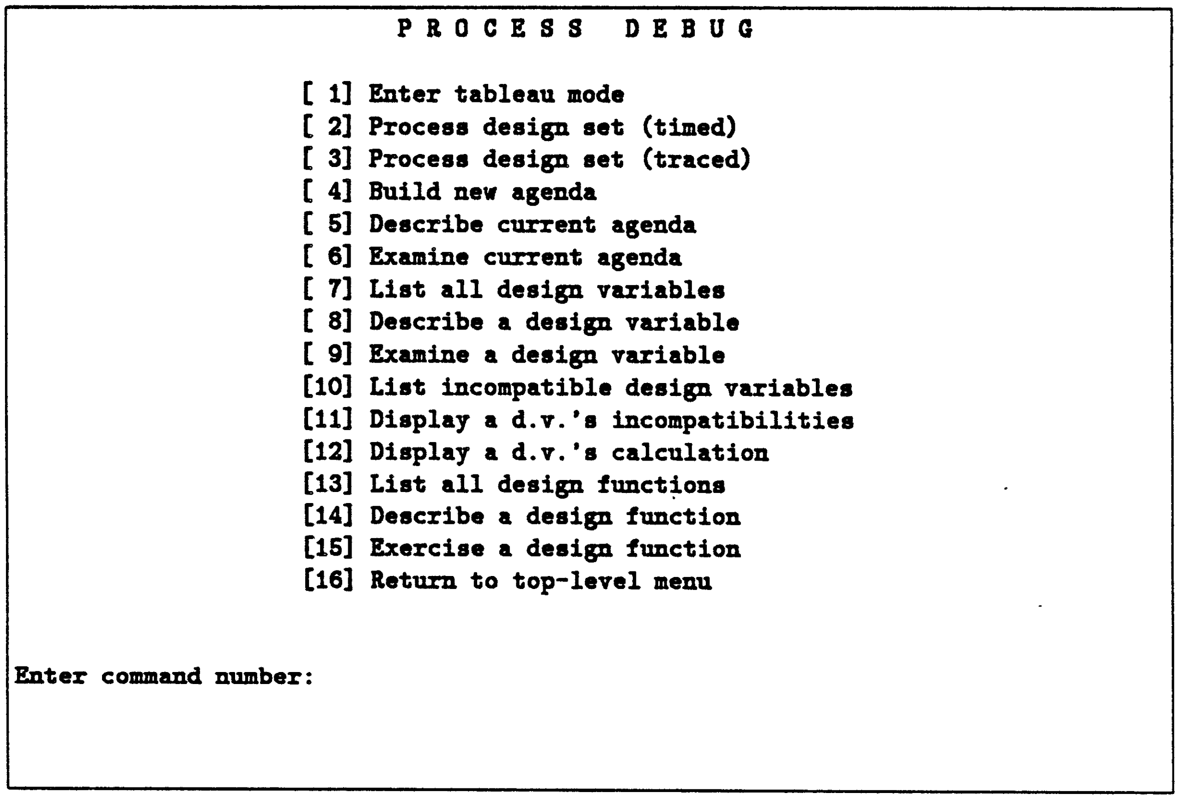
E.2 Processor Debug Menu

```
[ 1] Enter tableau mode
[ 2] Process design set (timed)
[ 3] Process design set (traced)
[ 4] Build new agenda
[ 5] Describe current agenda
[ 6] Examine current agenda
[ 7] List all design variables
[ 8] Describe a design variable
[ 9] Examine a design variable
[10] List incompatible design variables
[11] Display a d.v.'s incompatibilities
[12] Display a d.v.'s calculation
[13] List all design functions
[14] Describe a design function
[15] Exercise a design function
[16] Return to top-level menu
```

E.3 Library Menu

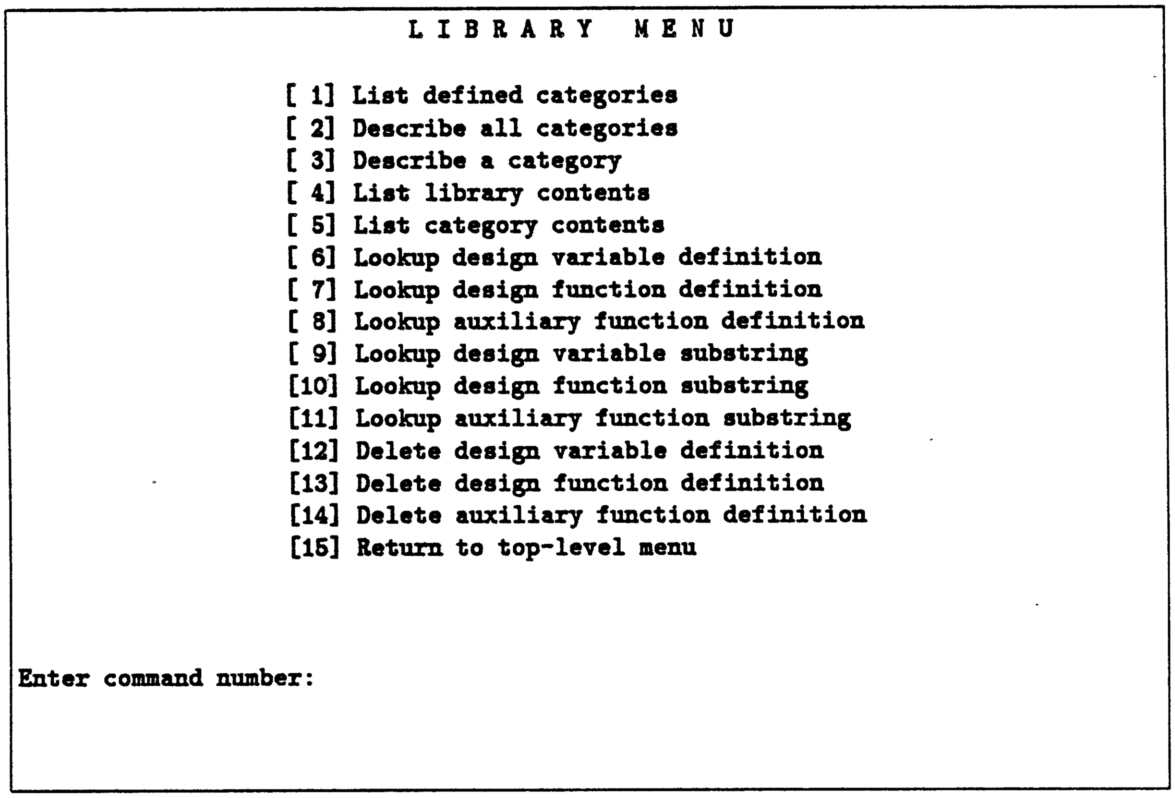
```
[ 1] List defined categories
[ 2] Describe all categories
[ 3] Describe a category
[ 4] List library contents
[ 5] List category contents
[ 6] Lookup design variable definition
[ 7] Lookup design function definition
[ 8] Lookup auxiliary function definition
[ 9] Lookup design variable substring
[10] Lookup design function substring
[11] Lookup auxiliary function substring
[12] Delete design variable definition
[13] Delete design function definition
[14] Delete auxiliary function definition
[15] Return to top-level menu
```

E.4 Design Set Editor Menu



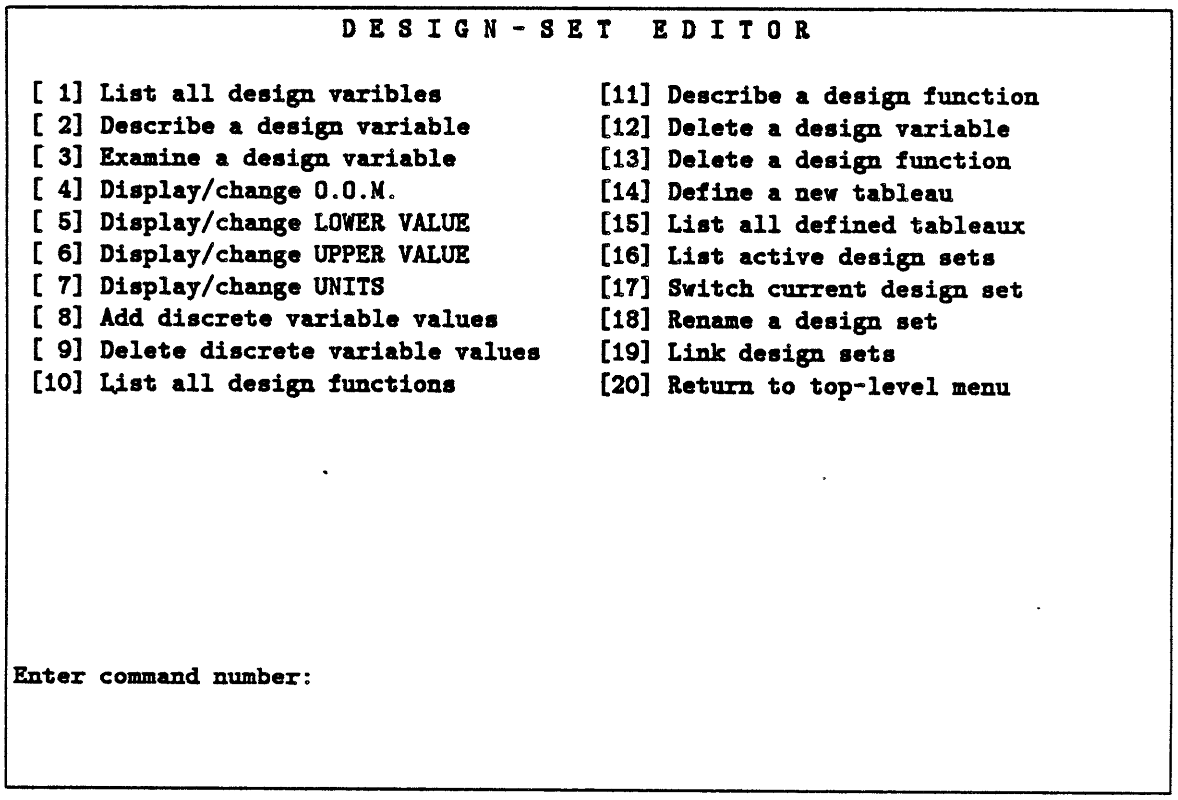
```
                        P R O C E S S   D E B U G

                   [ 1] Enter tableau mode
                   [ 2] Process design set (timed)
                   [ 3] Process design set (traced)
                   [ 4] Build new agenda
                   [ 5] Describe current agenda
                   [ 6] Examine current agenda
                   [ 7] List all design variables
                   [ 8] Describe a design variable
                   [ 9] Examine a design variable
                   [10] List incompatible design variables
                   [11] Display a d.v.'s incompatibilities
                   [12] Display a d.v.'s calculation
                   [13] List all design functions
                   [14] Describe a design function
                   [15] Exercise a design function
                   [16] Return to top-level menu


Enter command number:
```

Figure E.2: *Paper Airplane* Processor Debug Menu.



```
                         L I B R A R Y   M E N U

                  [ 1] List defined categories
                  [ 2] Describe all categories
                  [ 3] Describe a category
                  [ 4] List library contents
                  [ 5] List category contents
                  [ 6] Lookup design variable definition
                  [ 7] Lookup design function definition
                  [ 8] Lookup auxiliary function definition
                  [ 9] Lookup design variable substring
                  [10] Lookup design function substring
                  [11] Lookup auxiliary function substring
                  [12] Delete design variable definition
                  [13] Delete design function definition
                  [14] Delete auxiliary function definition
                  [15] Return to top-level menu


Enter command number:
```

Figure E.3: *Paper Airplane* Library Menu.



```
                         D E S I G N - S E T   E D I T O R

 [ 1] List all design varibles           [11] Describe a design function
 [ 2] Describe a design variable         [12] Delete a design variable
 [ 3] Examine a design variable          [13] Delete a design function
 [ 4] Display/change O.O.M.              [14] Define a new tableau
 [ 5] Display/change LOWER VALUE         [15] List all defined tableaux
 [ 6] Display/change UPPER VALUE         [16] List active design sets
 [ 7] Display/change UNITS               [17] Switch current design set
 [ 8] Add discrete variable values       [18] Rename a design set
 [ 9] Delete discrete variable values    [19] Link design sets
 [10] List all design functions          [20] Return to top-level menu




Enter command number:
```

Figure E.4: *Paper Airplane* Design Set Editor Menu.

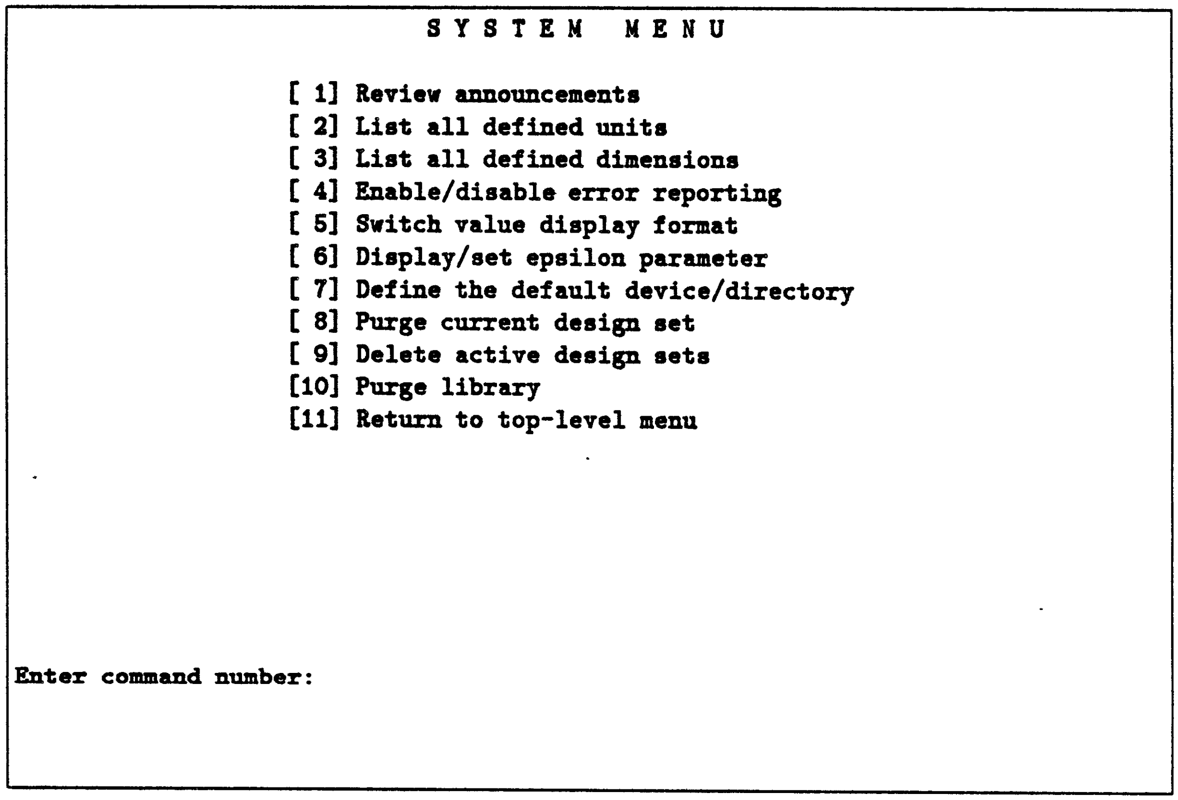
```
[ 1] List all design variables
[ 2] Describe a design variable
[ 3] Examine a design variable
[ 4] Display/change O.O.M.
[ 5] Display/change LOWER VALUE
[ 6] Display/change UPPER VALUE
[ 7] Display/change UNITS
[ 8] Add discrete variable values
[ 9] Delete discrete variable values
[10] List all design functions
[11] Describe a design function
[12] Delete a design variable
[13] Delete a design function
[14] Define a new tableau
[15] List all defined tableaux
[16] List active design sets
[17] Switch current design set
[18] Rename a design set
[19] Link design sets
[20] Return to top-level menu
```

E.5 System Menu

```
[ 1] Review announcements
[ 2] List all defined units
[ 3] List all defined dimensions
[ 4] Enable/disable error reporting
[ 5] Switch value display format
[ 6] Display/set epsilon parameter
[ 7] Define the default device/directory
[ 8] Purge current design set
[ 9] Delete active design sets
[10] Purge library
[11] Return to top-level menu
```

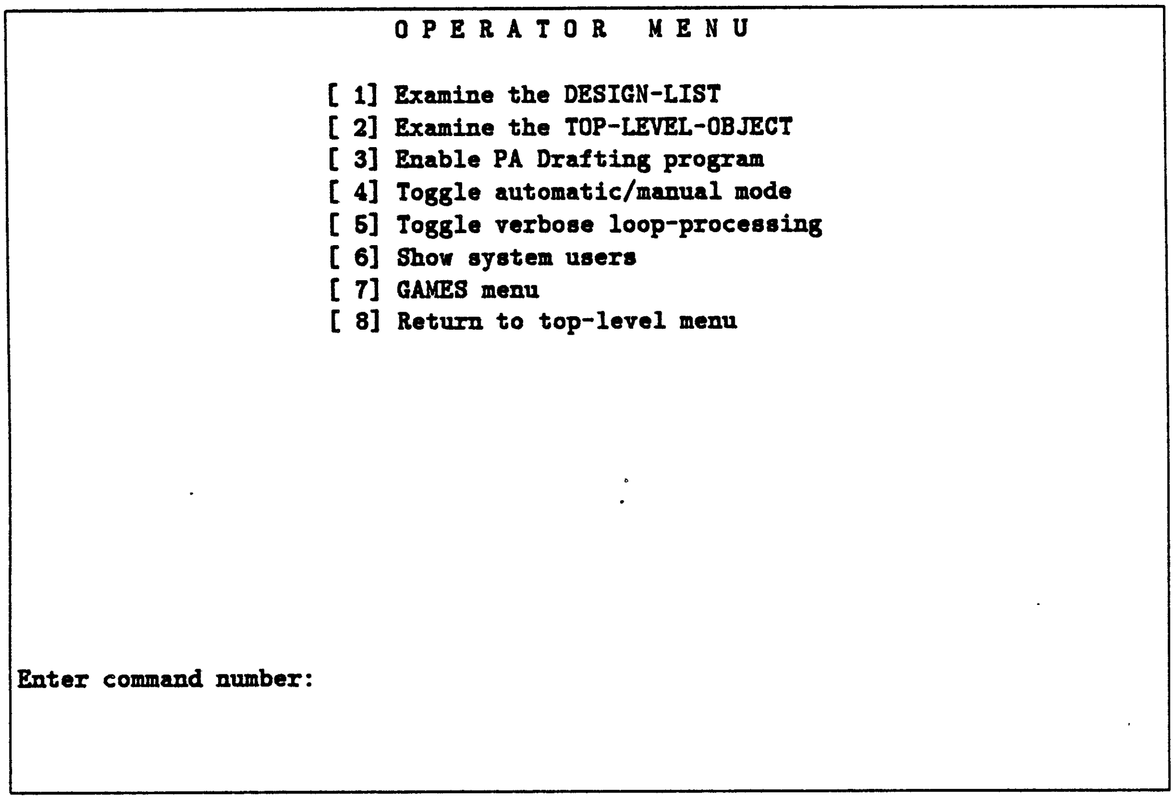
E.6 Operator Menu

```
[ 1] Examine the DESIGN-LIST
[ 2] Examine the TOP-LEVEL-OBJECT
[ 3] Enable PA Drafting program
[ 4] Toggle automatic/manual mode
[ 5] Toggle verbose loop-processing
```



```
                                 S Y S T E M   M E N U

                     [ 1] Review announcements
                     [ 2] List all defined units
                     [ 3] List all defined dimensions
                     [ 4] Enable/disable error reporting
                     [ 5] Switch value display format
                     [ 6] Display/set epsilon parameter
                     [ 7] Define the default device/directory
                     [ 8] Purge current design set
                     [ 9] Delete active design sets
                     [10] Purge library
                     [11] Return to top-level menu







Enter command number:
```

Figure E.5: *Paper Airplane* System Menu.



```
                         O P E R A T O R   M E N U

                     [ 1] Examine the DESIGN-LIST
                     [ 2] Examine the TOP-LEVEL-OBJECT
                     [ 3] Enable PA Drafting program
                     [ 4] Toggle automatic/manual mode
                     [ 5] Toggle verbose loop-processing
                     [ 6] Show system users
                     [ 7] GAMES menu
                     [ 8] Return to top-level menu









Enter command number:
```

Figure E.6: *Paper Airplane* Operator Menu.

```
[ 6] Show system users
[ 7] GAMES menu
[ 8] Return to top-level menu
```

Glossary

agenda: is the common name for the computational agenda.

agenda building: is the process of determining how design functions will be used to find a solution to a chosen design path of knowns and unknowns. Agenda building does not involve any numerical methods since it does use the values of the design variables, only their states.

agenda entry: is an entry into the computational agenda consisting of a perfectly constrained design function and the unknowns to be solved for using it.

base variable: is the official designation of an I-state design variable, commonly referred to as a "known".

branch: is one of two independent sequences of perfectly constrained design functions for computing the value of the loop variable to solve a loop.

C: is the letter assigned to the state of a design variable whose value has been computed by *Paper Airplane* via processing.

C-state design variable: is a design variable whose value has been computed by *Paper Airplane*.

CEMISS: is a Computer-based Engineering Model Information Sharing System.

computational agenda: is the actual path, or sequence of design functions, to be evaluated to find the values of the unknowns once given the initialized values for the knowns and the guess values for the unknowns. The computational agenda is also called the computational path. The computational agenda consists of a forced path and loops.

computational path: is another name for the computational agenda.

Computed-value state: This indicates a design variable that *had* been given a trial value by the user, and was later given a known value by *Paper Airplane*. A design variable obtains state C only when the user processes the design set; and then only if *Paper Airplane* can find a solution which satisfies all the design functions in the user's design set.

computer program: is an external piece of code usually not written in COMMON LISP, such as a FORTRAN or PASCAL program.

derived variable: is the official designation of a G-state design variable, commonly referred to as an "unknown".

design function: is a relationship between design variables. A design function can range in complexity from a simple algebraic equation to a very large and complex computer program.

design path: is the selection of certain design variables as knowns and the rest as unknowns; thereby setting up some implied path, or sequence of design functions, for *Paper Airplane* to follow once values are provided for the design variables.

design point: is the values and states of all the design variables in a design set at any stage of the design process.

design set: is a set of certain design functions and the design variables those functions relate towards the goal of solving a particular design problem.

design variable: is a scalar parameter, such as Vehicle Length or Vehicle Weight, whose value uniquely determine part of the configuration of an aircraft, spacecraft, or any other system. A design variable has a number of attributes associated with it, such as its value, its dimensions, its order of magnitude, and the limits of its value.

engineering mode: is a sub-model (reduced model) of a mathematical model describing the structure and properties of an existing or proposed product.

external code interface: from one program to another allows both programs to share information without the need for a human to manipulate input and output files.

final design point: is the numerical solution to the initial design point of a particular design path. Specifically it is the states and values of the design variables after processing has been completed.

final path: is a sequence of perfectly constrained design functions whose unknowns can be solved for once a loop has been solved.

flavor: is a powerful LISP abstraction that allows for information storage and retrieval and data communications, all in a hierarchical structure.

floating: is the act of changing the state of a design variable to G, thereby setting the value of the design variable as guessed at and marking the design variable as an unknown.

forced path: is a sequence of perfectly constrained design functions, each of which can be solved individually, although sequentially. The path is called "forced" since there is no alternative but to solve the design functions in this sequence in order to compute the values of their unknowns.

forcing variable: is the design variable whose value is converged upon during the iteration of a loop. The forcing variable is usually the design variable most common to all the design functions involved.

forward computation: is a one-time single-function evaluation that computes the values of the output unknowns of a single design function by executing the function once using the values of the input knowns.

freezing: is the act of changing the state of a design variable to I, thereby setting the value of the design variable as initialized and marking the design variable as a known.

function: is an internal piece of code written in COMMON LISP.

G: is the letter assigned to the state of a design variable whose value has been guessed at by the user via floating.

G-state design variable: is a design variable whose value has been guessed at by the user.

Guessed-value state: This indicates a design variable that has been given a trial value by the user. A design variable obtains state G whenever the user floats it. G-state design variables are officially designated as "derived variables" and are commonly referred to simply as "unknowns".

I: is the letter assigned to the state of a design variable whose value has been initialized by the user via freezing.

I/O: Input and output. The data passed to and from a computer program.

I-state design variable: is a design variable whose value has been initialized by the user.

initial design point: is the initial setting of the values of design variables according a particular design path. This consists of initialized values for the chosen knowns and guess values for the chosen unknowns.

initial path: is a sequence of perfectly constrained design functions whose computed unknowns are required by both branches of a loop to solve that loop.

Initialized-value state: This indicates a design variable that has been given a known value by the user. A design variable obtains state I whenever the user changes its value, or when the user freezes it. I-state design variables are officially designated as "base variables" and are commonly referred to simply as "knowns".

instance variable: is a parameter that is an element of the structure of a flavor.

known: is the common name for an I-state design variable, officially designated as a "base variable".

loading: is a COMMON LISP term for reading and evaluating LISP code from a file into main memory.

loop: is a sequence of design functions, each of which computes values required by other design functions in a closed loop. Loops are solved by guessing the value of a forcing variable to compute two independent values of a loop variable. When the two values converge, the values of all the unknowns involved can be found.

loop computation: is an iterative multiple-function evaluation that computes the values of all the unknowns of a set of design functions by guessing values of a chosen forcing variable until two independent values of a chosen loop variable converge.

loop variable: is the design variable whose two independently computed values determine the convergence of a loop.

method: is a function that is specifically associated with a flavor.

MIMO: Multiple-Input Multiple-Output. Loosely, a multiple-input multiple-output design function.

MIMO Design Set: is the design set used to test the MIMO design function solving capability enhancement to *Paper Airplane*. Several MISO design functions from the MISO Design Set were merged to form MIMO design functions.

MISO: Multiple-Input Single-Output. Loosely, a multiple-input single-output design function.

MISO Design Set: is the design set that serves as the foundation of all other design sets, except for the NASP Design Set. The MISO Design Set contains 17 design variables and 7 design functions and is used for the conceptual design of aircraft.

mixin: is a flavor that is an element of the structure of another flavor.

NASP Design Set: is the design set used to test the final version of the enhanced *Paper Airplane*. The NASP Design Set is comprised of 12 design variables and 9 design functions, including MISO and MIMO design functions and design functions calling external codes. The NASP Design Set is used for the preliminary design of a national aerospaceplane.

NIL: the New Implementation of LISP, a dialect of COMMON LISP, and the programming language in which *Paper Airplane* is written.

overconstrained: problem is one in which the number of unknowns is less than the number of values computed by all the design functions (i.e., the number of user-specified knowns is greater than that required). This can lead to design variables receiving two or more incompatible values.

Paper Airplane: is the name of the code development at the Massachusetts Institute of Technology to solve systems of linear and/or non-linear functions.

perfectly constrained: problem is one in which the number of unknowns equals the number of values computed by all the design functions (i.e., the number of user-specified knowns is the same as that required). This usually leads to design variables whose values can be exactly determined.

postprocessor: reads output values from the file(s) an external code normally writes to and returns them to the system.

preliminary entries: is the initial sequence of perfectly constrained design functions created while trying to construct a closed loop.

preprocessor: takes input values from the system and writes them out to the file(s) an external code normally reads from.

processing: is the act of instructing *Paper Airplane* to attempt to compute the values of all the unknowns of a design set.

reverse computation: is an iterative single-function evaluation that computes the values of all the unknowns of a single design function by guessing values of the unknown input variables until values of the known output variables converge with their user-specified values.

source file: is a computer file containing the information on all of the design variables and design functions to be loaded internally into a *Paper Airplane* design set.

state: is the common name for variable state.

TAV Design Set: is the former name of the NASP Design Set.

underconstrained: problem is one in which the number of unknowns is greater than the number of values computed by all the design functions (i.e., the number of user-specified knowns is less than that required). This can lead to design variables whose values cannot be exactly determined.

unknown: is the common name for a G-state design variable, officially designated as a "derived variable".

variable state: is the condition of the value of a design variable. Variable states come in the following three varieties, which are assigned to design variables according to their initial letter: Initialized-value state, Guessed-value state, and Computed-value state.

variable tableau: is a spreadsheet of information on the design set arranged on a computer screen. This information includes a list of design variables and their current values, units, and states.

XCODE: is an external code.

XCODE Design Set: is the design set used to test the external code interface capability enhancement to *Paper Airplane*. One design function from the MISO Design Set was modified to call a FORTRAN program to compute its value.

Bibliography

[1] Conte, S. D., and de Boor, Carl, *Elementary Numerical Analysis: An Algorithmic Approach* (Third edition), McGraw-Hill, New York, NY, 1980.

[2] Elias, Antonio L., "Knowledge Engineering of the Aircraft Design Process," *Knowledge Based Problem Solving*, Prentice-Hall, Englewood Cliffs, NJ, 1985, chapter 6.

[3] Jenkinson, L. R., and Simos, D., "A Computer Program for Assisting in the Preliminary Design of Twin-Engined Propeller-Driven General Aviation Aircraft," *Canadian Aeronautics and Space Journal*, Vol. 30, No. 3, pp. 213-224, September 1984.

[4] Kolb, Mark A., "Problems in the Numerical Solution of Simultaneous Non-linear Equations in Computer-aided Preliminary Design," Memo No. 85-1, Flight Transportation Laboratory, Massachusetts Institute of Technology, May 1985.

[5] Kolb, Mark A., "On the Numerical Solution of Simultaneous, Non-linear Equations in Computer-aided Preliminary Design," S.M. Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, January 1986.

[6] Lajoie, Ronnie M., and Liu, John, *The AM410-Laser Executive Transport Jet: Final Report*, Boston, MA, Aerospace Engineering Department, Boston University, May, 1984.

[7] Lajoie, Ronnie M., and Elias, Antonio L., *Paper Airplane User's Manual* (Version III.1), Flight Transportation Laboratory, Massachusetts Institute of Technology, February, 1986.

[8] Lajoie, Ronnie M., "Integration of Engineering Models in Computer-Aided Preliminary Design," S.M. Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, January 1987.

[9] Lamport, Leslie, LaTeX: *A Document Preparation System*, Addison-Wesley, Reading, MA, 1985.

[10] Lancaster, J. W., and Bailey, D. B., "Naval Airship Program for Sizing and Performance (NAPSAP)," *Journal of Aircraft*, Vol. 18, No. 8, pp. 677-682, August 1981.

[11] Martin, James A., "Aerospaceplane Optimization and Performance Estimation," S.M. Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, August 1967.

[12] Nicolai, Leland M., *Fundamentals of Aircraft Design*, METS, Inc., Dayton, OH, 1975.

[13] Software Arts Inc., *TK!Solver Program Instruction Manual*, Software Arts Inc., Wellesley, MA, 1982.

[14] Steele Jr., Guy L., *Common LISP: The Language*, Digital Press, Burlington, MA, 1984.

[15] Torenbeek, Egbert, *Synthesis of Subsonic Airplane Design*, Delft, Delft University Press, 1982.