AEA FUS 251

AEA Technology
Fusion
(UKAEA/Euratom Fusion Association)

A User's Guide to the PROCESS Systems Code

P. J. Knight

July 1993

AEA Technology
Fusion

Culham, Abingdon
Oxfordshire OX14 3DB
United Kingdom
Telephone 0235 463295
Facsimile 0235 463647

ABSTRACT

The *PROCESS* systems code is being developed to provide an integrated and self-consistent treatment of the physics, engineering, economics, safety and environmental characteristics of tokamak fusion power plants. This user's guide relates to a version of the code prior to the incorporation of the safety and environmental modules, and to improvements and extensions to the modelling in other areas.

Contents

Chapter 1

Introduction

1.1 Rationale

The *PROCESS* systems code is being developed to provide an integrated and self-consistent treatment of the physics, engineering, economic, safety and environmental characteristics of tokamak fusion power plants. This will enable issues of the feasibility and safety advantages of different power plant designs to be systematically explored. This user's guide relates to a version of the code prior to the incorporation of the safety and environmental modules, and to improvements and extensions to the modelling in other areas.

During the course of studies of a proposed tokamak fusion power plant, there may be times when questions of the following type arise:

> Are the machine's physics and engineering parameters consistent with one another?
>
> Which machine of a given size and shape produces the cheapest electricity?
>
> What is the effect of a more optimistic limit on the plasma beta on the amount of auxiliary power required?

Questions such as these are extremely difficult to answer, since the large number of parameters involved are highly dependent on one another. Fortunately, computer programs have been written to address these issues, and *PROCESS* is one of them.

Suppose that an outline power plant design calls for a machine with a given size and shape, which will produce a certain net electric power. There may be a vast number of different conceptual machines that satisfy the problem as stated so far, and *PROCESS* can be used in non-optimisation mode to find one of these whose physics and engineering parameters are self-consistent. However, the machine found by *PROCESS* in this manner may not be possible to build in practice — the coils may be overstressed, for instance, or the plasma pressure may exceed the maximum possible value. *PROCESS* contains a large number of constraints to prevent the code from finding a machine with such problems, and running the code in optimisation mode forces these constraints to be met. The number of possible conceptual machines is thus considerably reduced, and optimisation of the parameters with respect to (say) the cost of electricity will reduce this number to a minimum (possibly one).

Formally then, *PROCESS* is a systems code that calculates in a self-consistent manner the parameters of a tokamak fusion power plant with a specified performance, ensuring that its operating limits are not violated, and with the option to optimise a given function of these parameters.

It would not be fair to call *PROCESS* a fusion power plant design code, as this implies that a great deal of complexity would need to be present in each and every model describing one of the tokamak systems. Such complexity is, however, incompatible with the code's iterative approach to solving the optimisation problem, since this requires repeated evaluation of the same (large number of) expressions. This is not to say that the models employed by the code are oversimplified — in general they represent good numerical estimates of present theoretical understanding, or are fits to experimental data. *PROCESS* provides a useful overall description of how a conceptual and feasible power plant may look.

1.2 History

PROCESS is derived from several earlier systems codes, but is largely based on the TETRA (Tokamak Engineering Test Reactor Analysis) code [1] and its descendant STORAC (Spherical TOrus Reactor Analysis Code) [2], which includes routines relevant to the tight aspect ratio class of tokamaks. These codes, and much of the original version of *PROCESS* itself, were written by personnel at Oak Ridge National Laboratory in Tennessee, USA, with contributions from a number of other laboratories in the USA. In addition, many of the mathematical routines have been taken from a number of different well-established source libraries.

Since the code is descended from such a wide range of sources, its structure was initially not ideal from the programmer's viewpoint. Non-standard practices and inconsistent layout within the code could have led to difficulties in modifying, interpreting and indeed running the code. A great deal of effort has therefore been expended at Culham since the code's arrival from ORNL to improve this situation, with the code being given a complete but careful upgrade, routine by routine. A *single* master copy of *PROCESS* now exists, the details of which are described here. The culmination of the work to improve the usability of the code is this user's guide, which hopefully will be of assistance to all users of *PROCESS*, whether they are planning to modify or run the code, or are simply trying to understand what the code aims to achieve.

As with all active research codes, *PROCESS* will continue to be developed for some time. As explained earlier in this introduction, the code will be used as the basis of power plant environmental and safety studies by the inclusion of further models and constraints.

1.3 Layout of the User's Guide

This user's guide is divided into a small number of logically separate units, each one of which provides specific information on a given topic. It depends on the user's motive for referring to the manual as to which chapter will be the most useful, although hopefully the style and structure adopted will allow one to browse through without difficulty.

Chapter 2 provides an overview of the program and the machine that is modelled by it. Chapter 3 goes into slightly more detail, and discusses the various physics and engineering models that are used within the code to describe the tokamak power plant systems. Chapter 4 describes how to run the program from scratch, and provides a number of hints and suggestions to bear in mind when the code does not find a feasible machine. Chapter 5 shows how to modify the code in specific ways, for example how extra constraints and variables should be added to the code. Appendices A and B contain example input files for *PROCESS* in non-optimisation and optimisation modes, respectively. Finally, Appendix C contains references for useful Work File Notes [15] that provide information about the code status, its location, and other details relating to the implementation of *PROCESS* to date.

This manual has been written in such a way as to (a) lead a new user of *PROCESS* into a clear understanding of the code's concepts, structure and models, and (b) help a more experienced user to set up and run the code efficiently and quickly. Potential users of *PROCESS* need only a basic knowledge of potential tokamak fusion power plants, and access to the code itself. No specialised knowledge in computers or computing is required.

Chapter 2

Program Overview — The Fundamentals

This chapter presents the reader with a first glimpse into the world of *PROCESS*. It is clearly important when one is trying to get to grips with a code of the proportions of *PROCESS* to be able to visualise the main aspects of the device being simulated at an early stage, without too much detail obscuring the overall picture. Similarly it is important to provide the potential users with an overview of the code structure and its operation before they embark on a closer study. The aim of this chapter is to aid this visualisation by presenting a brief description of the machine, its subsystems and the code layout in simple, general terms. Chapter 3 will provide users with more detail about how to customise the models and parameters mentioned here.

2.1 The Machine

A natural starting point for a systems code manual is the description of the system itself. In this case, of course, the system is a tokamak fusion power plant, modelled using a large number of equations based on knowledge of the underlying physics and engineering models of each subsystem of the machine. As will be emphasised later the bulk of the program is ordered into modules roughly corresponding to each of the subsystems, so an early attempt at familiarising the reader with them will hopefully be of some benefit.

2.1.1 Radial and Vertical Build

Figure 2.1 shows schematically the layout of a typical tokamak as modelled by *PROCESS*. This is the so-called 'build' of the machine — the relative locations of the major components. Their positions are referenced to the (R, Z) coordinate system, where R is the radial distance from the vertical centreline (axis) of the torus, and Z is the vertical distance from the equatorial midplane, about which the machine is up-down symmetrical. Components are often referred to as being 'inboard' or 'outboard', which simply means that they lie at a radius R less than or greater than R_0, respectively, where R_0 is the plasma major radius (`rmajor`).

Figure 2.2 shows the `FORTRAN` variables that describe the thicknesses and positions involved. It must be emphasised that these two figures are very much schematic, otherwise they could become slightly misleading. For the sake of clarity the thicknesses are not drawn to scale, and the space labelled as the divertor does not indicate in any way the actual shape of that component. The cryostat acts as a dewar or vacuum vessel, and is therefore continuous in reality, enclosing all of the components within it. Only in the code's build calculation is there an apparent gap in the cryostat beneath the top of the TF coil.

Most of the thicknesses shown in Figure 2.2 are input parameters, so are not changed during the course of the simulation. The rest are calculated by the code during execution. In addition, some of the component sizes can be used as *iteration variables* (see Section 4.1.4) to help in the optimisation process.

2.1.2 Principal Components

2.1.2.1 Plasma

Arguably, the most important component of the tokamak is the plasma itself. This is assumed to have an up-down symmetric, double null configuration, with elongation and triangularity specified by the user. A great number of physics models are coded within *PROCESS* describing the behaviour of the plasma parameters such as its current, temperature, density, pressure, confinement etc., and also the various limits that define the stable operating domain.

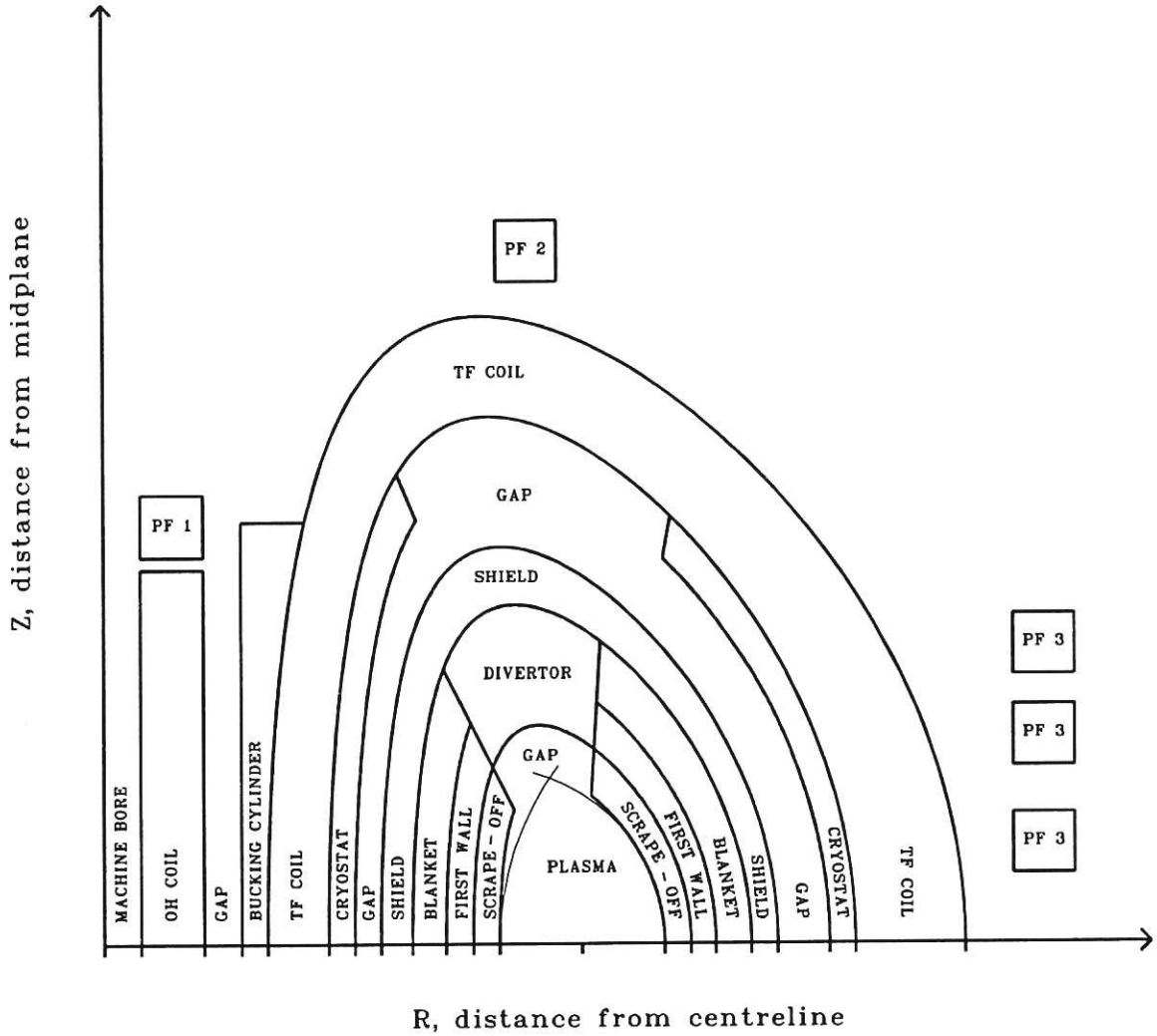

Figure 2.1: *Schematic diagram of a typical tokamak modelled by PROCESS. The relative positions of the components are as shown, assuming that switch* `iohcie` *is set to* 2. *If* `iohcie` *is set to* 1, *the inboard portion of the radial build is altered so that, starting at the centreline (*$R = 0$*), the component order is: bucking cylinder, TF coil, gap, OH coil, cryostat, etc. In this case there is no machine bore. The numbers shown in the PF coil blocks are the values of the* `ipfloc` *switch for that block.*

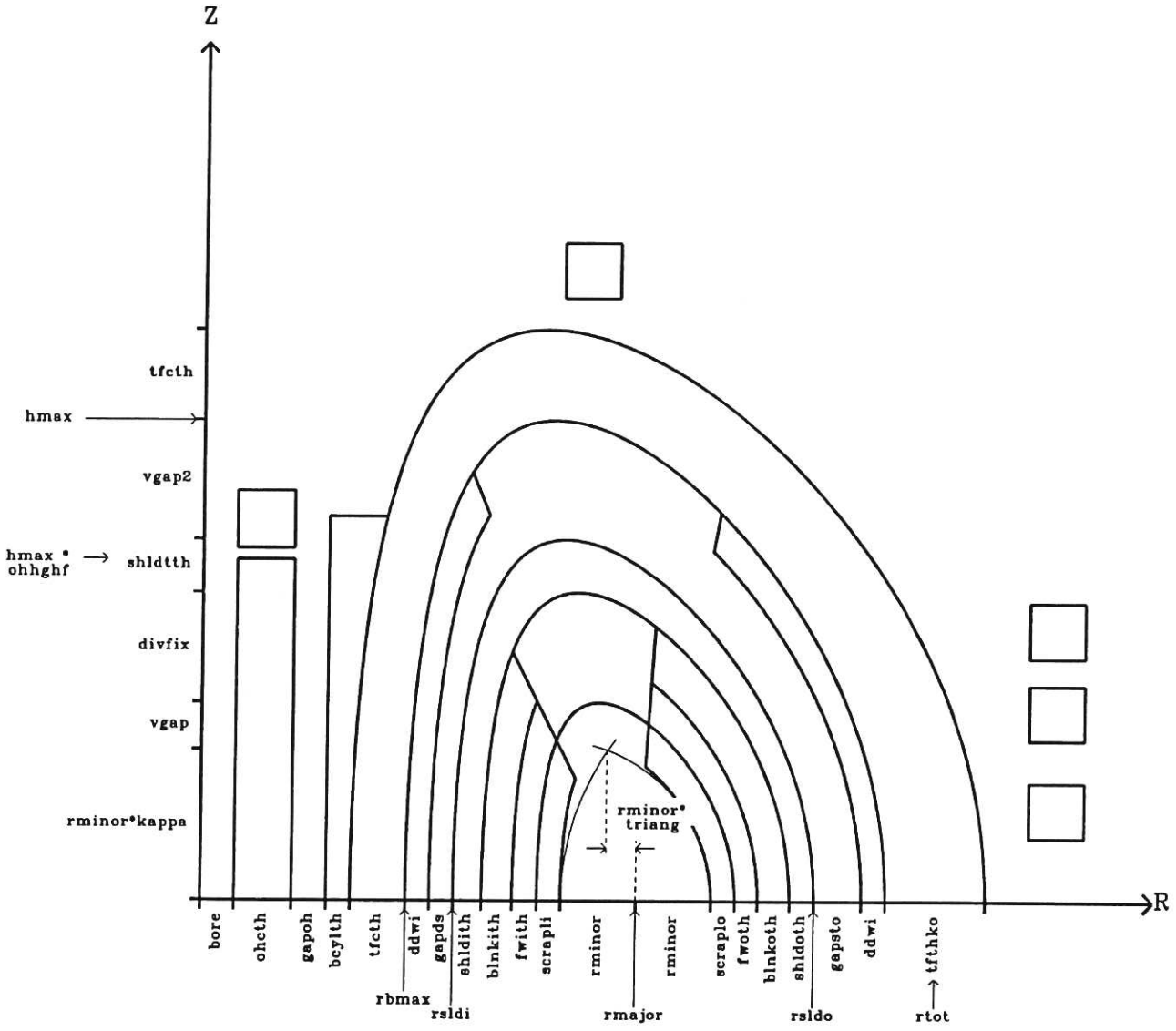

Figure 2.2: *Schematic diagram of a typical tokamak modelled by PROCESS, showing the variables used to define the thicknesses of the components. The arrowed labels adjacent to the axes are the total 'builds' to that point. As in Figure 2.1, the relative positions are shown for the case when* `iohcie = 2`. *The precise locations and sizes of the PF coils are calculated within the code.*

2.1.2.2 Scrape-off Layer

The region directly outside the last closed flux surface of the plasma is known as the scrape-off layer, and contains no structural material. Plasma entering this region is not confined and is removed by the divertor. *PROCESS* treats the scrape-off layer merely as a gap.

2.1.2.3 First Wall

The first wall acts as a physical barrier protecting the rest of the tokamak from the hot plasma. Due to its hostile environment the first wall has only a short lifetime and therefore needs to be replaced regularly. Its stainless steel structure is water cooled.

2.1.2.4 Divertor

The divertor provides a means of removing plasma reaching the scrape-off layer and heavy ions that are ejected from the first wall. Two divertors are assumed in the *PROCESS* tokamak, placed symmetrically above and below the plasma (N. B. see Section 3.2.6.3). The principal outputs from the code are the divertor heat load, used to determine its lifetime, and its peak temperature. The divertor is water cooled.

2.1.2.5 Blanket

The blanket performs a number of tasks. An incoming neutron from a deuterium-tritium (D-T) fusion reaction in the plasma loses energy in the blanket. This energy is removed by the blanket coolant and used to produce electricity. The neutron may also react with a lithium nucleus present in the blanket to produce a tritium nucleus which can be re-used as fuel. The competing requirements of heating and tritium synthesis mean that a neutron multiplier must be present, to ensure balance between tritium destruction and creation. The blanket therefore contains beryllium to fulfil this purpose. Again, the blanket has a relatively short lifetime because of the high neutron fluence. Steel is used as a structural material within the blanket.

2.1.2.6 Shield

The shield reduces the neutron flux reaching the TF coils and beyond. This minimises the radiological impact of the neutrons, and their heating of the TF coils which, if superconducting, need to remain at liquid helium temperatures. The shield is water cooled, and as with the blanket the energy deposited in the water is used to produce electricity.

2.1.2.7 TF Coils

The toroidal field (TF) coils can be either resistive or superconducting. In the superconductor model, the CICC (Conductor In Cable Conduit) structure shown in Figure 2.3 is used, and the coils are cooled using a liquid helium cryogenic system. Among the TF coil parameters calculated by the code are the maximum allowable current density, the stresses on the structure, the energy stored and the magnetic field produced by the coils.

The current in the TF coils must be sufficient to produce the right toroidal field at the centre of the plasma. The field falls off at a rate $1/R$, with the peak value occurring at the outer edge of the inboard portion of the TF coil ($R_{\text{max TF}}$ = `rbmax`). The maximum TF coil current depends on the field it produces and the allowable current density.

Each TF coil is defined in the (R, Z) plane by four circular arcs of different radius, which create a D-shaped profile. Because of the finite number of TF coils used in a tokamak (typically around 20), the toroidal field has a ripple introduced into it, the amplitude of which can be limited to a few per cent by the code by adjusting the outboard gap thickness (labelled `gapsto` in Figure 2.2). Ports are often necessary for auxiliary power systems etc., and the gaps between adjacent TF coils can be made large enough to accommodate such equipment.

2.1.2.8 Bucking Cylinder

The bucking cylinder provides some strength to the inboard TF coil structure. If the TF coils are superconducting, the bucking cylinder is cooled by the cryogenic system.

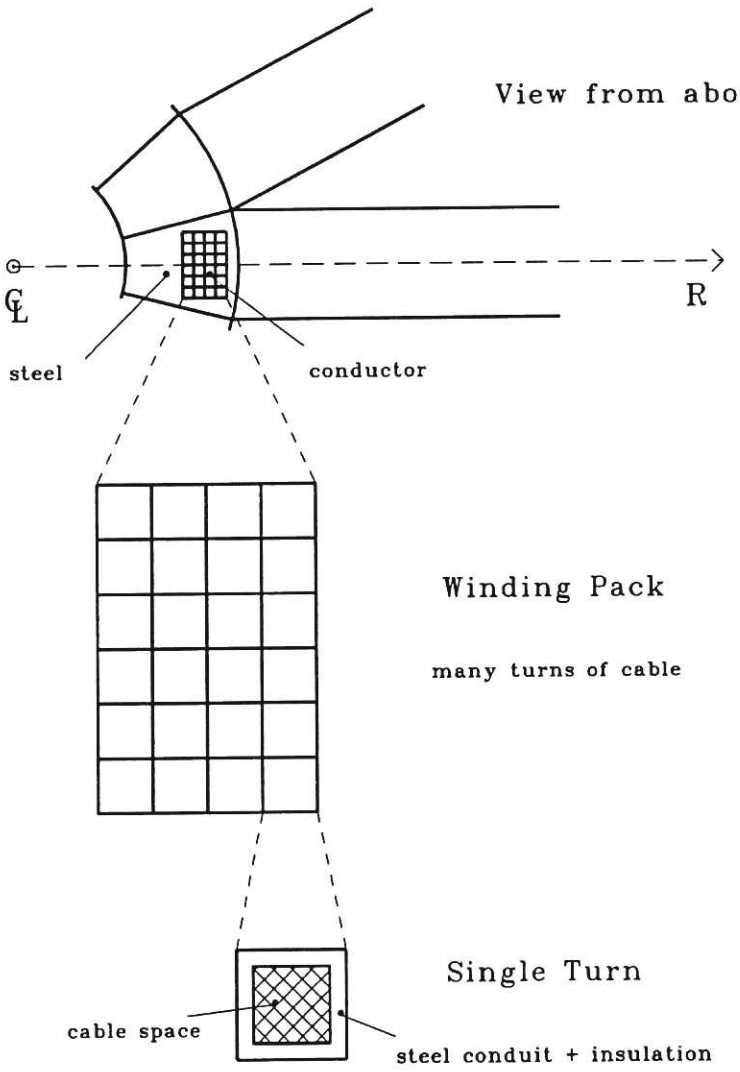

Figure 2.3: *Schematic diagram of the cross-section of a superconducting TF coil, showing the CICC (Conductor In Cable Conduit) construction. The cable space contains superconducting filaments and circulating liquid helium coolant.*

2.1.2.9 Cryostat

The cryostat acts as a dewar and vacuum vessel, and is used to cool those components that need to operate at liquid helium temperatures. These include any superconducting (TF or PF) coils, the intercoil structure and the bucking cylinder. *PROCESS* calculates the cryogenic power load and the resulting heat exchanger requirements. As stated earlier, the build picture in Figure 2.1 is slightly misleading in that the cryostat completely encloses the components within it — there is no large gap beneath the TF coil as is the case in Figure 2.1.

In addition to this (inner) cryostat, an external cylindrical dewar encloses the whole of the tokamak.

2.1.2.10 PF Coils

The poloidal field (PF) coils can be either resistive or superconducting, and are used initially to cancel the vertical field produced at the centre of the plasma by the OH coil during start-up, and then to maintain the plasma position and shape during the flat-top period. The positions and sizes of the PF coils are partly input, and partly calculated after consideration of the required currents and allowable current density. The PF coils are generally grouped into up-down symmetric pairs, and each group can be placed in one of three regions, either vertically above (and below) the OH coil (`ipfloc(i)=1`), vertically above (and below) the TF coil (`ipfloc(i)=2`), or radially outside the TF coil (`ipfloc(i)=3`).

The PF coil currents vary as a function of time during the tokamak operation as indicated in Figure 2.4.

2.1.2.11 OH Coil

The ohmic heating (OH) coil is a PF coil used primarily during start-up (but also during the burn phase) to create and maintain the plasma current by inductive means. Swinging (changing) the current through the OH coil causes a change in the flux linked to the plasma region, inducing a current in it. *PROCESS* calculates the amount of flux required to produce the plasma current, and also the amount actually available. The code measures the magnetic flux in units of Volt-seconds (= Webers). The OH coil is sometimes referred to as the central solenoid, and can be either resistive or superconducting.

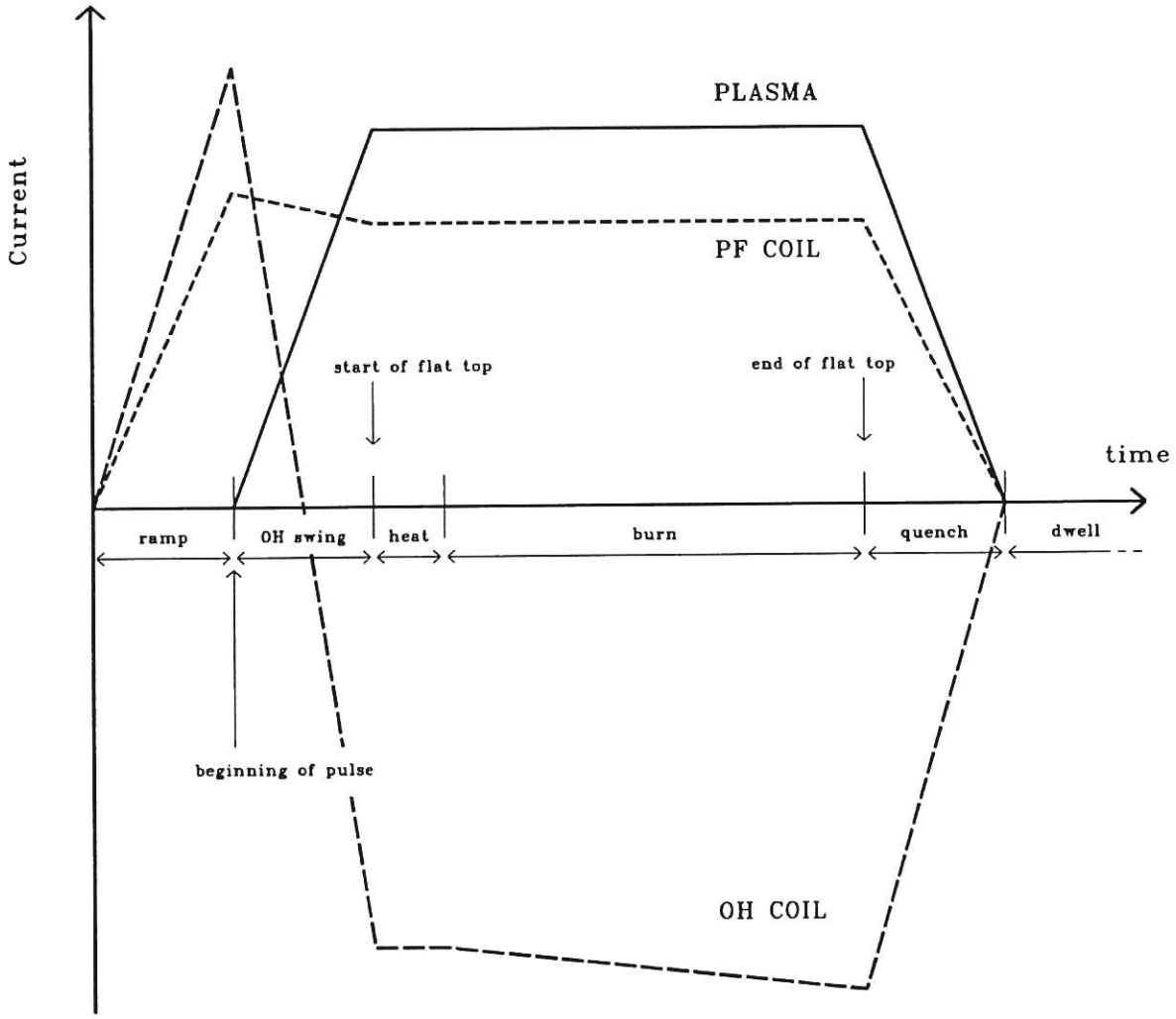

Figure 2.4: *Plot showing schematically the current waveforms for the plasma, a typical PF coil, and the OH coil.*

2.1.2.12 Auxiliary Power Systems

The use of purely inductive current drive leads to pulsed plant operation because of the limited flux swing that can be achieved using the OH coil. This poses problems due to the fact that fatigue failures may result, and there would also be a need for thermal storage to maintain a level supply between pulses. However, the plasma current can also be produced and maintained using non-inductive means which, in principle, removes this restriction. *PROCESS* contains a number of auxiliary current drive schemes, including various RF methods (Lower Hybrid, Electron Cyclotron, and Ion Cyclotron (Fast Wave) current drives) and also Neutral Beam current drive systems. The code calculates the efficiency and the resulting power requirements of the chosen system.

2.1.2.13 Structural Components

Structural components are required to provide support for the tokamak systems against gravity and the magnetic forces that will be encountered during operation. The required structural masses and their costs are calculated.

2.1.2.14 Power Conversion and Heat Dissipation Systems

The *PROCESS* power plant takes into account all the systems required to perform the necessary conversion of fusion power to electricity, from the coolant systems in the plant components to the heat exchangers and turbines. Figure 2.5 shows schematically the overall power transfer mechanisms used by the code.

2.1.2.15 Vacuum Systems

The vacuum system is used for four different processes. Firstly, before plasma operations the chamber must be evacuated to remove outgassed impurities from the structure. Secondly, the chamber must be re-evacuated between burn operations. Thirdly, helium ash must be removed to prevent it from diluting the fuel. Finally, deuterium and tritium is removed on a steady state basis. *PROCESS* calculates the parameters of a vacuum system that satisfy all four requirements, with the option of either turbo pumps or cryo pumps being used.

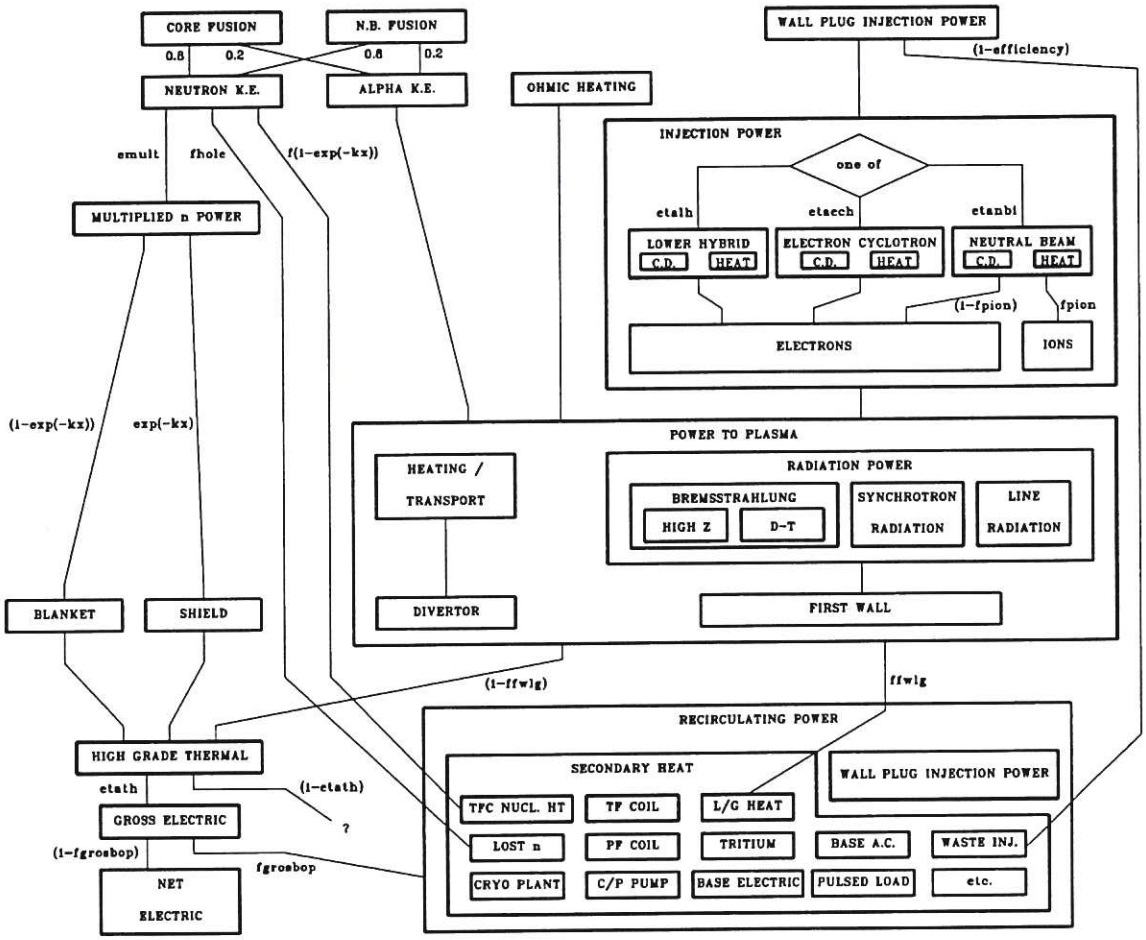

Figure 2.5: *Schematic diagram showing the power conversion mechanisms used in PROCESS [15, Note 0166].*

2.1.2.16 Buildings

The volume and ground area of all the various buildings on a power plant site are included in the *PROCESS* calculations for the benefit of the costing algorithms.

2.1.3 Tight Aspect Ratio Tokamaks

PROCESS has the ability to perform studies on tokamaks in the low aspect ratio regime (major radius $\leq 2\times$ minor radius). The physics and engineering issues [3] associated with these machines is somewhat different from those of conventional aspect ratio, and this is reflected by the following special models [2] present in *PROCESS*.

1. The inboard build of a TART is very different from that in a conventional tokamak. There is no inboard blanket or shield, and the inboard TF coil legs are replaced by a single centrepost.
2. The centrepost is constructed from copper (as are the outboard TF coil sections), and is tapered so that it is narrowest at the midplane of the device. The parameters for the centrepost coolant system are calculated by *PROCESS*, including the pump pressure, maximum temperature and pipe radius.
3. A gaseous divertor model is used, and a simple divertor heat load calculation is employed.
4. A simple PF coil current scaling algorithm is available for use with the TART option.
5. The plasma shaping terms (elongation and triangularity) can be calculated directly from the given aspect ratio.
6. Among the physics models that differ from those relevant to conventional aspect ratio machines are (i) the bootstrap current fraction, (ii) the Troyon beta limit, and (iii) the neutron heating of the centrepost.

Since *PROCESS* is set up by default to deal with conventional aspect ratio machines, TART power plants will only be referred to briefly in the rest of this manual.

2.2 The Code

As hinted above, tokamaks are complex devices consisting of many non-linear interactions. One method that can be used to model this kind of system is to iterate a number of free parameters in a controlled way so as to find a self-consistent set of device parameters that satisfy all of the system's constraints. *PROCESS* is organised in a standard equation solver format to enable this task to be performed efficiently. The physics and engineering routines together serve as a *function evaluator*, providing the information used in the solution of the constraints. The numerical software package present in *PROCESS* performs the iteration required, and also incorporates the option to maximise or minimise a given figure of merit.

2.2.1 Code Usage

PROCESS contains two non-linear equation solver packages, which reflect the two major modes of operation available. Each of these has its own uses, as is now discussed.

2.2.1.1 Non-Optimisation Mode

The first of the two equation solvers present in *PROCESS* is the non-optimisation package HYBRID [4, 5]. Formally, HYBRID finds a zero of a system of N non-linear functions in N variables. This means simply that N variables (tokamak parameters) are iterated by *PROCESS* in such a way as to solve a set of N equations (physics or engineering laws), i.e. a set of self-consistent tokamak parameters are found. This is useful for performing benchmark comparisons, when the device size is kept fixed, and one only wishes to find calculated stresses, beta values, fusion powers, etc. A flow diagram of *PROCESS* in non-optimisation mode is shown in Figure 2.6.

2.2.1.2 Optimisation Mode

The HYBRID equation solver will naturally find only one of perhaps several possible machines that may satisfy the prescribed problem. To choose one machine in preference to the others it is necessary to define a figure of merit, and the selection process then simply involves finding the machine parameters that maximise or minimise this figure of merit.

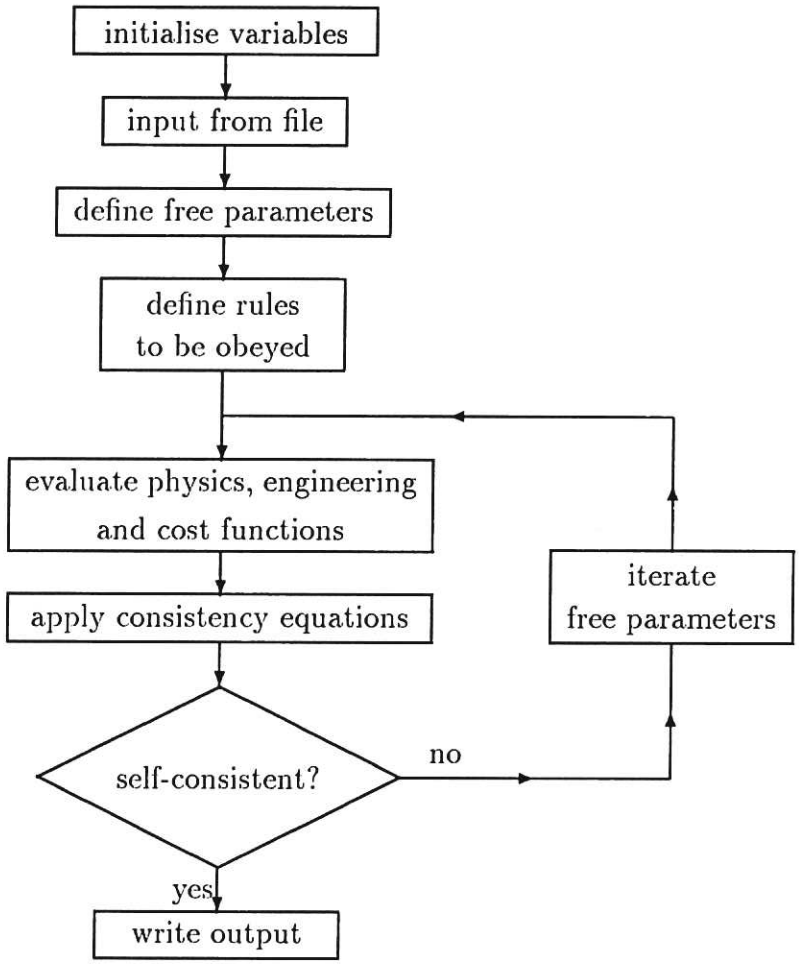

Figure 2.6: *Flow diagram of PROCESS in non-optimisation mode.*

The second equation solver within *PROCESS*, VMCON [6], performs this optimisation, and therefore finds the "best" machine that satisfies all the given constraints.

The other important advantage that VMCON has over HYBRID is its ability to limit the ranges of the variables it uses. This prevents the code from attempting to find machines that are physically unattainable, and ensures that operating limits are not violated. An example of VMCON's application is to find the device providing the minimum cost of electricity which also satisfies the physics and engineering constraints. There is in theory no upper limit to the number of variables that VMCON can use to optimise the machine, so a very large region of parameter space can be searched. A flow diagram of *PROCESS* in optimisation mode is shown in Figure 2.7.

2.2.1.3 Scans

It is often useful to be able to scan through a range of values of a given parameter to see what effect this has on the machine as a whole. Sensitivity studies of this kind can be achieved very easily using *PROCESS*. Scans are carried out in optimisation mode, whereby the code performs initially a run using the parameters specified in the input file, and then a series of runs using the parameters produced at the end of the previous iteration. The variable being scanned is specified at every stage. This method ensures that a smooth variation in the machine parameters is achieved.

2.2.2 Code Structure

The structure of the majority of the code reflects to a certain extent the layout of the machine being modelled. As stated above, a large proportion of the code is simply a description of the underlying physics and engineering issues in terms of numerous expressions and relationships. In effect these define the machine so that the numerical solver within the code can then get to work adjusting the parameters in its search for a self-consistent solution.

A modular structure extends throughout the code, from the actual `FORTRAN` source files (suffixed with `.f`) to the input / initialisation routines and variable descriptor file sections. It is essential for a program of the size and complexity of *PROCESS* to be modular to a high degree, in order to simplify the tasks of understanding and maintaining the code. The `INCLUDE` files (suffixed with `.h`) contain `COMMON` blocks that pass the information for a given module between the various relevant subroutines, and are summarised in Table 2.1. The following sections describe briefly the modules into which *PROCESS* is divided.

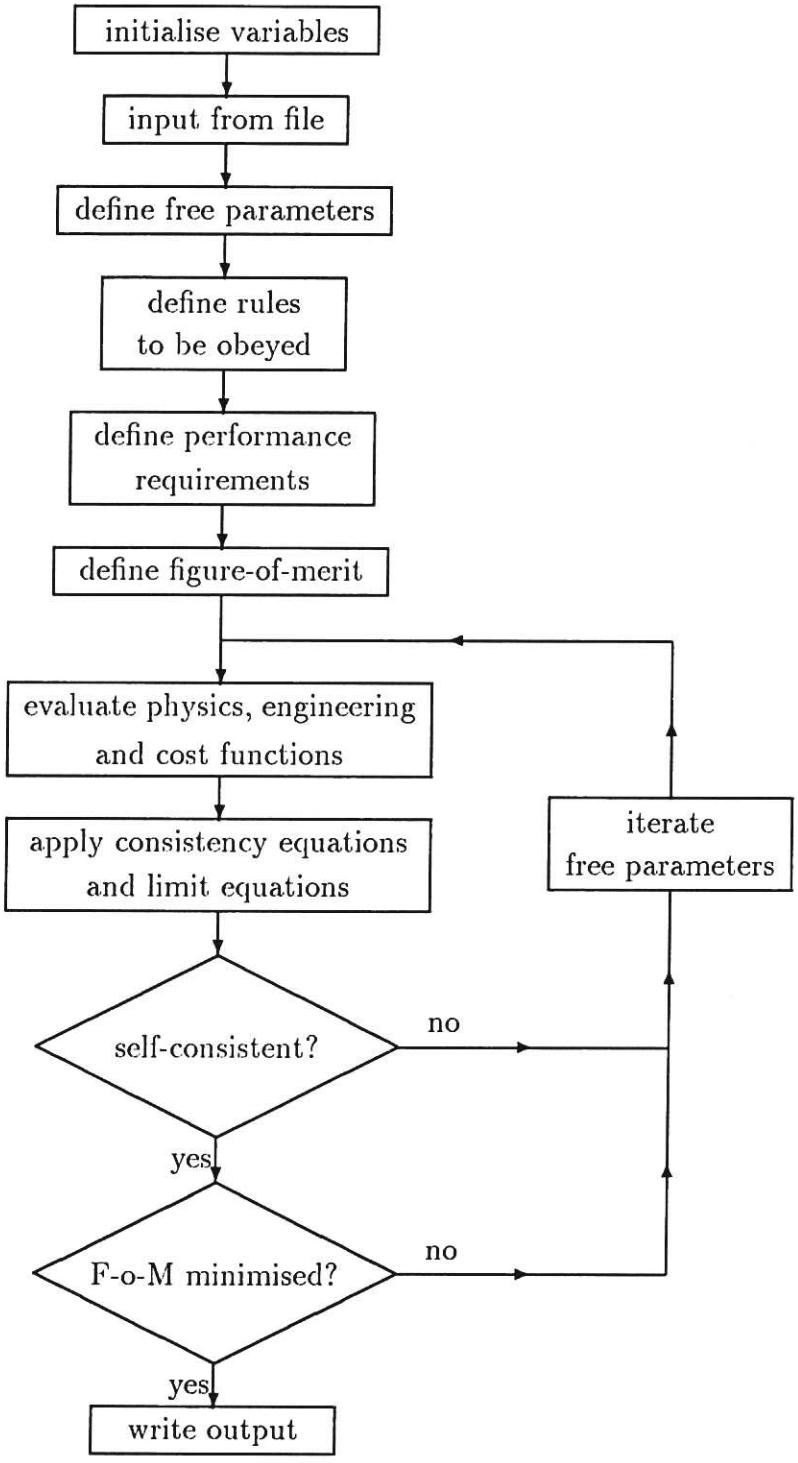

Figure 2.7: *Flow diagram of PROCESS in optimisation mode.*

include file	associated input block	stored parameters
bldgcom.h	BCOM	building volumes and clearances
bldgvol.h	BLDINP	building volumes
build.h	BLD	tokamak radial and vertical builds
cdriv.h	CDDAT	current drive quantities
cost.h	COSTINP, UCSTINP	cost information
divrt.h	DIVT	divertor parameters
estocom.h	EST	energy storage data
fwblsh.h	FWBLSH	first wall, blanket and shield data
heatrinp.h	HTRINP	heat transport input parameters
heattr.h	HTTINP	heat transport parameters
htpwr.h	HTPWR	heat transport and power data
ineq.h	INEQDAT	f-values and input limits
labels.h	—	descriptions
numer.h	INPT1	numerics quantities
osections.h	OSECTS	output section flags
param.h	—	global dimensioning parameters
pfcoil.h	PFC	PF coil data
pfelect.h	—	peak MVA requirements
phydat.h	PHYDAT	physics parameters
pwrcom.h	—	power conversion data
struccom.h	—	structural masses
sweep.h	SWEP	scan variable data
tfcoil.h	TFC	TF coil data
times.h	TIME	times of different plasma phases
torsdat.h	—	vacuum pump information
vaccom.h	VACCY	vacuum system parameters
vltcom.h	VOLTS	volt-second and inductance data

Table 2.1: *Summary of the* INCLUDE *files used within PROCESS. Each file has an associated named block in the input file, as shown.*

source file	description
aamain.f	main program
eqns.f	constraint equations
caller.f	calls physics and engineering routines
eqsolv.f	calls HYBRID non-optimising package
math.f	miscellaneous maths routines
math2.f	miscellaneous Linpack maths routines
minpac.f	various Minpack routines, including VMCON and HYBRID
optimiz.f	calls VMCON optimising package
svd.f	singular value decomposition routine
xc.f	adjusts iteration variables

Table 2.2: *Summary of the numerics modules in PROCESS.*

2.2.2.1 Numerics Modules

```
eqns.f, caller.f, eqsolv.f, math.f, math2.f, minpac.f,
eqsolv.f, optimiz.f, svd.f, xc.f
param.h, ineq.h, labels.h, numer.h, sweep.h
```

These files contain the equation solvers, their calling routines and other relevant procedures, including the main program itself. Various mathematical routines from a number of standard libraries are also incorporated into these files. Table 2.2 summarises the numerics source file contents.

2.2.2.2 Physics Modules

```
physics.f, cudriv.f, beams.f, lwhymod.f, ech.f, divtmod.f,
geomty.f, outplas.f
phydat.h, cdriv.h, divrt.h
```

These files contain the main physics routines that evaluate the plasma and fusion parameters. Also included here are the routines describing the current drive and divertor systems. Table 2.3 summarises the physics source file contents.

source file	description
physics.f	plasma and fusion calculations
cudriv.f	current drive efficiency parameters
beams.f	neutral beam wall plug power
lwhymod.f	lower hybrid wall plug power
ech.f	electron cyclotron CD wall plug power
divtmod.f	divertor model calculations
geomty.f	plasma geometry algorithms
outplas.f	writes physics output to file

Table 2.3: *Summary of the physics modules in PROCESS.*

2.2.2.3 Engineering Modules

```
radialb.f, tfcoil.f, induct.f, fwbs.f, heatpwr.f, acpow.f,
pfcoil.f, pfscl.f, pwrconv.f, sctfcoil.f, struct.f, supercond.f,
tfcpwr.f, bldgs.f, vacuum.f
bldgcom.h, bldgvol.h, build.h, estocom.h, fwblsh.h, heatrinp.h,
heattr.h, htpwr.h, pfcoil.h, pfelect.h, pwrcom.h, struccom.h,
tfcoil.h, times.h, torsdat.h, vaccom.h, vltcom.h
```

These files contain the description of the machine geometry and its major systems, including the PF and TF coil sets, the first wall, blanket and shield, and other items such as the buildings, vacuum system, power conversion and the structural components. Table 2.4 summarises the engineering source file contents.

2.2.2.4 Costing Modules

```
costs.f
cost.h
```

These files perform all the cost calculations, including values in M$ for each machine system, and the cost of electricity in m$/kWh.

source file	description
radialb.f	machine build
fwbs.f	first wall, blanket and shield
pfcoil.f	PF coil calculations
pfscl.f	PF coil current scaling method
induct.f	inductance calculations
tfcoil.f	TF coil calculations
sctfcoil.f	superconducting TF coil calculations
supercond.f	superconducting coil limits
tfcpwr.f	superconducting TF coil power conversion
pwrconv.f	TF and PF coil power conversion
heatpwr.f	heat transport and power balances
acpow.f	total plant power needs
struct.f	support structure calculations
bldgs.f	buildings calculations
vacuum.f	vacuum system calculations

Table 2.4: *Summary of the engineering modules in PROCESS.*

2.2.2.5 Other Modules

```
aamain.f, initial.f, input.f, output.f
osections.h
```

These files perform miscellaneous tasks, such as initialisation of variables and file input / output. File `aamain.f` contains the main program, and includes the overall controlling loop.

2.2.3 Variable Descriptor File

The variable descriptor file, `var.des`, contains a great deal of information of value to the user of *PROCESS*. Every variable contained in the `INCLUDE` files is described in detail, including the meanings of all the switch settings, the default values of those parameters not calculated in the code, and the units that quantities are measured in. The file is ordered into sections that again reflect the modular structure of *PROCESS*, and by referring to it the user can instantly see which variables are capable of being changed in the input file.

Chapter 3

Physics and Engineering Models

There are a great number of individual models within *PROCESS*, characterising many different aspects of a tokamak power plant. Several of these will always be used by the code and so require no input by the user to activate them. However, in many cases there is a choice of model available, and each of these has its own user-controlled switches or flags. This chapter summarises these models, and indicates their location and interaction within the code, together with the relevant switch settings and required parameter values. The nomenclature used, and instructions on how to set switches, etc., are explained fully in Chapter 4.

3.1 Physics Models

3.1.1 Plasma Profiles

All the plasma profiles are assumed to be parabolic, that is, of the form

$$\text{Density} : n(r) = n_0 \left(1 - \left(\frac{r}{a}\right)^2\right)^{\alpha_n} \tag{3.1}$$

$$\text{Temperature} : T(r) = T_0 \left(1 - \left(\frac{r}{a}\right)^2\right)^{\alpha_T} \tag{3.2}$$

$$\text{Current} : J(r) = J_0 \left(1 - \left(\frac{r}{a}\right)^2\right)^{\alpha_J} \tag{3.3}$$

where r varies from 0 to a, the plasma minor radius. This gives volume-averaged values $\langle n \rangle = n_0/(1+\alpha_n)$, etc. These volume averages are used throughout the code, along with the profile indices α, thus making the code "$\frac{1}{2}$-D". The relevant profile index variables `alphan`, `alphat` and `alphaj` in the code are set in input block `PHYDAT`.

3.1.2 Beta Limits

3.1.2.1 Troyon limit on the total beta [8]

The Troyon beta limit is given by

$$\langle\beta\rangle < \frac{gI(\mathrm{MA})}{a(\mathrm{m})B_0(\mathrm{T})} \tag{3.4}$$

where B_0 is the axial vacuum toroidal field, and β is defined with respect to the total equilibrium **B**-field [7].

Flags :
- Set `dnbeta` in input block `PHYDAT` to the required coefficient g.
- Set `iculbl = 0` in input block `PHYDAT`
- Turn on constraint equation no. 24 with iteration variable no. 36 (`fbetatry`) — see Chapter 4.

Routines :
- `CULBLM` is called by `PHYSICS`

3.1.2.2 Troyon limit on the thermal beta [7]

The Troyon beta limit is given by Equation 3.4. To apply the limit to only the *thermal* component of the plasma beta, use the following:

Flags :
- Set `dnbeta` in input block `PHYDAT` to the required coefficient g.
- Set `iculbl = 1` in input block `PHYDAT`
- Turn on constraint equation no. 24 with iteration variable no. 36 (`fbetatry`)

Routines :
- `CULBLM` is called by `PHYSICS`

3.1.2.3 Limit on $\epsilon.\beta_p$ value

To apply a limit to the value of $\epsilon.\beta_p$, where $\epsilon = a/R$ is the inverse aspect ratio:-

Flags :
- Set `epbetmax` in input block `PHYDAT` to the required limiting value
- Turn on constraint equation no. 6 with iteration variable no. 8 (`fbeta`)

3.1.3 Density Limits

Six density limit models [7] are available in *PROCESS*. These are calculated in routine CULDLM, which is called by PHYSICS. To enforce any of these limits, set `iculdl = 1` in input block PHYDAT, and turn on constraint equation no. 5 with iteration variable no. 9 (fdene). In addition, flag `idensl` must be set to the relevant value, as follows:-

3.1.3.1 ASDEX model

Flags : • Set `idensl = 1` in input block PHYDAT

3.1.3.2 Borrass model for ITER, I

Flags : • Set `idensl = 2` in input block PHYDAT

3.1.3.3 Borrass model for ITER, II

Flags : • Set `idensl = 3` in input block PHYDAT

3.1.3.4 JET edge radiation model

Flags : • Set `idensl = 4` in input block PHYDAT

3.1.3.5 JET simplified model

Flags : • Set `idensl = 5` in input block PHYDAT

3.1.3.6 Hugill-Murakami $M.q$ model

Flags : • Set `idensl = 6` in input block PHYDAT

3.1.4 Plasma Current Scaling Laws

Seven plasma current scaling laws exploiting the inverse relationship between plasma current and edge safety factor q_ψ [7] are available in *PROCESS*. These are calculated in routine CULCUR, which is called by PHYSICS. Flag iculcr must be set to 1 in input block PHYDAT. In addition, flag icurr must be set to the relevant value, as follows:-

3.1.4.1 Peng analytic fit

Flags : • Set icurr = 1 in input block PHYDAT

3.1.4.2 Peng double null divertor scaling (TART)

Flags : • Set icurr = 2 in input block PHYDAT

Routines : • PLASC is called by CULCUR

3.1.4.3 Simple ITER scaling

Flags : • Set icurr = 3 in input block PHYDAT

3.1.4.4 Revised ITER scaling

Flags : • Set icurr = 4 in input block PHYDAT

3.1.4.5 Todd empirical scaling I

Flags : • Set icurr = 5 in input block PHYDAT

3.1.4.6 Todd empirical scaling II

Flags : • Set icurr = 6 in input block PHYDAT

3.1.4.7 Connor-Hastie model

Flags : • Set icurr = 7 in input block PHYDAT

Routines : • CONHAS is called by CULCUR

3.1.5 Confinement Time Scaling Laws

No less than twenty energy confinement time scaling laws [7, 8, 9] are present within *PROCESS*. These are calculated in routine `PCOND`. The value of `isc` in input block `PHYDAT` determines which of the scalings is used in the plasma energy balance calculation. Table 3.1 summarises the available scaling laws.

Routines :

- `GAMFUN` is called by `PCOND`
- `PCOND` is called by `PHYSICS`, `FHZ` and `IGMARCAL`
- `FHZ` is called (indirectly) by `IGMARCAL`
- `IGMARCAL` is called by `OUTPUT`

isc	scaling law
1	Neo-Alcator (ohmic)
2	Mirnov (H-mode)
3	Merezhkin-Muhkovatov (L-mode)
4	Shimomura (H-mode)
5	Kaye-Goldston (L-mode)
6	ITER 89-P (L-mode)
7	ITER 89-O (L-mode)
8	Rebut-Lallia (L-mode)
9	Goldston (L-mode)
10	T10
11	JAERI-88
12	Kaye-Big Complex
13	ITER H90-P (H-mode)
14	ITER Mix (minimum of 6 and 7)
15	Riedel (L-mode)
16	Christiansen et al. (L-mode)
17	Lackner-Gottardi (L-mode)
18	Neo-Kaye (L-mode)
19	Riedel (H-mode)
20	ITER H90-P (amended)

Table 3.1: *Summary of the energy confinement time scaling laws in PROCESS. These laws are all cited in [7, 8] and references therein, with the exception of law 20 [9].*

3.1.6 Bootstrap Current Scalings

The fraction of the plasma current provided by the so-called bootstrap effect can be either input into the code directly, or calculated using one of three methods, as summarised here.

3.1.6.1 Direct input

Flags :
- Set `bscfmax` in input block `PHYDAT` to (-1) times the required bootstrap current fraction.

3.1.6.2 ITER scaling (not TART) [8]

Flags :
- Set `bscfmax` in input block `PHYDAT` to the maximum required bootstrap current fraction (≤ 1).
- Set `ibss = 1` in input block `PHYDAT`

Routines :
- `BOOTST` is called by `PHYSICS`

3.1.6.3 General scaling [10]

Flags :
- Set `bscfmax` in input block `PHYDAT` to the maximum required bootstrap current fraction (≤ 1).
- Set `ibss = 2` in input block `PHYDAT`

Routines :
- `BSINTEG` is called (indirectly) by `FNEWBS`
- `FNEWBS` is called by `PHYSICS`

3.1.6.4 Numerically fitted scaling [7]

Flags :
- Set `bscfmax` in input block `PHYDAT` to the maximum required bootstrap current fraction (≤ 1).
- Set `ibss = 3` in input block `PHYDAT`

Routines :
- `CULBST` is called by `PHYSICS`

3.1.7 Current Drive

In addition to inductive current drive, eight non-inductive current drive efficiency models [7] are present in *PROCESS*. The fraction of the volt-seconds to be produced by non-inductive means, `fvsbrnni`, should be set in input block `PHYDAT`, and flag `irfcd` in input block `CDDAT` should be set to `0` for purely inductive scenarios, or `1` otherwise. The current drive efficiency model to be used in this latter case is defined by the value of flag `iefrf` in input block `CDDAT`:-

3.1.7.1 Fenstermacher Lower Hybrid model

Flags : • Set `iefrf = 1` in input block `CDDAT`

Routines : • `CUDRIV` is called by `CALLER`

3.1.7.2 Ion cyclotron model [8]

Flags : • Set `iefrf = 2` in input block `CDDAT`

Routines : • `CUDRIV` is called by `CALLER`

3.1.7.3 Fenstermacher electron cyclotron resonance model

Flags : • Set `iefrf = 3` in input block `CDDAT`

Routines : • `CUDRIV` is called by `CALLER`

3.1.7.4 Ehst Lower Hybrid model

Flags : • Set `iefrf = 4` in input block `CDDAT`

Routines : • `CUDRIV` is called by `CALLER`

3.1.7.5 ITER neutral beam model [8, 7]

Flags : • Set `iefrf = 5` in input block `CDDAT`

Routines : • `XLMBDABI` is called by `CFNBI`

• `CFNBI`, `ETANB` and `SIGBEAM` are called by `ITERNB`

• `ITERNB` is called by `CUDRIV`

3.1.7.6 Culham Lower Hybrid model [7]

Flags :
- Set `iefrf = 6` in input block `CDDAT`

Routines :
- `LHEVAL` is called by `LHRAD`
- `LHRAD` is called by `CULLHY`
- `CULLHY` is called by `CUDRIV`

3.1.7.7 Culham electron cyclotron model [7]

Flags :
- Set `iefrf = 7` in input block `CDDAT`

Routines :
- `LEGEND` is called by `ECCDEF`
- `ECCDEF` is called by `CULECD`
- `CULECD` is called by `CUDRIV`

3.1.7.8 Culham neutral beam model [7]

Flags :
- Set `iefrf = 8` in input block `CDDAT`

Routines :
- `XLMBDABI` is called by `CFNBI`
- `CFNBI`, `GAMFUN`, `SIGBEAM` and `ETANB2` are called by `CULNBI`
- `CULNBI` is called by `CUDRIV`

It is sometimes useful to adjust artificially the current drive efficiency values produced by these routines. This can be achieved by setting the scaling coefficient `feffcd` in input block `CDDAT`. The wall plug to plasma efficiencies can also be adjusted, by changing the relevant variable `etaech`, `etalh` or `etanbi` in input block `CDDAT`.

3.1.8 Other Physics Switches

3.1.8.1 Plasma cross-sectional shape

Switch `ishape` in input block `PHYDAT` controls whether the the input values for the plasma elongation (`kappa`) and triangularity (`triang`) should be used (`ishape = 0`), or whether they should be scaled with the plasma aspect ratio (`ishape = 1`). The latter case should only be used with a TART machine.

3.1.8.2 Fusion power calculations

Switch `iiter` in input block `PHYDAT` controls which model for the fusion power calculations should be used. If `iiter = 1`, the ITER model [8] is used.

3.1.8.3 Neo-classical correction effects

Switch `ires` in input block `PHYDAT` controls whether neo-classical correction effects are included in the calculation of ohmic heating power in routine `POHM`, which is called by routine `PHYSICS`. If `ires = 1`, these effects are included.

3.1.8.4 Aspect ratio scaling of Troyon g coefficient

Switch `gtscale` in input block `PHYDAT` determines whether the Troyon g coefficient `dnbeta` (Equation 3.4) should scale with aspect ratio (`gtscale` $\neq$ `0`), or be fixed at the input value (`gtscale = 0`).

3.1.8.5 Inverse quadratic in τ_E scaling laws

Switch `iinvqd` in input block `PHYDAT` determines whether the energy confinement time scaling laws due to Kaye-Goldston (`isc = 5`) and Goldston (`isc = 9`) should include an inverse quadratic scaling with the Neo-Alcator result (`isc = 1`). A value `iinvqd = 1` includes this scaling.

3.2 Engineering Models

3.2.1 TF Coil Position

The relative radial positions of the TF coil inboard legs and the OH coil are controlled using flag `iohcie` in input block `BLD`. If this is set to 2, then the relative positions are as shown in Figure 2.1. If, however, `iohcie = 1`, the radial build is altered so that, starting from the centreline ($R = 0$), the component order is: bucking cylinder, TF coil, gap, OH coil, cryostat, and then continuing as in Figure 2.1. This latter case is generally only used for TART machines, which have a zero thickness machine bore, thereby creating a single centrepost out of the inner TF coil legs.

3.2.2 TF Coil Type

The TF coils can be either resistive (copper) or superconducting. The following switches control the options available for the different TF coil types.

3.2.2.1 Copper centrepost

Flags :
- Set `itart = 1` in input block `PHYDAT`
- Set `itfsup = 0` in input block `TFC`

Routines :
- `CNTRPST` is called by `CALLER` and `OUTPUT`
- `TFCOIL` is called by `CALLER`
- `FWBS` is called by `CALLER` and `OUTPUT`

3.2.2.2 Superconducting TF coils

Flags :
- Set `itart = 0` in input block `PHYDAT`
- Set `itfsup = 1` in input block `TFC`

Routines :
- `COILSHAP`, `TFCIND`, `STRESSCL` and `OUTTF` are called by `SCTFCOIL`
- `SCTFCOIL` is called by `TFCOIL`
- `PROTECT` is called by `SUPERCON`
- `SUPERCON` is called by `TFSPCALL`
- `TFSPCALL` is called by `CALLER` and `OUTPUT`

3.2.2.3 Resistive TF coils (not TART)

Flags :
- Set `itart = 0` in input block `PHYDAT`
- Set `itfsup = 2` in input block `TFC`

Routines :
- `CONCOPTF` is called by `TFCOIL`

3.2.3 Superconducting TF Coil Options

The following options are available within the superconducting TF coil model, for which flags `itart = 0` and `itfsup = 1`.

3.2.3.1 Stress model

Switch `itfmod` in input block `TFC` controls whether a simple or more complex stress model should be used.
Simple stress model:

Flags :
- Set `itfmod = 0` in input block `TFC`

Routines :
- `SCTFJALW` is called by `STRESSCL`
- `STRESSCL` is called by `SCTFCOIL`

Complex stress model:

Flags :
- Set `itfmod = 1` in input block `TFC`

Routines :
- `EYNGEFF`, `TFSTRESS` and `SIGVM` are called by `STRESSCL`
- `STRESSCL` is called by `SCTFCOIL`

3.2.3.2 Superconducting materials

Three superconducting materials are presently available within *PROCESS*. The one used by the code is determined from the value of switch `isumat` in input block `TFC`:

Flags :
- Set `isumat = 1` for binary Nb_3Sn superconductor
- Set `isumat = 2` for ternary Nb_3Sn superconductor
- Set `isumat = 3` for NbTi superconductor

Routines :
- `PROTECT` is called by `SUPERCON`
- `SUPERCON` is called by `TFSPCALL`
- `TFSPCALL` is called by `CALLER` and `OUTPUT`

3.2.4 PF Coil Options

3.2.4.1 PF Coil Position

The PF coil locations are controlled using a set of switches stored in array `ipfloc` (see Figure 2.1), and are calculated in routine `PFCOIL`. The coils are (usually) organised into groups containing two PF coils placed symmetrically above and below the midplane, and each group `j` has an element `ipfloc(j)` assigned to it.
In the following, all variables are defined in the variable descriptor file `var.des`. The values for `rpf1`, `rpf2`, `zref(j)` and `routr` in input block `PFC` should be adjusted by the user to locate the PF coils accurately.

Flags :
- Set `ngrp` in input block `PFC` to the number of symmetric pairs of PF coils.
- Set `ncls(j)` in input block `PFC` to the number of coils in each group `j` — this should be 2 in each case.
- Set `ipfloc(j)` in input block `PFC` as follows:

`ipfloc(j) = 1` : PF coils are placed above the OH coil:

$$R = \texttt{rohc + rpf1}$$
$$Z = \pm\texttt{(hmax*ohhghf + 0.3)}$$

Clearly, only one group of PF coils can be placed here.

`ipfloc(j) = 2` : PF coils are placed above the TF coils:

$$R = \texttt{rmajor + rpf2*triang*rminor}$$
$$Z = \begin{cases} \pm\texttt{(hmax - zref(j))} & \texttt{iohcie = 1 (TART)} \\ \pm\texttt{(hmax + tfcth + 0.86)} & \texttt{iohcie = 2} \end{cases}$$

Clearly, only one group of PF coils can be placed here if `iohcie = 2`.

`ipfloc(j) = 3` : PF coils are placed radially outside the TF coils:

$$R = \texttt{rtot + tfthko/2.0D0 + routr}$$
$$Z = \pm\texttt{(rminor*zref(j))}$$

Any number of groups of PF coils can be placed here.

3.2.4.2 PF Coil Resistance

The PF coils can be either resistive or superconducting. This is determined from the value of `ipfres` in input block `PFC`. If `ipfres = 0`, the PF and OH coils are assumed to be superconducting. If `ipfres = 1`, they are assumed to be resistive, with resistivity given by the value of variable `pfclres` in input block `PFC`.

3.2.4.3 PF Coil Current Scaling

The PF coil current scaling algorithm used by *PROCESS* depends on the value of switch `istokpf` in input block `PFC`. If `istokpf = 1`, then a simple scaling is used which is relevant to TART machines. If `istokpf = 2` a more complicated scaling is used, which is valid for conventional aspect ratio machines.

3.2.5 OH Coil Options

Switch `iohcl` in input block `BLD` controls whether an OH coil is present. A value of `1` denotes that this coil is present, and should be assigned a non-zero thickness `ohcth` in input block `BLD`. A value of `iohcl = 0` denotes that no OH coil is present, in which case the thickness `ohcth` should be set to zero.

3.2.5.1 OH Coil Swing Time

The length of time taken for the OH coil current to reverse (see Figure 2.4) is determined from the value of switch `tohsin` in input block `TIME`. If `tohsin = 0.0D0`, then the swing time `tohs` is given by `tohs` $= I_p/0.5$, where I_p is the plasma current in MA. If `tohsin` $\neq$ `0.0D0`, the swing time `tohs = tohsin`.

3.2.6 Other Engineering Models

3.2.6.1 Scrape-off width

Switch `iscrp` in input block `PHYDAT` determines whether the scrape-off widths should be calculated as 10% of the plasma minor radius (`iscrp = 0`), or set equal to the input values `scrapli` and `scraplo` (`iscrp = 1`).

3.2.6.2 First wall, blanket and shield

The various material fractions making up these components are all available to be changed in input block `FWBLSH`.

3.2.6.3 Divertor model

Two switches in *PROCESS* determine which divertor model to use. The first of these, `idivrt` in input block `PHYDAT`, controls the plasma configuration:

Flags :
- Set `idivrt = 0` for limiter configuration
- Set `idivrt = 1` for single null configuration (diverted side down)
- Set `idivrt = 2` for double null configuration

In fact, it is recommended that only `idivrt = 2` should be used, as the PF coil current scaling algorithms only allow for this model.

The second switch relevant to the divertor system is `istok` in input block `PHYDAT`. This controls whether a gaseous divertor concept [2] is assumed and a simple divertor heat load calculation is employed (`istok = 1`), or whether the Harrison-Kukushkin-Hotston model [8] developed for ITER is used (`istok = 2`). The first of these is relevant only for TART machines. The second, more complicated model, is appropriate for conventional aspect ratio machines, but care should be taken in inputting the divertor magnetics for this model, and projections far from the ITER CDA machine parameters are likely to be unreliable.

The divertor calculations are carried out in routines `DIVCALL` and `DIVERT`.

3.2.6.4 Heat transport and power conversion

Many of the power conversion efficiencies shown in Figure 2.5 can be adjusted by the user. The primary coolant is controlled by switch `ihts` in input block `HTPWR`. If `ihts = 0`, water is used; if `ihts = 1`, a liquid metal coolant is assumed.

3.2.6.5 Vacuum pump

Switch `ntype` in input block `VACCY` controls whether a turbopump (`ntype = 0`) or a cryopump (`ntype = 1`) is used in the vacuum system.

3.3 Cost Models

The cost accounting used by *PROCESS* combines methods [11] used in the TETRA code [1] and the Generomak [12] scheme. The costs are split into the standard accounting categories [13] generally used in the reporting of power plant costs. The best references for the algorithms used are [2], and source file `costs.f` in the code itself.

The majority of the costed items have a unit cost associated with them. These values scale with (for example) power output, volume, component mass etc., and many are available to be changed in input block `COSTINP`. All costs and their algorithms correspond to 1990 dollars.

3.3.1 Cost Options

3.3.1.1 Level of safety assurance

Many of the unit costs have four possible choices, relating to the level of safety assurance [14] flag `lsa` in input block `COSTINP`. A value `lsa = 1` corresponds to a plant with a full safety credit (i.e. is truly passively safe). Levels `2` and `3` lie between the two extremes, and level `4` corresponds to a present day fission reactor, with no safety credit. It is recommended that a value of `lsa = 3` or `lsa = 4` should be used.

3.3.1.2 Replaceable components

The first wall, blanket, divertor and current drive system have relatively short lifetimes because of their hostile environment, after which they must be replaced. Because of this frequent renewal they can be regarded as though they are "fuel" items, and can be costed accordingly. Switch `ifueltyp` in input block `COSTINP` is used to control whether this option is used in the code. If `ifueltyp = 1`, the costs of the first wall, blanket, divertor and a fraction `fcdfuel` of the cost of the current drive system are treated as fuel costs. If `ifueltyp = 0`, these are treated as capital costs. Variable `fcdfuel` is contained in input block `COSTINP`.

3.3.1.3 Cost of electricity calculations

Switch `ireactor` in input block `COSTINP` determines the type of cost of electricity calculation that is performed. If `ireactor = 0`, no cost of electricity calculation is performed. If `ireactor = 1`, then this calculation *is* performed, with the value quoted in units of m$/kWh. If `ireactor = 3` a simple capital cost calculation is performed only.

3.3.1.4 Net electric power calculation

Related to the cost of electricity is the net electric power calculation performed in routine POWER. It is possible that the net electric power can become negative due to a high recirculating power. Switch ipnet in input block COSTINP determines whether the net electric power is scaled to always remain positive (ipnet = 0), or whether it is allowed to become negative (ipnet = 1), in which case no cost of electricity calculation is performed.

3.4 TART Switches

As stated above, there are many switches that must be set if a TART machine is to be modelled by *PROCESS*. Switch itart is used to provide overall control of the various switches relating to conventional and tight aspect ratio machine options. Table 3.2 summarises the switch values relevant to each aspect ratio regime.

switch	conventional aspect ratio itart = 0	tight aspect ratio itart = 1
ibss	1, 2, 3	2, 3
icurr	1, 3, 4, 5, 6, 7	2
ishape	0	0, 1
istok	2	1
itfsup	1, 2	0
istokpf	2	1
iohcie	2	1

Table 3.2: *Summary of the switch values in PROCESS that relate to conventional aspect ratio and tight aspect ratio machines. The value of* itart *provides overall control of these options — the code will stop if an inconsistent set of values is attempted.*

3.5 Other Switches and Models

3.5.1 Output Control

Since the user may only be interested in a small proportion of the code's output, a set of switches exist in input block `OSECTS` that control whether a given section of the output file is produced. Table 3.3 indicates how these switches affect the output.

switch	relevant output section
`sect01`	power plant costs
`sect02`	detailed costings
`sect03`	plasma
`sect04`	current drive system
`sect05`	divertor
`sect06`	machine build
`sect07`	TF coils
`sect08`	PF coils
`sect09`	volt second consumption
`sect10`	support structure
`sect11`	PF coil inductances
`sect12`	shield / blanket
`sect13`	power conversion
`sect14`	heat transport
`sect15`	vacuum system
`sect16`	plant buildings
`sect17`	AC power
`sect18`	neutral beams
`sect19`	electron cyclotron heating
`sect20`	Lower Hybrid heating

Table 3.3: *Summary of the switches in PROCESS that control the format of the output file. If a switch has a value* `0`*, the relevant output section does not appear in the output file. If its value is* `1`*, the output section is included in the output file.*

3.5.2 Code Parameters Affecting Other Models

This chapter has summarised the methods by which several of the models in the code can be activated. There are many others present, however, and it is suggested that the user refers to the variable descriptor file, `var.des`. As stated earlier, this contains details of

all the parameters within the code that can be changed by the user, in order to customise the machine modelled by *PROCESS*.

Chapter 4

Execution of the Code

The intention of this chapter is to provide a comprehensive prescription for setting up and performing runs with the code. Firstly, the main concepts relating to the numerics of *PROCESS* are defined. Then the input file's structure and format is described. The user is then taken through the process of setting up the code to model a new machine, and finally an attempt is made to indicate and solve the problems that the user will face whilst trying to achieve a feasible solution.

4.1 Main Concepts

4.1.1 Variable Descriptor File

Great emphasis has already been placed on using the variable descriptor file because of its role as an invaluable resource for the user of *PROCESS*. It acts as a dictionary / reference manual for the code's variables, and contains the following information about each:

- name
- type — `real` = single precision real; `dble` = double precision real; `intg` = integer; `char` = character string
- dimensions (of arrays)

- default value(s) of those variables that are not initially derived from a combination of other values. The default values are mostly set in routine `INITIAL`
- description, including physical units if relevant
- for switches/flags, the meanings of all allowed values
- iteration variable number, if relevant
- corresponding constraint equation, if relevant

In addition, global code parameters are labelled `PAR`. These can only be changed by editing the relevant `INCLUDE` file, but this should not be carried out unless it is absolutely necessary.

All the variables that are shown with a default value are available to be changed by the user using the input file (Section 4.2), except for those which are labelled `FIX`. Variables not shown with a default value are calculated by the code from a combination of other parameters, and so it would be meaningless to initialise them. Obviously, these variables cannot be changed using the input file.

It is exceedingly important to keep the variable descriptor file up to date.

4.1.2 Input Parameters

Input parameters make up a large proportion of the variables listed in the variable descriptor file. They comprise all those variables that, once set in the initialisation routine or redefined in the input file, do not change throughout a *PROCESS* run. In fact, only those variables defined as iteration variables (Section 4.1.4) can change during the course of a run.

4.1.3 Constraint Equations

Any computer program naturally contains myriads of equations. The built-in equation solvers within *PROCESS* act on a special class, known as *constraint equations*, all of which are formulated in routine `CON1` in source file `eqns.f`. Table 4.1 summarises the constraint equations available in *PROCESS*. These can be split into two types — (1) consistency equations, that enforce consistency between the physics and engineering parameters, and

(2) limit equations, that enforce various parameters to lie within their allowed limits. The `neqns` constraint equations that the user chooses for a given run are activated by including the equation numbers in the first `neqns` elements of array `icc` in input block `INPT1`.

4.1.3.1 Consistency equations

Consistency equations are usually *equalities* that ensure that the machine produced by *PROCESS* is self-consistent. This means, therefore, that many of these constraint equations should *always* be used, namely equations 1, 2, 10 and 11 (see Table 4.1). Equation 7 should also be activated if neutral beam injection is used and equation 15 should be used if a TART machine is being modelled. The other consistency equations can be activated if required.

A typical consistency equation ensures that two functions g and h are equal:

$$\begin{aligned} g(x,y,z,\ldots) &= h(x,y,z,\ldots) \\ c_i &= 1 - \frac{g}{h} \end{aligned}$$

The equation solvers VMCON and HYBRD need the constraint equations c_i to be given in the form shown, since they adjust the iteration variables so as to obtain $c_i = 0$, thereby ensuring that $g = h$.

4.1.3.2 Limit equations

The limit equations are usually *inequalities* that ensure that various physics or engineering limits are not exceeded. Each of these equations has an associated *f-value*, which allow them to be *coded* as equalities. The f-values are used as follows.

In optimisation mode, all iteration variables have prescribed lower and upper bounds. In general, limit equations have the form

$$\textit{calculated quantity} = f \times \textit{maximum allowable value}$$

where f is the f-value. If f has a lower bound of zero and an upper bound of one, then the limit equation does indeed constrain the calculated quantity to lie between zero and its maximum allowable value, as required.

As with the consistency equations, the general form of the limit equations is

$$c_i = 1 - f.\frac{g}{h}$$

where g is the maximum allowed value of the quantity h.

Sometimes, the limit equation and f-value are used to ensure that quantity h is *larger* than its *minimum* value g. In this case, $0 \leq f \leq 1$ (as before), but the equation takes the form

$$c_i = 1 - f.\frac{h}{g}$$

By fixing the f-value (i.e. not including it in the `ixc` array), the limit equations can be used as equality constraints. For example, to set the net electric power to a certain value, the following should be carried out:

1. Activate constraint equation 16 by including it in the first `neqns` elements of array `icc` in input block `INPT1`
2. Set `fpnetel = 1.0D0` in input block `INEQDAT`
3. Ensure that `fpnetel` (iteration variable no. 25) *DOES NOT* appear in array `ixc` in input block `INPT1`
4. Set `pnetelin` in input block `INEQDAT` to the required net electric power

Limit equations are not restricted to optimisation mode. In non-optimisation mode, the iteration variables are not bounded, but the f-values can still be used to provide information about how calculated values compare with limiting values, without having to change the characteristics of the device being benchmarked to find a solution.

It is for this reason that all the constraint equations used in *PROCESS* are formulated as equalities, despite the fact that equation solver VMCON can solve for inequalities as well. The use of f-values precludes this need, and allows the non-optimising equation solver HYBRD to use the same constraint equations.

icc no.	description	type	corresponding ixc variables
1	plasma beta consistency	C	**5**
2	global power balance	C	**10**,1,2,3,4,6,11
3	ion power balance	C	**10**,1,2,3,4,6,11
4	electron power balance	C	**10**,1,2,3,4,6,11
5	density limit	L	**9**,1,2,3,4,5,6
6	epsilon-beta poloidal limit	L	**8**,1,2,3,4,6
7	beam ion density (NBI)	C	**7**
8	wall load limit	L	**14**,1,2,3,4,6
9	fusion power limit	L	**6**,1,2,3,4
10	field at coil / field on axis	C	**12**,1,2,3,13
11	radial build = major radius	C	**3**,1,13,16,29
12	volt second limit	L	**15**,1,2,3
13	edge q limit	L	**17**,1,2,3,18
14	beam energy (NBI)	C	**19**,1,2,3,6
15	centrepost conductor average temperature (TART)	C	**20**,21,22,23
16	net electric power limit	L	**25**,1,2,3
17	peak centrepost temperature limit (TART)	L	**26**,13,16,21,22,23
18	divertor heat load limit	L	**27**
19	MVA limit	L	**30**
20	port size limit	L	**33**,31,3,13
21	minor radius limit	L	**32**
22	divertor collisionality limit	L	**34**
23	TF coil current density limit	L	**28**,12
24	Troyon beta limit	L	**36**,1,2,3,4,6,18
25	toroidal field limit	L	**35**,3,13,29
26	OH coil current density at End of Flat-top	L	**38**,12
27	OH coil current density at Beginning of Pulse	L	**39**,12
28	energy multiplication Q limit	L	**45**,47,40
29	inboard radial build = specified value	C	**3**,1
30	injection power limit	L	**46**,47,11
31	TF coil case stress limit (SCTF)	L	**48**,56,57,58,59,60
32	TF coil conduit stress limit (SCTF)	L	**49**,56,57,58,59,60
33	$I_{\text{operational}}/I_{\text{critical}}$ limit (SCTF)	L	**50**,56,57,58,59,60
34	dump voltage limit (SCTF)	L	**51**,56,57,58,59,60
35	$J_{\text{winding pack}}/J_{\text{protection}}$ limit (SCTF)	L	**53**,56,57,58,59,60
36	TF coil temperature margin limit (SCTF)	L	**54**,56,57,58,59,60
37	current drive gamma limit	L	**40**,47

Table 4.1: *Summary of the constraint equations used in PROCESS. Consistency equations are marked C, limit equations are marked L. Some (non-exhaustive) iteration variable numbers (see Table 4.2) that directly affect the associated constraint equations are given, the one listed first being the most relevant.*

4.1.4 Iteration Variables

It is necessary to calculate numerical derivatives during the solution of the constraint equations. The iteration variables are the parameters that the equation solvers use for this purpose — all the other code variables (input parameters — see above) remain fixed at their initial value. Successive calls are made to the physics and engineering routines, with slightly different values for the iteration variables on each call, and the equation solver determines the effect on the output due to these small changes to the input (see Figures 2.6 and 2.7). The `nvar` iteration variables that the user chooses for a given run are activated by including the variable numbers in the first `nvar` elements of array `ixc` in input block `INPT1`. Table 4.2 summarises the iteration variables available in *PROCESS*.

Clearly, the equation solvers need at least as many variables to iterate as there are equations to solve, i.e. `nvar` $\geq$ `neqns`. If the run is a non-optimising case, then `neqns` variables are iterated — the values of the remaining `(nvar-neqns)` variables are left alone. If the run is an optimising case, then all the active iteration variables are adjusted so as to find the minimum (or maximum) value of a parameter (the *figure of merit*) in the `nvar`-dimensional space of the problem.

All the iteration variables are constrained to lie between lower and upper bounds, stored in arrays `boundl` and `boundu`, respectively, in input block `INPT1`. For instance, the plasma electron density is, by default, confined to lie between the values 10^{19} m^{-3} and 10^{21} m^{-3}. Of course, it can also be constrained to lie below the *calculated* density limit, if constraint equation 5 is activated and the f-value `fdene` (iteration variable no. 9) is bounded by the values 0 and 1.

4.1.5 Figures of Merit

In optimisation mode, *PROCESS* finds the self-consistent set of iteration variable values that maximises or minimises a certain function of them, known as the *figure of merit*. Several possible figures of merit are available, all of which are formulated in routine `FUNFOM` in source file `optimiz.f`. Switch `minmax` in input block `INPT1` is used to control which figure of merit is to be used, as summarised in Table 4.3. If the figure of merit is to be minimised, `minmax` should be positive, and if a maximised figure of merit is desired, `minmax` should be negative.

ixc no.	variable name	description	icc eqn	lower bound	upper bound
1	aspect	plasma aspect ratio		1.100D0	10.00D0
2	bt	toroidal field on axis		0.010D0	100.0D0
3	rmajor	plasma major radius		0.100D0	10.00D0
4	te	electron temperature		5.000D0	500.0D0
5	beta	plasma beta		0.001D0	1.000D0
6	dene	electron density		1.00D19	1.00D21
7	rnbeam	hot beam density / electron density		1.00D-6	1.00D20
8	fbeta	f-value for $\epsilon.\beta_p$ limit eqn	6	0.001D0	1.000D0
9	fdene	f-value for density limit eqn	5	0.001D0	1.000D0
10	hfact	confinement time H-factor		0.100D0	3.000D0
11	pheat	heating power not used for current drive		1.000D6	1.000D9
12	oacdcp	overall current density in TF coil inner leg		1.000D5	1.500D8
13	tfcth	TF coil inner leg thickness		0.100D0	5.000D0
14	fwalld	f-value for wall load limit eqn	8	0.001D0	1.000D0
15	fvs	f-value for volt second limit eqn	12	0.000D0	1.000D0
16	ohcth	OH coil thickness		0.001D0	1.000D2
17	fq	f-value for edge q limit eqn	13	0.001D0	1.000D0
18	q	edge safety factor		2.000D0	100.0D0
19	enbeam	neutral beam energy		1.000D0	1.000D6
20	tcpav	average centrepost temperature		40.00D0	1.000D3
21	vcool	maximum coolant flow speed in centrepost		1.000D0	1.000D2
22	rcool	average radius of coolant channel in centrepost		0.001D0	0.010D0
23	fcoolcp	coolant fraction of centrepost		0.100D0	0.500D0
24	cdtfleg	TF coil leg overall current density		1.000D4	1.000D8
25	fpnetel	f-value for net electric power limit eqn	16	1.000D0	1.000D0
26	fptemp	f-value for peak centrepost temperature limit eqn	17	0.001D0	1.000D0
27	fhldiv	f-value for divertor heat load limit eqn	18	0.001D0	1.000D0
28	fjtfc	f-value for TF coil current density limit eqn	23	0.100D0	1.000D0
29	bore	machine bore		0.100D0	10.00D0
30	fmva	f-value for MVA limit eqn	19	0.010D0	1.000D0
31	gapomin	minimum gap between outer shield and TF coil		0.001D0	1.000D1
32	frminor	f-value for minor radius limit eqn	21	0.001D0	1.000D0
33	fportsz	f-value for port size limit eqn	20	0.010D0	1.000D0
34	fdivcol	f-value for divertor collisionality limit eqn	22	0.001D0	1.000D0
35	fpeakb	f-value for peak toroidal field limit eqn	25	0.001D0	1.000D0
36	fbetatry	f-value for Troyon beta limit eqn	24	0.001D0	1.000D0
37	coheof	OH coil current density at end of flat-top		1.000D5	1.000D8
38	fjohc	f-value for OH coil current at EOF limit eqn	26	0.010D0	1.000D0
39	fjohc0	f-value for OH coil current at BOP limit eqn	27	0.001D0	1.000D0
40	fgamcd	f-value for current drive gamma limit eqn	37	0.001D0	1.000D0
41	fcohbop	OH coil current density ratio BOP/EOF		0.001D0	1.000D0
42	fhldiv (REDUNDANT)	f-value for divertor heat load limit eqn		0.001D0	1.000D0
43	cfe0	iron impurity fraction		1.00D-6	3.00D-3
44	fvsbrnni	fraction of volt seconds from non-inductive means		0.001D0	1.000D0
45	fqval	f-value for energy multiplication limit eqn	28	0.010D0	0.330D0
46	fpinj	f-value for injection power limit eqn	30	0.001D0	1.000D0
47	feffcd	current drive efficiency multiplier		0.001D0	1.000D0
48	fstrcase	f-value for TF coil case stress limit eqn	31	0.001D0	1.000D0
49	fstrcond	f-value for TF coil conduit stress limit eqn	32	0.001D0	1.000D0
50	fiooic	f-value for TF coil operational current limit eqn	33	0.001D0	0.500D0
51	fvdump	f-value for TF coil dump voltage limit eqn	34	0.001D0	1.000D0
52	vdalw	allowable TF coil dump voltage		0.001D0	1.000D6
53	fjprot	f-value for TF coil current protection limit eqn	35	0.001D0	1.000D0
54	ftmargtf	f-value for TF coil temperature margin limit eqn	36	0.001D0	1.000D0
55	tmargmin	minimum allowable TF coil temperature margin		0.001D0	100.0D0
56	tdmptf	dump time for TF coil		10.00D0	1.000D6
57	thkcas	TF coil external case thickness		0.050D0	1.000D0
58	thwcndut	TF coil conduit case thickness		0.001D0	1.000D0
59	fcutfsu	copper fraction of cable conductor		0.001D0	1.000D0
60	cpttf	current per turn in the TF coils		0.001D0	4.000D4

Table 4.2: *Summary of the iteration variables used in PROCESS. The f-values correspond to the given constraint equations (see Table 4.1).*

minmax	description
±1	plasma major radius
±2	ratio of fusion power to input power
±3	neutron wall load
±4	total TF coil + PF coil power
±5	ratio of fusion power to injection power
±6	cost of electricity
±7	capital cost if `ireactor = 3` direct cost if `ireactor = 0` constructed cost otherwise
±8	aspect ratio
±9	divertor heat load
±10	toroidal field on axis
±11	injection power

Table 4.3: *Summary of the figures of merit used in PROCESS. If the figure of merit is to be minimised,* `minmax` *should be positive, and if a maximised figure of merit is desired,* `minmax` *should be negative.*

4.1.6 Scanning Variables

One of a number of variables can be scanned during the course of a *PROCESS* run. This option provides a method of determining the sensitivity of the results to different input assumptions. The user specifies which variable is to be scanned (see Table 4.4) and its required value at each point in the scan. The scanned variable is defined by the value of `nsweep` in input block `SWEP`, and this variable's values during the scan are set in array `sweep`, also in input block `SWEP`.

Runs involving scans of this kind can only be performed in optimisation mode. The results from the previous scan point are used as the input to the next scan point. Routine `SCAN` in source file `aamain.f` stores many of the output quantities in a separate file for use with a plotting program.

Scanning of derived quantities requires use of the appropriate constraint equations. For instance, if the net electric power is scanned, constraint equation 16 should be employed.

nsweep	scan variable	description
1	aspect	aspect ratio
2	hldivlim	maximum divertor heat load
3	pnetelin	required net electric power
4	hfact	confinement time H-factor
5	oacdcp	overall current density in TF coil inner leg
6	walalw	allowable wall load
7	beamfus0	beam-background fusion multiplier
8	fqval	f-value for energy multiplication equation
9	te	electron temperature
10	boundu(15)	upper bound on f-value fvs
11	dnbeta	Troyon g coefficient
12	bscfmax	bootstrap current fraction (use negative values)

Table 4.4: *Summary of the scanning variables available in PROCESS.*

4.2 The Input File

The input file is used to change the values of the physics, engineering and other code parameters from their default values, and to set up the numerics (constraint equations, iteration variables etc.) required to define the problem to be solved.

4.2.1 File Structure

The input file is divided into *input blocks* pertaining to the modular structure of *PROCESS* itself. Each INCLUDE file has a corresponding input block, as shown in Table 2.1.

This file structure is derived from that used in an old version of *PROCESS*, which used the NAMELIST method to read in data. This has now been superseded by FORTRAN 77 standard routines, which also allow a great deal of error-trapping to be carried out at the input stage. All input data are now screened for non-sensible values. This is a necessary addition to the code, since UNIX workstation systems using RISC processors are notorious at not terminating programs when an arithmetically impossible or undefined operation ("NaN" error) is encountered.

4.2.2 Format Rules

The following rules must be obeyed when writing an input file:

1. Each input block must start with the block name preceded by a dollar sign (`$`), and end with a `$END` statement.
2. Each variable must be on a separate line, within the correct input block.
3. There should be no leading spaces before the variable name, input block name or `$END` statement.
4. Variable names, input block names and `$END` statements can be upper case, lower case, or a mixture of both.
5. Spaces may not appear within a variable name or data value.
6. Other spaces within a line, and trailing spaces, are ignored.
7. Commas are not necessary between variables.
8. Data can extend over more than one line.
9. One-dimensional arrays can be explicitly subscripted, or unscripted, in which case the following element order is assumed: `A(1), A(2), A(3),...`
10. At present, multiple dimension arrays can only be handled without reference to explicit subscripts, in which case the following element order is assumed: `B(1,1), B(2,1), B(3,1),...` The use of the input file to specify multiple dimension array elements is prone to error.
11. Unscripted array elements must be separated by commas.
12. Blank lines are allowed anywhere in the input file.
13. Lines starting with a * are assumed to be comments.
14. Comment lines starting with five or more asterisks (i.e. *****) are reproduced verbatim in the output file. These should be used copiously to give a great deal of information about the run being performed, and should be updated before every single run of the code, as it is very easy to lose track of what is being attempted.

It is good practice to include all the input blocks in the input file, even if none of the variables in some of them need changing. In this case, the $END statement should immediately follow (on the next line) the input block name statement, or a variable should be added in the usual way, but given its default value as set in routine INITIAL.

The following is a valid input block in the input file (the vertical lines denote the "edge" of the input file) :

```
* This line is a comment that will not appear in the output
***** This line is a comment that will appear in the output
$inpt1
boundl(1) = 2.5,
BOUNDU(10) = 3.,
BOUNDU(45) = 1,
* Another comment...  Note that real values can be entered as if
* they were integers, but NOT vice versa.
epsfcn = 10.e-4,
Ftol = 1.D-4,
ICC = 2, 10, 11, 24, 31
ixc = 10, 12, 3, 36, 48,
       1, 2, 6, 13, 16,
IOPTIMZ = 1,
maxcal = 200
NEQNS = 5,
NVAR = 10,
$END
```

The following are *invalid* entries in the input file:

```
 $inpt1
boundl(1,1) = 2.5,
BOUNDU(N) = 3.,
A line of 'random' characters like this will clearly wreak havoc
 * This is a comment with the asterisk not in the first column
eps fcn = 10.e-4, ftol = 1.D-4
epsvmc = 1.0 e-4
 maxcal = 200
ICC = 2 10 11 24 31
IOPTIMZ = 1.0,
 $END
```

4.3 Running the Code

This section will attempt to guide the user through the actual running of the code in its various modes. In most cases only minor changes to the input file are necessary to change the code's mode of operation — usually the physics and engineering variables, etc. remain unchanged, with the major differences occurring in the numerical input only.

4.3.1 Non-optimisation Mode

Non-optimisation mode is used to perform benchmark comparisons, whereby the machine size, output power etc. are known and one only wishes to find the calculated stresses, beta values and fusion powers, for example. When starting to model a new machine, *PROCESS* should always be run first in non-optimisation mode, before any attempt is made to optimise the machine's parameters.

The first thing to do is to add to the input file all the known details about the machine to be modelled. This may include some or all of the following:

- machine build (input block `BLD`)
- aspect ratio (input block `PHYDAT`)
- PF coil locations (input block `PFC`)
- type of current drive to be used (input block `CDDAT`)
- net electric power (input block `INEQDAT`)
- various physics parameters (input block `PHYDAT`), e.g.
  - toroidal field on axis
  - electron density
  - electron temperature
  - elongation
  - triangularity
  - Troyon g coefficient
  - edge safety factor

In addition, some of the switch values summarised in Chapter 3 may have to be altered from their default values.

Next, the relevant numerics information must be entered. Switch `ioptimz` in input block `INPT1` must be set to `-1` for non-optimisation mode. Then the user must decide which constraint equations and iteration variables to activate — this choice is dictated partly by the information required by the user, and partly by the machine being modelled itself.

As stated earlier, all the relevant consistency equations must be activated, together with the corresponding iteration variables. A number of limit equations can also be activated, to investigate how the calculated values compare with the physics or engineering limits. The following is part of an example non-optimisation input file:

```
$INPT1
IOPTIMZ = -1
NEQNS = 8
NVAR = 8
ICC =  1,  2, 10, 11,  7, 16,  5, 24,
IXC =  5, 10, 12, 29,  7,      9, 36, 4,
$END

$INEQDAT
FPNETEL = 1.0
PNETELIN = 1200.0
$END
⋮
```

Consistency equations 1, 2, 10, 11 and 7 are activated, together with limit equations 16, 5 and 24. This example assumes that neutral beam current drive is present (equation 7 with variable 7), and that the net electric power is to be fixed at 1200 MW. Note the practice of vertically aligning corresponding equations and variables — constraint equation 16 has no corresponding iteration variable (which would normally be no. 25, `fpnetel`), as we want the net electric power to be fixed at the value given by `pnetelin`. Since in non-optimisation mode, the number of variables must be equal to the number of equations, we have scope to add a "free" iteration variable, in this case no. 4 — electron temperature, to help raise the fusion power sufficiently to obtain the required net electric power. Finally, note the use of the density and Troyon beta limit equations; the values of the corresponding f-values will indicate if the limits are exceeded and by how much.

On running *PROCESS* and (hopefully) achieving a feasible result, examination of the output may well show up discrepancies between some of the parameter values produced

and their known values (if available). Remember that, of all the variables shown in the variable descriptor file with a default value, only those declared as *active iteration variables* can change from their initial values, whether they are set in the input file or in the initialisation routine `INITIAL`. However some of the calculated parameters may be wrong, the most common of which are as follows:

- Plasma current. This can be adjusted using the edge safety factor `q` in input block `PHYDAT`: $I_p \propto 1/q$
- Fusion power. This scales roughly with the density profile factor `alphan` in input block `PHYDAT`.
- Build parameters. It may be necessary to change non-critical thicknesses to achieve the correct machine build.

It may still be difficult, if not impossible, to reconcile the fusion power and the net electric power with the required values. This may well be due to the power conversion efficiency values being used — refer to Figure 2.5.

With luck, a few iterations of this process will produce an adequate benchmark case. A typical input file for use with *PROCESS* in non-optimisation mode is contained in Appendix A.

4.3.2 Optimisation Mode

Running *PROCESS* in optimisation mode requires few changes to be made to the input file from the non-optimisation case, except in input blocks `INPT1`, `INEQDAT` and `SWEP` — the blocks associated with the numerics of the problem. The main differences between optimisation mode and non-optimisation mode are:

1. Optimisation mode applies lower and upper bounds to all active iteration variables.
2. There is no upper limit to the number of active iteration variables in optimisation mode.
3. A figure of merit must be specified in optimisation mode.
4. Scans can be performed in optimisation mode.

Switch `ioptimz` in input block `INPT1` must be set to `1` for optimisation mode. As before, the user must decide which constraint equations and iteration variables to activate. Again, the choice depends largely on the information required by the user and the extent of the freedom that the code may have with the machine's parameters.

The following is part of an example optimisation input file:

```
$INPT1
IOPTIMZ = 1
NEQNS = 16
NVAR = 19
ICC =  1,  2, 10, 11,  7, 16,  5, 24, 14,  8, 31, 32, 33, 34, 35, 36,
IXC =  5, 10, 12, 29,  7,      9, 36, 19, 14, 48, 49, 50, 51, 53, 54,
4, 6, 1, 18,
BOUNDL(1) = 2.5
BOUNDU(10) = 2.0
MINMAX = 6
$END

$INEQDAT
FPNETEL = 1.0
PNETELIN = 1200.0
WALALW = 4.4
$END

$SWEP
ISWEEP = 3
NSWEEP = 11
SWEEP = 3.5, 3.7, 3.9
$END
⋮
```

The figure of merit in this example is the (minimum) cost of electricity (`minmax = 6`). Note that additional limit equations are now active, along with a second consistency equation related to the neutral beam current drive — the number of decay lengths to the plasma centre is constrained to be equal to the input value (`tbeamin` in input block `CDDAT`, which is not shown here). Furthermore, there are now more iteration variables than constraint equations, to aid the minimisation process. Finally, note that a three-point scan in the Troyon g coefficient `dnbeta` — scanning variable 11, is to be performed.

A useful practice in optimisation mode is to perform "stationary" scans, whereby the same value is given to the scanning variable at successive iterations. This provides a check as to how well converged the solution has become. If scans of a given variable are to be made over a large range of values, it is often a good idea to start the scan in the middle of the desired range, and to split the scan in two — one going downwards from the initial value, and the other upwards. This ensures that the whole range of the scan produces well-converged machines (assuming a "good" initial point), without sharp changes in gradient in the parameter values.

It should be remembered that the value of the scan variable is set in the array `sweep`, and this overrules any value set for the variable elsewhere in the input file. For instance, in the example above, the values of `dnbeta` set in the `sweep` array would overrule any value for `dnbeta` set in the `PHYDAT` input block.

The output from an optimisation run contains an indication as to which iteration variables lie at their limiting values. On the whole there is a greater chance of unfeasible solutions being found whilst in optimisation mode, and Section 4.4 will hopefully be of some use in this situation. A typical input file for use with *PROCESS* in optimisation mode is contained in Appendix B.

4.4 Problem Solving

Experience has shown that the first few attempts at running *PROCESS* with a new input file tends to produce unfeasible results — that is, the code will not find a consistent set of machine parameters. The highly non-linear nature of the numerics of *PROCESS* is the reason for this difficulty, and it can be a painstaking task to overcome.

4.4.1 General Problems

A code of the size and complexity of *PROCESS* contains myriads of equations and variables. Virtually everything depends indirectly on everything else because of the nature of the code structure, so perhaps it is not surprising that it is often difficult to achieve a successful outcome.

Naturally, problems will occur if some of the parameters become unphysical. For example, if the aspect ratio becomes less than or equal to one, then we must expect problems to appear. For this reason, the bounds on the iteration variables and the allowed ranges of

all the input variables have been selected with great care.

The code contains a large (though probably not exhaustive) number of error traps to try and prevent problems from propagating. These include tests for unphysical values, and checks to prevent divisions by zero, and non-sensible arguments for logarithms and square roots, etc. However, occasionally arithmetic ("NaN") errors still occur, although their incidence is low. They now usually only occur due to unfeasibility problems (see later).

The error messages produced by the code attempt to provide diagnostic information, telling the user where the problem occurs, and also suggest a possible solution. These messages are out of necessity brief, and so cannot promise to lead to a more successful outcome.

4.4.2 Optimisation Problems

On reflection it is perhaps surprising that *PROCESS* ever does manage to find the global minimum figure of merit value, since if there are `nvar` iteration variables active the search is over `nvar`-dimensional parameter space, in which there may be many shallow minima of approximately equal depth. Remember that `nvar` is usually of the order of twenty.

The machine found by *PROCESS* may not, therefore, be the absolutely optimal device. It is quite easy to have two or more solutions, with results only a few per cent different, but a long way apart in parameter space. The technique of "stationary" scans described in Section 4.3.2 above can often help in this situation, which is why this method is recommended at all times.

Scans should be started in the middle of a range of values, to try to keep the scan within the same family of machines. The optimum machine found may otherwise suddenly jump to a new region of parameter space, causing the output variables to seem to vary unpredictably with the scanning variable.

It should be noted that in general the machine produced by *PROCESS* will always sit against one or more operation limits. If, during a scan, the limit being leant upon changes (i.e. if the machine jumps from leaning on the beta limit to leaning on the density limit) the output parameters may well become discontinuous in gradient, and trends may suddenly change direction.

4.4.3 Unfeasible Results

In the numerics section of the output file, the code indicates whether the run produced a feasible or unfeasible result. The former implies a successful outcome. An unfeasible result, however, occurs if *PROCESS* cannot find a set of values for the iteration variables which satisfies all the given constraints. In this case, the values of the constraint residues shown in the output give some indication of which constraint equations are not being satisfied — those with the highest residues should be examined further. In optimisation mode, the code also indicates which iteration variables lie at the edge of their allowed range.

Unfeasible runs are caused either by ill-defining the problem to be solved, or by starting the problem in an unfavourable region of parameter space. The latter can be checked simply by changing the initial values of the *active* iteration variables in the input file, but the former requires some extra work. This situation arises if there are insufficient iteration variables for the given constraint equations. It is important to choose the right number of *useful* iteration variables for the problem to be solved — it is possible to activate too many iteration variables as well as too few, some of which may be redundant.

Unfeasible cases often produce unrealistic machines, so one should not believe the output values from these runs. Unfortunately, the stationary scan method sometimes, though not always, fails to help these cases, since it will tend to keep starting the run at the same point. Ill-defined problems sometimes produce arithmetic errors, for obscure reasons.

Though a great deal of work has been performed on the code to improve its standard, there can be no guarantee that *PROCESS* is entirely bug-free, simply because of its large size. Rarely, then, it may be that an unfeasible result indicates that the code has encountered a programming error, although its precise location will be almost impossible to find by simply examining the output file.

It may be the case that the act of satisfying all the required constraints is impossible. No machine can exist if the allowed operating regime is too restrictive, or if two constraint equations require conflicting parameter spaces. In this case some relaxation of the requirements is needed for the code to produce a successful machine design.

4.4.4 Hints

The above sections should indicate that it is the complex interplay between the constraint equations and the iteration variables that determines whether the code will be successful at producing a useful result. It can be a somewhat laborious process to arrive at a working case, and (unfortunately, perhaps) experience is often of great value in this situation.

It should be remembered that sufficient iteration variables should be used to solve each constraint equation. For instance, a particular limit equation may be $A \leq B$, i.e. $A = fB$, where the f-value f must lie between zero and one for the relation to be satisfied. However, if none of the iteration variables have any effect on the values of A and B, and A happens to be *greater* than B, then *PROCESS* will clearly not be able to solve the constraint.

The lower and upper bounds of the iteration variables are all available to be changed in the input file. Constraints can be relaxed in a controlled manner by moving these bounds, although in some cases care should be taken to ensure that unphysical values cannot occur. The code indicates which iteration variables lie at the edge of their range.

It is suggested that constraint equations should be added one at a time, with sufficient new iteration variables activated at each step. If the situation becomes unfeasible it can be helpful to reset the initial iteration variable values to those shown in the output from a previous feasible case, and rerun the code.

Finally, it should be borne in mind that the machine that is envisaged may not be a valid solution to the constraints being imposed, no matter how many degrees of freedom (i.e. iteration variables) are available. In this case, and many others, the user has to relax the constraints slowly until a feasible result is found.

Chapter 5

Inclusion of Additional Variables and Equations

It is often useful to add extra features to the code in order to model new situations. This chapter provides instructions on how to add various numerics related items to *PROCESS*.

5.1 Input Parameters

Input parameters (see Section 4.1.2) are added to the code in the following way:

1. Choose the most relevant `INCLUDE` file, and, keeping everything in alphabetical order, add the parameter to
   (a) the correct type declaration block, and
   (b) the corresponding `COMMON` block.
2. Ensure that all the routines that use the new variable reference the relevant `INCLUDE` file.
3. Add the parameter to the relevant section in routine `INITIAL` in source file `initial.f`, giving it a "sensible" default value. Keep to alphabetical order.
4. Add the parameter to the relevant routine in source file `input.f`, including the comments at the start of the routine. The comments in routine `READNL` provide

full instructions on how to do this. Note that real (i.e. double precision) and integer variables are treated differently, as are scalar quantities and arrays. Keep to alphabetical order. Also, add the parameter to the relevant comment in routine `INPUT`.

5. Add the details of the parameter to the relevant section of the variable descriptor file `var.des`, keeping to alphabetical order.

5.2 Iteration Variables

New iteration variables (see Section 4.1.4) are added in the same way as input parameters, with the following additions:

1. Increment the parameter `ipnvars` in `INCLUDE` file `param.h` to accommodate the new iteration variable.
2. Add the variable to routines `LOADXC` and `CONVXC` in source file `xc.f`, mimicking the way that the existing iteration variables are coded. Remember to ensure that these routines reference the relevant `INCLUDE` file.
3. Assign sensible values for the variable's bounds to the relevant elements in arrays `boundl` and `boundu` in routine `INITIAL` in source file `initial.f`.
4. Assign the relevant element of character array `lablxc` to the name of the variable, in routine `INITIAL` in source file `initial.f`.
5. Document the changes to `ipnvars` and `ixc` in the variable descriptor file `var.des`.

If an existing input parameter is now required to be an iteration variable, then simply carry out the tasks mentioned here.

5.3 Other Global Variables

This type of variable embraces all those present in the `INCLUDE` files which do not need to be given initial values or to be input, as they are calculated within the code. These should be added to the code in the following way:

1. Choose the most relevant `INCLUDE` file, and, keeping everything in alphabetical order, add the parameter to
   (a) the correct type declaration block, and
   (b) the corresponding `COMMON` block.
2. Ensure that all the routines that use the new variable reference the relevant `INCLUDE` file.
3. Add the parameter to the relevant section in routine `INITIAL` in source file `initial.f`, giving it a default value of zero. This is done to ensure that the variable is defined immediately, preventing possible problems later. Keep to alphabetical order, as always.
4. Add the details of the parameter to the relevant section of the variable descriptor file `var.des`.

5.4 Constraint Equations

Constraint equations (see Section 4.1.3) are added to *PROCESS* in the following way:

1. Increment the parameter `ipeqns` in `INCLUDE` file `param.h` to accommodate the new constraint.
2. Add the constraint equation to routine `CON1` in source file `eqns.f`, ensuring that all the variables used in the formula are contained in the `INCLUDE` files present at the start of this routine. Use a similar formulation to that used for the existing constraint equations, remembering that the code will try to force `cc(i)` to be zero.
3. Assign a description of the new constraint to the relevant element of array `lablcc` in routine `INITIAL` in source file `initial.f`, using 34 characters or less.
4. Document the changes to `ipeqns` and `icc` in the variable descriptor file `var.des`.

Remember that if a limit equation is being added, a new f-value iteration variable may also need to be added to the code.

5.5 Figures of Merit

New figures of merit (see Section 4.1.5) are added to *PROCESS* in the following way:

1. Increment the parameter `ipnfoms` in `INCLUDE` file `param.h` to accommodate the new figure of merit.
2. Add the new figure of merit equation to routine `FUNFOM` in source file `optimiz.f`, following the method used in the existing examples. The value of `fc` should be of order unity, so select a reasonable scaling factor if necessary.
3. Ensure that all the variables used in the formula are contained in the `INCLUDE` files present at the start of this routine.
4. Add a short description of the new figure of merit to the `minmax` entry in routine `RDNL01` in source file `input.f`.
5. Assign a description of the new figure of merit to the relevant element of array `lablmm` in routine `INITIAL` in source file `initial.f`, using 22 characters or less.
6. Document the changes to `ipnfoms` and `minmax` in the variable descriptor file `var.des`.

5.6 Scanning Variables

Scanning variables (see Section 4.1.6) are added to *PROCESS* in the following way:

1. Increment the parameter `ipnscnv` in `INCLUDE` file `param.h` to accommodate the new scanning variable.
2. Add a new assignment to the relevant part of routine `SCAN` in source file `aamain.f`, following the examples already present, including the inclusion of a short description of the new scanning variable in variable `xlabel`.

3. Ensure that the scanning variable used in the assignment is contained in one of the `INCLUDE` files present at the start of this routine.

4. Add a short description of the new scanning variable to the **nsweep** entry in routine `RDNL15` in source file `input.f`.

5. Document the changes to **ipnscnv** and **nsweep** in the variable descriptor file `var.des`.

Chapter 6

Acknowledgements & Bibliography

The author would like to thank the following people for many useful and revealing discussions during his work on *PROCESS*:

— John D. Galambos, Paul C. Shipe and Y-K. Martin Peng from Oak Ridge National Laboratory,

— Roger Hancox, Neill Taylor and John Hicks from Theoretical and Strategic Studies Department, AEA Fusion,

— Tim Hender from Microwave and Interpretation Department, AEA Fusion, and all the co-authors of reference [7].

This manual was produced as part of Theoretical and Strategic Studies, AEA Fusion project number CIRE 5523 on the *PROCESS* Reactor Systems Code.

This work was funded by the UK Department of Trade and Industry, Euratom, and by internal research funds of AEA.

Bibliography

[1] R. L. Reid et al., *"ETR/ITER Systems Code"*, Oak Ridge Report ORNL/FEDC-87/7 (1988)

[2] J. D. Galambos, *"STAR Code : Spherical Tokamak Analysis and Reactor Code"*, Unpublished internal Oak Ridge document. A copy exists in the *PROCESS* Project Work File [15].

[3] Y-K. M. Peng and J. B. Hicks, *"Engineering Feasibility of Tight Aspect Ratio Tokamak (Spherical Torus) Reactors"*, AEA Fusion Report AEA FUS 64 (1990)

[4] J. J. More, B. S. Garbow and E. Hillstrom, *"User Guide for MINPAC-1"*, Argonne National Laboratory Report ANL-80-74 (1980)

[5] M. J. D. Powell, *"A Hybrid Method for Non-linear Algebraic Equations"*, Numerical Methods for Non-linear Algebraic Equations, ed. P. Rabinowitz, Prentice-Hall

[6] R. L. Crane, K. E. Hillstrom and M. Minkoff, *"Solution of the General Nonlinear Programming Problem with Subroutine VMCON"*, Argonne National Laboratory Report ANL-80-64 (1980)

[7] T. C. Hender, M. K. Bevir, M. Cox, R. J. Hastie, P. J. Knight, C. N. Lashmore-Davies, B. Lloyd, G. P. Maddison, A. W. Morris, M. R. O'Brien, M. F. Turner and H. R. Wilson, *"Physics Assessment for the European Reactor Study"*, AEA Fusion Report AEA FUS 172 (1992)

[8] N. A. Uckan and ITER Physics Group, *"ITER Physics Design Guidelines: 1989"*, ITER Documentation Series, No. 10, IAEA/ITER/DS/10 (1990)

[9] J. P. Christiansen et al., *"Global Energy Confinement H-Mode Database for ITER"*, Nuclear Fusion **32** (1992) 291-338, Eqn. 6

[10] W. M. Nevins et al., *"Summary Report: ITER Specialists' Meeting on Heating and Current Drive"*, ITER-TN-PH-8-4, 13–17 June 1988, Garching, FRG

[11] R. L. Reid and Y-K. M. Peng, *"Potential Minimum Cost of Electricity of Superconducting Coil Tokamak Power Reactors"*, Proceedings of 13th IEEE Symposium on Fusion Engineering, Knoxville, Tennessee, October 1989, p. 258

[12] J. Sheffield et al., *"Cost Assessment of a Generic Magnetic Fusion Reactor"*, Fusion Technology **9** (1986) 199

[13] S. Thompson, *"Systems Code Cost Accounting"*, memo FEDC-M-88-SE,-004 (1988)

[14] J. P. Holdren et al., *"Report of the Senior Committee on Environmental Safety and Economic Aspects of Magnetic Fusion Energy"*, LLNL Report UCRL-53766 (1989)

[15] N. P. Taylor (holder) and P. J. Knight, *"PROCESS Reactor Systems Code"*, AEA Fusion Project Work File, F/RS/CIRE5523/PWF (1992)

Appendix A

Non-optimisation Input File

The following is a typical input file used to run *PROCESS* in non-optimisation mode. Comments have been added to the right of each line.

```
$inpt1                                  | Start of input block INPT1 (numerics)
NEQNS = 14,                             | Number of active constraint equations
NVAR = 14,                              | Number of active iteration variables
ICC =   1,  2, 10, 11, 7, 16, 24, 5, 31, 32, 33, 34, 35, 36, | Constraint eqns
ixc =   5, 10, 12, 3,  7,  6, 36, 9, 48, 49, 50, 51, 53, 54, | Iteration variables
IOPTIMZ = -1,                           | Turn off optimisation
$end                                    | End of input block INPT1
                                        |
$INEQDAT                                | Start of input block INEQDAT (f-values etc)
FBETATRY = 1.0                          | N.B. active iteration variable 36
$end                                    | End of input block INEQDAT
                                        |
$PHYDAT                                 | Start of input block PHYDAT (physics)
ASPECT = 3.5,                           | Machine aspect ratio
BETA = 0.042,                           | N.B. active iteration variable 5
BT = 6.,                                | Toroidal field on axis
DENE = 1.5e20,                          | N.B. active iteration variable 6
FVSBRNNI = 1.0,                         | Non-inductive volt seconds fraction
DNBETA = 3.5,                           | Troyon g coefficient
HFACT = 2.,                             | N.B. active iteration variable 10
ICURR = 4,                              | Use ITER current scaling
ISC = 6,                                | Use ITER 89-P confinement time scaling law
ISTOK = 2,                              | Use conventional tokamak divertor model
IINVQD = 1,                             | Use inverse quadrature
IITER = 1,                              | Use ITER fusion power calculations
ISHAPE = 0,                             | Use input values for KAPPA and TRIANG
KAPPA = 2.218,                          | Plasma elongation
```

```
Q = 3.0,                    | Edge safety factor
RMAJOR = 7.0,               | N.B. active iteration variable 3
RNBEAM = 0.0002,            | N.B. active iteration variable 7
TE = 15.,                   | Electron temperature
TRIANG = 0.6                | Plasma triangularity
$END                        | End of input block PHYDAT
                            |
$CDDAT                      | Start of input block CDDAT (current drive)
IRFCD = 1,                  | Use current drive
IEFRF = 5                   | Use ITER neutral beam current drive
FEFFCD = 3.,                | Artificially enhance efficiency
$END                        | End of input block CDDAT
                            |
$TIME                       | Start of input block TIME (times)
TBURN = 227.9               | Burn time
$END                        | End of input block TIME
                            |
$DIVT                       | Start of input block DIVT (divertor)
ANGINC=0.262,               | Angle of incidence of field lines on plate
PRN1=0.285                  | Density ratio
$END                        | End of input block DIVT
                            |
$BLD                        | Start of input block BLD (machine build)
BORE = 0.12,                | Machine bore
OHCTH = 0.1,                | OH coil thickness
GAPOH = 0.08,               | Inboard gap
TFCTH = 0.9,                | Inboard TF coil leg thickness
DDWI = 0.07,                | Internal dewar thickness
SHLDITH = 0.69,             | Inboard shield thickness
BLNKITH = 0.115,            | Inboard blanket thickness
FWITH = 0.035,              | Inboard first wall thickness
SCRAPLI = 0.14,             | Inboard scrape-off layer thickness
SCRAPLO = 0.15,             | Outboard scrape-off layer thickness
FWOTH = 0.035,              | Outboard first wall thickness
BLNKOTH = 0.235,            | Outboard blanket thickness
SHLDOTH = 1.05,             | Outboard shield thickness
GAPOMIN = 0.21,             | Outboard gap
VGAPTF = 0,                 | Vertical gap
IOHCIE = 2                  | Use conventional machine layout
$END                        | End of input block BLD
                            |
$TFC                        | Start of input block TFC (TF coils)
OACDCP = 1.4e7,             | N.B. active iteration variable 12
ITFSUP = 1,                 | Use superconducting TF coils
RIPMAX = 5.,                | Maximum TF ripple
$END                        | End of input block TFC
                            |
$PFC                        | Start of input block PFC (PF coils)
ISTOKPF = 2,                | Use conventional PF coil current scaling
```

```
NGRP = 3,                          | Three groups of PF coils
IPFLOC = 1,2,3,                    | Locations for each group
NCLS = 2,2,2,1,                    | Number of coils in each group
COHEOF = 1.85e7,                   | OH coil current at End Of Flat-top
FCOHBOP = 0.9,                     | OH coil current at Begin. Of Pulse / COHEOF
ROUTR = 1.5,                       | Radial position for group 3
ZREF(3) = 2.5,                     | Z position for group 3
OHHGHF = .71                       | Height ratio OH coil / TF coil
$END                               | End of input block PFC
                                   |
$VOLTS                             | Start of input block VOLTS (empty)
$END                               | End of input block VOLTS
                                   |
$FWBLSH                            | Start of input block FWBLSH (1st wall etc.)
DENSTL=7800.                       | Steel density
$END                               | End of input block FWBLSH
                                   |
$COSTINP                           | Start of input block COSTINP (costs)
IREACTOR = 1,                      | Calculate cost of elctricity
IFUELTYP = 0                       | Treat blanket, first wall etc as capital cost
$END                               | End of input block COSTINP
                                   |
$UCSTINP                           | Start of input block UCSTINP (unit costs)
UCHRS = 87.9,                      | }
UCCPCL1 = 250,                     | } Unit costs
UCCPCLB = 150                      | }
$END                               | End of input block UCSTINP
                                   |
$SWEP                              | Start of input block SWEP (scans)
ISWEEP=0                           | No scans (non-optimisation mode)
$END                               | End of input block SWEP
                                   |
$BCOM                              | Start of input block BCOM (buildings)
FNDT = 2.                          | Foundation thickness
$END                               | End of input block BCOM
                                   |
$BLDINP                            | Start of input block BLDINP (buildings)
EFLOOR=1.d5                        | Effective total floor space
$END                               | End of input block BLDINP
                                   |
$HTPWR                             | Start of input block HTPWR (heat transport)
ETATH=0.35                         | Thermal to electric conversion efficiency
$END                               | End of input block HTPWR
                                   |
$HTTINP                            | Start of input block HTTINP (heat transport)
FMGDMW = 0.                        | Power to MGF units
$END                               | End of input block HTTINP
                                   |
$HTRINP                            | Start of input block HTRINP (heat transport)
```

```
BASEEL=5.e6                          | Base plant electric load
$END                                 | End of input block HTRINP
                                     |
$EST                                 | Start of input block EST (energy storage)
ISCENR=2                             | Energy store option
$END                                 | End of input block EST
                                     |
$VACCY                               | Start of input block VACCY (vacuum system)
NTYPE = 1                            | Use cryopump
$END                                 | End of input block VACCY
                                     |
$OSECTS                              | Start of input block OSECTS (output sections)
SECT01 = 1,                          | }
SECT02 = 1,                          | }
SECT03 = 1,                          | }
SECT04 = 1,                          | }
SECT05 = 1,                          | }
SECT06 = 1,                          | }
SECT07 = 1,                          | }
SECT08 = 1,                          | }
SECT09 = 1,                          | }
SECT10 = 1,                          | } Turn on all output sections
SECT11 = 1,                          | }
SECT12 = 1,                          | }
SECT13 = 1,                          | }
SECT14 = 1,                          | }
SECT15 = 1,                          | }
SECT16 = 1,                          | }
SECT17 = 1,                          | }
SECT18 = 1,                          | }
SECT19 = 1,                          | }
SECT20 = 1                           | }
$END                                 | End of input block OSECTS
```

Appendix B

Optimisation Input File

The following is a typical input file used to run *PROCESS* in optimisation mode. Comments have been added to the right of each line.

```
$inpt1                                  | Start of input block INPT1 (numerics)
boundl(1) = 2.5,                        | }
BOUNDU(10) = 3.,                        | } bounds on iteration variables
BOUNDU(60) = 4.d4,                      | }
NEQNS = 15,                             | Number of active constraint equations
NVAR = 25                               | Number of active iteration variables
ICC =   1,  2, 10, 11, 7, 16, 8, 24, 31, 32, 33, 34, 35, 36, 14, | Constraint eqns
ixc =   5, 10, 12, 3,  7,        36, 48, 49, 50, 51, 53, 54, 19, | Corresponding
        1, 2, 4, 6, 13, 16, 29, 56, 57, 58, 59, 60,             | iteration variables
IOPTIMZ = 1,                            | Turn on optimisation
MINMAX = 6,                             | Minimise cost of electricity
$end                                    | End of input block INPT1
                                        |
$INEQDAT                                | Start of input block INEQDAT (f-values etc)
FBETATRY = 1.0                          | N.B. active iteration variable 36
$end                                    | End of input block INEQDAT
                                        |
$PHYDAT                                 | Start of input block PHYDAT (physics)
ASPECT = 3.5,                           | N.B. active iteration variable 1
BETA = 0.042,                           | N.B. active iteration variable 5
BT = 6.,                                | N.B. active iteration variable 2
DENE = 1.5e20,                          | N.B. active iteration variable 6
FVSBRNNI = 1.0,                         | Non-inductive volt seconds fraction
DNBETA = 3.5,                           | Troyon g coefficient
HFACT = 2.,                             | N.B. active iteration variable 10
ICURR = 4,                              | Use ITER current scaling
ISC = 6,                                | Use ITER 89-P confinement time scaling law
```

```
ISTOK = 2,                  | Use conventional tokamak divertor model
IINVQD = 1,                 | Use inverse quadrature
IITER = 1,                  | Use ITER fusion power calculations
ISHAPE = 0,                 | Use input values for KAPPA and TRIANG
KAPPA = 2.218,              | Plasma elongation
Q = 3.0,                    | Edge safety factor
RMAJOR = 7.0,               | N.B. active iteration variable 3
RNBEAM = 0.0002,            | N.B. active iteration variable 7
TE = 15.,                   | N.B. active iteration variable 4
TRIANG = 0.6                | Plasma triangularity
$END                        | End of input block PHYDAT
                            |
$CDDAT                      | Start of input block CDDAT (current drive)
IRFCD = 1,                  | Use current drive
IEFRF = 5                   | Use ITER neutral beam current drive
FEFFCD = 3.,                | Artificially enhance efficiency
$END                        | End of input block CDDAT
                            |
$TIME                       | Start of input block TIME (times)
TBURN = 227.9               | Burn time
$END                        | End of input block TIME
                            |
$DIVT                       | Start of input block DIVT (divertor)
ANGINC=0.262,               | Angle of incidence of field lines on plate
PRN1=0.285                  | Density ratio
$END                        | End of input block DIVT
                            |
$BLD                        | Start of input block BLD (machine build)
BORE = 0.12,                | N.B. active iteration variable 29
OHCTH = 0.1,                | N.B. active iteration variable 16
GAPOH = 0.08,               | Inboard gap
TFCTH = 0.9,                | N.B. active iteration variable 13
DDWI = 0.07,                | Internal dewar thickness
SHLDITH = 0.69,             | Inboard shield thickness
BLNKITH = 0.115,            | Inboard blanket thickness
FWITH = 0.035,              | Inboard first wall thickness
SCRAPLI = 0.14,             | Inboard scrape-off layer thickness
SCRAPLO = 0.15,             | Outboard scrape-off layer thickness
FWOTH = 0.035,              | Outboard first wall thickness
BLNKOTH = 0.235,            | Outboard blanket thickness
SHLDOTH = 1.05,             | Outboard shield thickness
GAPOMIN = 0.21,             | Outboard gap
VGAPTF = 0,                 | Vertical gap
IOHCIE = 2                  | Use conventional machine layout
$END                        | End of input block BLD
                            |
$TFC                        | Start of input block TFC (TF coils)
OACDCP = 1.4e7,             | N.B. active iteration variable 12
ITFSUP = 1,                 | Use superconducting TF coils
```

```
RIPMAX = 5.,                          | Maximum TF ripple
$END                                  | End of input block TFC
                                      |
$PFC                                  | Start of input block PFC (PF coils)
ISTOKPF = 2,                          | Use conventional PF coil current scaling
NGRP = 3,                             | Three groups of PF coils
IPFLOC = 1,2,3,                       | Locations for each group
NCLS = 2,2,2,1,                       | Number of coils in each group
COHEOF = 1.85e7,                      | OH coil current at End Of Flat-top
FCOHBOP = 0.9,                        | OH coil current at Begin. Of Pulse / COHEOF
ROUTR = 1.5,                          | Radial position for group 3
ZREF(3) = 2.5,                        | Z position for group 3
OHHGHF = .71                          | Height ratio OH coil / TF coil
$END                                  | End of input block PFC
                                      |
$VOLTS                                | Start of input block VOLTS (empty)
$END                                  | End of input block VOLTS
                                      |
$FWBLSH                               | Start of input block FWBLSH (1st wall etc.)
DENSTL=7800.                          | Steel density
$END                                  | End of input block FWBLSH
                                      |
$COSTINP                              | Start of input block COSTINP (costs)
IREACTOR = 1,                         | Calculate cost of elctricity
IFUELTYP = 0                          | Treat blanket, first wall etc as capital cost
$END                                  | End of input block COSTINP
                                      |
$UCSTINP                              | Start of input block UCSTINP (unit costs)
UCHRS = 87.9,                         | }
UCCPCL1 = 250,                        | } Unit costs
UCCPCLB = 150                         | }
$END                                  | End of input block UCSTINP
                                      |
$SWEP                                 | Start of input block SWEP (scans)
ISWEEP=7,                             | Seven point scan
NSWEEP=6,                             | Use WALALW as scanning variable
SWEEP= 6.0,5.5,4.5,4.0,3.5,3.0,2.5    | Values of WALALW for each scan point
$END                                  | End of input block SWEP
                                      |
$BCOM                                 | Start of input block BCOM (buildings)
FNDT = 2.                             | Foundation thickness
$END                                  | End of input block BCOM
                                      |
$BLDINP                               | Start of input block BLDINP (buildings)
EFLOOR=1.d5                           | Effective total floor space
$END                                  | End of input block BLDINP
                                      |
$HTPWR                                | Start of input block HTPWR (heat transport)
ETATH=0.35                            | Thermal to electric conversion efficiency
```

```
$END                                    | End of input block HTPWR
                                        |
$HTTINP                                 | Start of input block HTTINP (heat transport)
FMGDMW = 0.                             | Power to MGF units
$END                                    | End of input block HTTINP
                                        |
$HTRINP                                 | Start of input block HTRINP (heat transport)
BASEEL=5.e6                             | Base plant electric load
$END                                    | End of input block HTRINP
                                        |
$EST                                    | Start of input block EST (energy storage)
ISCENR=2                                | Energy store option
$END                                    | End of input block EST
                                        |
$VACCY                                  | Start of input block VACCY (vacuum system)
NTYPE = 1                               | Use cryopump
$END                                    | End of input block VACCY
                                        |
$OSECTS                                 | Start of input block OSECTS (output sections)
SECT01 = 1,                             | }
SECT02 = 1,                             | }
SECT03 = 1,                             | }
SECT04 = 1,                             | }
SECT05 = 1,                             | }
SECT06 = 1,                             | }
SECT07 = 1,                             | }
SECT08 = 1,                             | }
SECT09 = 1,                             | }
SECT10 = 1,                             | } Turn on all output sections
SECT11 = 1,                             | }
SECT12 = 1,                             | }
SECT13 = 1,                             | }
SECT14 = 1,                             | }
SECT15 = 1,                             | }
SECT16 = 1,                             | }
SECT17 = 1,                             | }
SECT18 = 1,                             | }
SECT19 = 1,                             | }
SECT20 = 1                              | }
$END                                    | End of input block OSECTS
```

Appendix C

Source Code Documentation

The development of the *PROCESS* code since its shipment from Oak Ridge National Laboratory in April 1992 has been fully documented in the Project Work File [15]. Presented here is a list of Project Work File Notes as of July 23, 1993 that address various issues related to the source code.

- Documentation of each individual source routine is an ongoing task. A Work File Note will be produced as each routine is processed, with the eventual aim of bringing all these together into a single document, i.e. a second volume of this manual.
- Summary of work performed since April 1992 : Note 0160
- Code status (routine SCCS version numbers) : Note 0165
- SCCS (Source Code Control System) implementation for *PROCESS* : Note 0003
- Present code standard : Note 0160
- Future code standard (to be adhered to) : Note 0167
- Directory structure and location of all relevant files : Note 0168
- Proposed future work : Note 0160