

## Equations of the Aries Systems Code

**Revision:** 2/2012 by L. Carlson

**Code Version:** ASC v.35

**Code Modules:** *ALL modules*

**History:** The code has been documented before but has failed to clearly define and reference all the equations that reside within it. The Fusion Engineering and Design journal article by Z. Dragojlovic, et al “[An Advanced Computational Algorithm for systems analysis of tokamak power plants](#),” FED 85 (2010) 243-265 is a starting point for the equations of the ASC.

**Notes:** This document attempts to account for all the equations that reside within the systems code which are hidden in and amongst all the modules. The mentality with which this is written is “How would you write the code from scratch? What equations are necessary to construct the parts of a tokamak power plant and what information do you need to build each part? What do you need from other modules? What are the references and where is the documentation found?” Costing Account Equations can be found in the document **CostingAccounts.cpp**.

### Table of Contents

|                                                                             |           |
|-----------------------------------------------------------------------------|-----------|
| <b>1. Physics Module Equations .....</b>                                    | <b>2</b>  |
| <b>2. Building Volumes.....</b>                                             | <b>2</b>  |
| <b>3. Blanket Pumping Power .....</b>                                       | <b>3</b>  |
| <b>4. Divertor Pumping Power .....</b>                                      | <b>4</b>  |
| <b>5. Divertor Heat Flux .....</b>                                          | <b>4</b>  |
| <b>6. Power Summation and Accountability .....</b>                          | <b>6</b>  |
| <b>7. Gross Cycle Efficiency of the Brayton Power Conversion Cycle.....</b> | <b>11</b> |
| <b>8. Current Drive.....</b>                                                | <b>12</b> |
| <b>9. Parts .....</b>                                                       | <b>13</b> |
| A. Plasma Surface Area .....                                                | 13        |
| B. Last Closed Magnet Surface (Sepatrix) .....                              | 14        |
| C. First Wall IB Geometry .....                                             | 15        |
| D. First Wall OB Geometry .....                                             | 16        |
| E. Blanket IB Geometry.....                                                 | 16        |
| F. Blanket OB Geometry .....                                                | 16        |
| G. Back Wall IB Geometry (DCLL) .....                                       | 17        |
| H. Back Wall OB Geometry (DCLL) .....                                       | 17        |
| I. Blanket II Geometry (SCLL) .....                                         | 17        |
| J. Divertor Plates (SCLL).....                                              | 18        |
| K. Divertor Plates (DCLL) .....                                             | 19        |
| L. Replaceable HT Shield .....                                              | 20        |
| M. HT Shield.....                                                           | 21        |
| N. Skeleton Ring (DCLL) .....                                               | 22        |
| O. Manifold (DCLL OB and vert).....                                         | 22        |
| P. Manifolding (SCLL) .....                                                 | 22        |
| Q. Vacuum Vessel (SCLL).....                                                | 23        |
| R. Vacuum Vessel (DCLL) .....                                               | 24        |
| S. TF Magnet.....                                                           | 26        |
| T. Bucking Cylinder.....                                                    | 30        |

|                                            |    |
|--------------------------------------------|----|
| U. Upper Toroidal Cap on the TF Coil ..... | 30 |
| V. Lower Toroidal Cap on the TF Coil ..... | 31 |
| W. PF Coil Design .....                    | 32 |
| X. Cryogenic Dome .....                    | 35 |
| Y. Maintenance Port Window .....           | 36 |
| Z. Maintenance Ports .....                 | 36 |
| AA. Upper Vacuum Vessel Duct .....         | 37 |
| ...END of Parts ... ..                     | 39 |

## 1. Physics Module Equations

**Reference:** Z. Dragojlovic et al. 1/2010 [\*An Advanced Computational Algorithm for systems analysis of tokamak power plants\*](#), "FED 85 (2010) 243-265.

**Code Modules:** *inpar.txt* & *inpar2.txt* (input files), *corona* (plasma equilibrium look-up table), *makefile* (instructions to build the executable), *PlasmaPhysics.exe* (executable), *runphysics.sh* (run command shell), *sysplas\_PC.F* (source code)

**Notes:** The author of the physics module is C. Kessel. It is the first module that is run when utilizing the Aries Systems Code. It scans the input files and creates a listing of viable physics operating points. The physics module must be compiled on a Linux machine with g++ and gfortran compilers.

The specific equations contained in the source code *sysplas\_PC.F* are lengthy and beyond the scope of this overview document. For details regarding the equations, the author C. Kessel or the systems code administrator should be contacted. The physics module is described in detail in the physics section of the Fusion Engineering and Design journal article as referenced above. See also the Physics Module on the Documentation Homepage.

## 2. Building Volumes

**Reference:** *Already in code, no reference.*

**Code Modules:** *DesignPoint.cpp*

**Notes:** Volumes are computed for costing specifications in *CostingAccounts.cpp*

### A. Reactor Building Volume:

Calculates the exterior reactor building volume in m<sup>3</sup> from the cryodome part.

- $\text{vol\_reactor\_bldg} = (\text{pow}((\text{ksihi} + 9.0), 2.0) * 6.0 * \text{etahi} + 1.55\text{e}5) * \text{cATfit};$ 
  - $\text{cATfit} = 138698.0 / 181655.0$  (investigation found that 138698 is Aries-AT volume from published literature and 181655 is Prometheus-HI volume, no reference)
  - $\text{ksihi}$  = outboard max width of the cryodome
  - $\text{etahi}$  = vertical max height of the cryodome

### B. Support Structure Building Volume:

Approximated as 20% of the fusion power core volume.

- $\text{vol\_str} = 0.20 * \text{FPC\_vol};$

### 3. Blanket Pumping Power

#### **SCLL Blanket**

**Reference:** M. Tillack 3/2011 "SiC Blanket Pumping Power.xls"

**Code Modules:** DesignPoint.cpp

**Notes:** SCLL = no helium, liquid metal-cooled only. The old method simply divided the pumping power 50:50 between the blanket and divertor. The correlation below only calculates the IB first wall and blanket pumping power now. The OB will be much lower due to lower field strength and more space available. Therefore, the code only uses P\_rad\_first\_wall\_IB and NWL\_ave\_IB to calculate P\_blanket\_IB. The output is filtered out if the average neutron wall flux exceeds the data range given in the spreadsheet, which is 2 – 4 MW/m<sup>2</sup>.

- $P_{\text{pump\_blanket}} = (0.43840 * (\text{flux\_neutron\_wall\_Ave}/1.0\text{e}6) + 0.00036) * P_{\text{blanket\_sic}} / 100.0;$ 
  - $P_{\text{rad\_first\_wall\_IB}} = (f_{\text{rad\_blanket\_ib}} * P_{\text{rad\_chamb}} + (1.0 - f_{\text{rad\_edge}}) * P_{\text{rad\_edge}} * f_{\text{rad\_edge\_blanket\_ib}});$
  - $A_{\text{FW\_IB}} = 16.0/2.0 * ( (*\text{FirstWallInb}).\text{surface\_area}(\text{"outer"}) );$
  - $\text{flux\_neutron\_wall\_IB} = P_{\text{neutron}} * f_{\text{n\_blanket\_ib}} / A_{\text{FW\_IB}};$
  - $P_{\text{blanket\_sic}} = (1.0 - f_{\text{alpha\_div}}) * P_{\text{alpha\_loss}} + f_{\text{n\_blanket\_ib}} * P_{\text{neutron}} * f_{\text{nmul\_fw\_inboard}} + P_{\text{rad\_first\_wall\_IB}};$
  - $P_{\text{blanket}} = P_{\text{blanket\_sic}};$
  - $P_{\text{pump\_blanket\_loss}} = (1.0 - \text{eta\_pump\_lm\_mech}) * P_{\text{pump\_blanket}} / \text{eta\_pump\_lm\_mech};$

#### **DCLL Blanket**

**Reference:** M. Tillack 1/2011 "DCLL Blanket Pumping Power.xls"

**Code Modules:** DesignPoint.cpp

**Notes:** SCLL = liquid-metal and helium cooling. The correlation below combines both the liquid-metal and helium cooling, as well as the inboard and outboard pumping powers. The output is filtered out if the average neutron wall flux exceeds the data range given in the reference spreadsheet, which is 2 – 4 MW/m<sup>2</sup>.

- $P_{\text{pump\_blanket}} = (0.03602 * \text{pow}(\text{flux\_neutron\_wall\_Ave}/1.0\text{e}6, 2) + 0.03056 * (\text{flux\_neutron\_wall\_Ave}/1.0\text{e}6) + 0.38359) * P_{\text{blanket\_dcll}}/100.0;$ 
  - $P_{\text{blanket\_dcll}} = (1.0 - f_{\text{alpha\_div}}) * P_{\text{alpha\_loss}} + (f_{\text{n\_blanket\_ib}} * P_{\text{neutron}} * f_{\text{nmul\_fw\_inboard}} + (f_{\text{n\_blanket\_ob}} * P_{\text{neutron}} * f_{\text{nmul\_fw\_outboard}} + P_{\text{rad\_first\_wall}}); // \text{both IB+OB}$

// Need coolant fractional power split for DCLL blanket pump losses (LM pump vs He-blower); M. Tillack's spreadsheet has PbLi 62.3%, He 37.7%

  - $\text{double } f_{\text{pump\_PbLi}} = 0.623;$
  - $\text{double } f_{\text{pump\_He}} = 0.377;$
  - $P_{\text{pump\_blanket\_loss}} = ((1.0 - \text{eta\_pump\_lm\_mech}) * f_{\text{pump\_PbLi}} + (1.0 - \text{eta\_pump\_he}) * f_{\text{pump\_He}}) * P_{\text{pump\_blanket}} / ((\text{eta\_pump\_he} + \text{eta\_pump\_lm\_mech})/2);$

## 4. Divertor Pumping Power

**Reference:** M. Tillack, X. Wang, "Divertor Pumping Powers - ppcompare2.xls" Dec. 2010

**Code Modules:** DesignPoint.cpp

**Notes:** This calculates the pumping power requirement for the divertor in use. The divertor in use is selected from the input file Setup.data. Note that P\_thermal\_divertor is used, not the overall P\_thermal. The output is filtered if the maximum heat flux on the divertor is less than 5 or greater than 15 MW/m<sup>2</sup>, which is the range of the tabulated reference data.

- $$\begin{aligned} P_{\text{thermal\_divertor}} &= P_{\text{alpha\_loss}} * f_{\text{alpha\_div}} \\ &+ P_{\text{neutron}} * f_{\text{n\_div}} * f_{\text{nmul\_divertor}} \\ &+ P_{\text{rad\_edge}} * f_{\text{rad\_edge}} \\ &+ P_{\text{rad\_chamb}} * f_{\text{rad\_div}} \\ &+ P_{\text{cond\_div}}; \end{aligned}$$
- $$P_{\text{pump\_divertor}} = \text{pumping\_power\_divertor}(\text{flux\_divertor\_Max}) * P_{\text{thermal\_divertor}}/100.0;$$
- $$P_{\text{pump\_divertor\_loss}} = (1.0 - \text{eta\_pump\_he}) * P_{\text{pump\_divertor}}/\text{eta\_pump\_he};$$

### 1. T-tube design:

J. Burke's T-Tube 2010 divertor design pumping power curve. Linear-taper with fixed slot width of 0.45 mm, inlet temp 600C, outlet 677C, P = 10 MPa; surface heat flux (MW/m<sup>2</sup>) vs P\_pump/P\_thermal (%).

- $$\begin{aligned} \text{pptube} &= 0.05594 * \text{pow}(\text{flux\_divertor\_Max}/1.0\text{e}6, 2) + 0.04931 * \\ &(\text{flux\_divertor\_Max}/1.0\text{e}6) - 0.08702; \end{aligned}$$

### 2. Plate design:

X. Wang's plate divertor design (HCFP). Helium-cooled flat plate, inlet temp 600C, outlet 677C, P = 10 MPa.

- $$\begin{aligned} \text{ppplate} &= 0.0392 * \text{pow}(\text{flux\_divertor\_Max}/1.0\text{e}6, 3) - 0.743 * \\ &\text{pow}(\text{flux\_divertor\_Max}/1.0\text{e}6, 2) + 5.44 * (\text{flux\_divertor\_Max}/1.0\text{e}6) - 11.1; \end{aligned}$$

### 3. Combined plate-finger design:

X. Wang's finger divertor design (HCPF). Helium-cooled combined plate and finger, inlet temp 600C, outlet 700C, P = 10 MPa.

- $$\begin{aligned} \text{ppfinger} &= 0.0335 * \text{pow}(\text{flux\_divertor\_Max}/1.0\text{e}6, 2) + 0.22 * (\text{flux\_divertor\_Max}/1.0\text{e}6) \\ &- 0.892; \end{aligned}$$

## 5. Divertor Heat Flux

### Max Heat Flux on Divertor

**Reference:** C. Kessel, existing code, no specific reference. Edited "adivo" and "adivi" with C. Kessel 1/2011

**Code Modules:** DesignPoint.cpp

**Notes:** Find the maximum heat flux incident on the divertor. This section of the code stems from C. Kessel's sysengr.f systems code. The output is filtered if the maximum heat flux on the divertor is > 8 MW/m<sup>2</sup>.

### Definitions:

- sym = 1.0 for up-down symmetric, 0.0 for up-down asymmetric
- facdiv = factor for splitting power if up-down symmetric or not
- adivo = area in outboard divertor slot for absorbing radiated power

- `adivi` = area in inboard divertor slot for absorbing radiated power
- `fdivrad` = total radiated power fraction in divertor
- `fdivcondo` = fraction of conducted power to divertor going to the outboard
- `fdivcondi` = fraction of conducted power to divertor going to the inboard
- `fdivrado` = fraction of radiated power to divertor going to the outboard
- `fdivradi` = fraction of radiated power to divertor going to the inboard
- `fluxexpo` = poloidal mag. flux exp. on outboard from midplane to divertor target
- `fluxexpi` = poloidal mag. flux exp. on inboard from midplane to divertor target
- `pdiv` = total power going to divertors
- `pdivrad` = total power radiated in divertors
- `pdiv2` = total power conducted to divertors
- `pdiv2o` = power conducted to outboard divertor
- `pdiv2i` = power conducted to inboard divertor
- `pdivrado` = power radiated in outboard divertor
- `pdivradi` = power radiated in inboard divertor
- `pdivtoto` = conducted + radiated power in outboard divertor
- `pdivtoti` = conducted + radiated power in inboard divertor
- `den` = density factor in power scrape-off formula
- `xlam` = power scrape-off layer width at plasma midplane
- `qpeakcondo` = peak conducted power heat flux to outboard divertor
- `qpeakcondi` = peak conducted power heat flux to inboard divertor
- `qpeakrado` = peak radiated power heat flux to outboard divertor
- `qpeakradi` = peak radiated power heat flux to inboard divertor
- `f_greenwald` = greenwald fraction
- `Bt` = toroidal magnetic field at plasma major radius
- `Aip` = plasma current
- `Qmhd` = `qmhd`

#### Equations:

- `if(sym == 1.0)` { `facdiv = 0.65;` } `else` { `facdiv = 1.00;` }
- `Rmaj` = `plasma_major_radius()`;
- `Asp` = `plasma_major_radius()/plasma_minor_radius()`;
- `adivo` = `2.*pi*(Rmaj - ((Rmaj/Asp)/2.0))*2.0*(Rmaj/(Asp*2.0))`;
- `adivi` = `2.*pi*(Rmaj - (Rmaj/Asp))*2.0*(Rmaj/(Asp*4.0))`;
- `P_div` = `(P_plasma - P_rad_first_wall)/1.e6`;
- `P_div_rad` = `f_div_rad*P_div`;
- `P_div_2` = `P_div - P_div_rad`;
- `P_div_2_o` = `facdiv*fdivcondo*P_div_2`;
- `P_div_2_i` = `facdiv*fdivcondi*P_div_2`;
- `P_div_rad_o` = `facdiv*fdivrado*P_div_rad`;
- `P_div_rad_i` = `facdiv*fdivradi*P_div_rad`;
- `den` = `f_greenwald*(Aip/pi*pow(Rmaj/Asp, 2.0))*0.35`;
- `xlam` = `2.9e-2*2.5*pow(Qmhd, 0.75)*pow(den, 0.15)/(pow(P_div_2_o, 0.4)*Bt)`; // power scrape off width
- `Q_peak_cond_o` = `P_div_2_o/(2.0*pi*(Rmaj - ((Rmaj/Asp)/2.0))*fluxexpo*xlam)`;

- $Q\_peak\_cond\_i = P\_div\_2\_i / (2.0 * \pi * (R_{maj} - (R_{maj}/Asp))) * flux_{expi} * x_{lam};$
- $Q\_peak\_rad\_o = P\_div\_rad\_o / a_{divo};$
- $Q\_peak\_rad\_i = P\_div\_rad\_i / a_{divi};$
- $Q\_peak\_tot\_o = Q\_peak\_cond\_o + Q\_peak\_rad\_o;$
- $Q\_peak\_tot\_i = Q\_peak\_cond\_i + Q\_peak\_rad\_i;$
- `get_divertor_loads(Q_peak_tot_i*1.e6, Q_peak_tot_o*1.e6); // Get's peak divertor loads from a simple comparison`
- $flux\_divertor\_Max = \max(Q\_peak\_tot\_o, Q\_peak\_tot\_i) * 1.e6;$

## 6. Power Summation and Accountability

**Reference:** *This already existed in the code; L. Carlson added accounting for losses and verified power balance 8/2011.*

**Code Modules:** *DesignPoint.cpp, powerflow.data input file*

**Notes:** These equations describe the net electric power and power losses. See also the [Power Flow Diagram](#) on the Documentation Homepage for a visual representation. There is a check in the code to ensure that the power generated is equal to the net electric power plus the unrecoverable power lost through heat, friction, or inefficiencies.

### Definitions:

#### Power Conversion Factors (from powerflow.data input file)

- $f\_alpha\_loss$  = fraction of loss of alfa particles; = 0 for tokamak
- $f\_alpha\_div$  = fraction of alpha loss to divertor
- $f\_div\_rad$  = fraction of edge radiation from particles
- $f\_rad\_edge$  = fraction of edge radiation to divertor
- *Fraction of edge radiation to FWBS =  $(1.0 - f\_rad\_edge)$ ; IB/OB split of edge radiation to FWBS is dependent on FW area*
- $f\_rad\_edge\_blanket\_ib = (*FirstWallInb).surface\_area("outer") / ((*FirstWallInb).surface\_area("outer") + (*FirstWallOutb).surface\_area("outer"));$
- $f\_rad\_edge\_blanket\_ob = (1.0 - f\_rad\_edge\_blanket\_ib);$

#### Neutron Energy Multiplication (all M = 1.1 for now)

- $f\_nmul\_fw\_inboard$  = FW/blanket IB
- $f\_nmul\_fw\_outboard$  = FW/blanket OB
- $f\_nmul\_shield\_inboard$  = shield IB
- $f\_nmul\_shield\_outboard$  = shield OB
- $f\_nmul\_divertor$  = divertor plates
- $f\_nmul\_div\_blanket$  = blanket behind divertor
- $f\_nmul\_div\_shield$  = shield behind divertor

#### Efficiency of Pumps – He, liquid-metal MHD, liquid-metal mechanical

- $\eta_{pump\_he}$  = efficiency of the He pump/blower
- $\eta_{pump\_he\_recovered}$  = fraction of heat recoverable
- $\eta_{pump\_lm\_mhd}$  = efficiency of the Liquid Metal MHD pump
- $\eta_{pump\_lm\_mhd\_recovered}$  = fraction of heat recoverable
- $\eta_{pump\_lm\_mech}$  = efficiency of the Liquid Metal mechanical pump

- $\eta_{\text{pump\_lm\_mech\_recovered}}$  = fraction of heat recoverable

## **Powers**

### **Power from Neutron Particles**

- $P_{\text{alpha}}$  = inputdata[npt][53]\*1.0e6; // *alpha power contained in plasma*
- $P_{\text{neutron}}$  = 4.0\* $P_{\text{alpha}}$ ;
- $P_{\text{fusion}}$  =  $P_{\text{alpha}}$  +  $P_{\text{neutron}}$ ; //  *$P_{\text{fusion}} = 5*P_{\text{alpha}}$*

### **Auxiliary Power**

*L. El-Guebaly recommends 65 MW = 30 misc + 35 MW cryo load (Dec 2010), scaling from ITER. Also recommends 2 MW for low temperature superconductor and 0.5 MW for high temperature SC*

- $P_{\text{aux\_func}}$  = power to auxiliary functions
- $P_{\text{cryo}}$  = cryogenic magnet power

### **Plasma Heating Power**

- $\eta_{\text{plasma\_heating}}$  = aux. power into plasma for power balance
- $P_{\text{aux\_for\_power\_balance}}$  = (inputdata[npt][26]) \* 1.0e6; // *incl. CD power*
- $P_{\text{plasma\_heating}}$  =  $P_{\text{aux\_for\_power\_balance}}$  -  $P_{\text{cd\_generic}}$ ; // *MW*
- $P_{\text{plasma\_heating\_loss}}$  = (1.0 -  $\eta_{\text{plasma\_heating}}$ ) \*  $P_{\text{plasma\_heating}}$  /  $\eta_{\text{plasma\_heating}}$ ;

### **Power Radiated from Chamber**

*Bremsstrahlung radiation + Cyclotron power + Line radiation*

- $P_{\text{rad\_chamb}}$  = (inputdata[npt][22] + inputdata[npt][27] + inputdata[npt][29])\*1.0e6;

### **Current Drive Power**

*Current drive power is listed in its own section titled "Current Drive."*

### **Power From Alpha Particles**

*If there is any alpha loss, it would go into the FW and be recovered as heat*

- $P_{\text{alpha\_loss}}$  =  $f_{\text{alpha\_loss}}$  \*  $P_{\text{alpha}}$ ; // = 0 for a tokamak
- $P_{\text{transport}}$  =  $P_{\text{alpha}}$  -  $P_{\text{alpha\_loss}}$ ;
- $P_{\text{plasma}}$  =  $P_{\text{transport}}$  +  $P_{\text{aux\_for\_power\_balance}}$ ; // *in MW*
- $P_{\text{particle}}$  =  $P_{\text{plasma}}$  -  $P_{\text{alpha\_loss}}$  -  $P_{\text{rad\_chamb}}$ ;
- $P_{\text{rad\_edge}}$  =  $f_{\text{div\_rad}}$  \*  $P_{\text{particle}}$ ; // *Edge radiation power*
- $P_{\text{cond\_div}}$  = (1.0 -  $f_{\text{div\_rad}}$ ) \*  $P_{\text{particle}}$ ; // *Edge conducted power*

### **Average Neutron Wall Flux "NWL\_ave"**

*C.Kessel: neutron wall load = 4/5\* $P_{\text{fusion}}/A_{\text{plasma}}$  SA =  $P_{\text{neutron}}/A_{\text{plasma\_surface}}$ . Neutron wall loading expresses the neutron wall flux against the wall divided by the surface area of the plasma.*

- $\text{flux\_neutron\_wall\_Ave}$  =  $P_{\text{neutron}}/A_{\text{plasma\_surface}}$ ; // *area of plasma surface is found "Plasma" section*
- $\text{flux\_neutron\_wall\_Ave\_w\_SOL}$  =  $P_{\text{neutron}}/A_{\text{plasma\_surface\_w\_SOL}}$ ;

### **Maximum Neutron Wall Flux Using Peaking Factor**

Two tables, one for SCLL and one for DCLL blanket, give peaking factors depending on plasma aspect ratio. If the aspect ratio is not found in the table, then the code interpolates. L. El-Guebaly updated 5/2011 at project meeting.

- DCLL: at A = 1.6, PF = 1.94; at A = 4.0, PF = 1.5
- SCLL: at A = 1.7, PF = 2.0; at A = 4.0, PF = 1.55
- flux\_neutron\_wall\_Max = flux\_neutron\_wall\_Ave \* peaking\_factor;

### **Heat Flux to First Wall IB + OB (Average and Maximum):**

L. Carlson changed old geometric factor (1.0-f\_geo\_div) to f\_rad\_blanket\_ib + f\_rad\_blanket\_ob. This has increased P\_rad\_first\_wall as well as flux\_heat\_wall.

- P\_rad\_first\_wall = (f\_rad\_blanket\_ib + f\_rad\_blanket\_ob) \* P\_rad\_chamb + (1.0 - f\_rad\_edge) \* P\_rad\_edge;
- flux\_heat\_wall\_Ave = P\_rad\_first\_wall / A\_first\_wall; // Average heat flux on the wall
- flux\_heat\_wall\_Max = 1.25 \* flux\_heat\_wall\_Ave; // Maximum heat flux on the wall

### **Neutron Power Distribution**

Neutron power distribution from plasma given from Laila, read from powerflow.data. "For A=4, 10% of power is deposited in divertor, 25% in IB blanket, and 65% in OB.

- f\_n\_div = fraction of neutron power from plasma to divertor (A=4)
- f\_n\_blanket\_ib = fraction of neutron power from plasma to IB blanket
- f\_n\_blanket\_ob = fraction of neutron power from plasma to OB blanket

### **Radiated power distribution from plasma:**

Estimated by M. Tillack, read from powerflow.data input file.

- f\_rad\_div = fraction of radiated power from plasma to divertor
- f\_rad\_blanket\_ib = fraction of radiated power from plasma to IB blanket
- f\_rad\_blanket\_ob = fraction of radiated power from plasma to OB blanket

### **Power Absorbed**

#### **A. FWBS (First-wall, blanket, shield):**

Set up array:

- P\_FWBSGroup[2][2][2];
- neutron = 0; // i's
- condRad = 1;
- fw\_blanket = 0; // j's
- shield = 1;
- outboard = 0; // k's
- inboard = 1;
- P\_lm\_pump\_recovered = P\_pump\_blanket; // LM mechanical pump's thermal contribution
- P\_neutron\_FWBSGroup\_in = (f\_n\_blanket\_ib + f\_n\_blanket\_ob) \* P\_neutron;
- P\_neutron\_FWBSGroup\_ib = f\_n\_blanket\_ib \* P\_neutron;
- P\_neutron\_FWBSGroup\_ob = f\_n\_blanket\_ob \* P\_neutron;
- P\_condRad\_FWBSGroup\_in = (1.0 - f\_alpha\_div) \* P\_alpha\_loss

$$\begin{aligned}
& + (f\_rad\_blanket\_ib + f\_rad\_blanket\_ob) & * P\_rad\_chamb \\
& + (1.0 - f\_rad\_edge) & * P\_rad\_edge;
\end{aligned}$$

*// Conduction + radiation powers are split here. They do not get deposited in the shield, only in the FW and blanket, which are the surface layers exposed to plasma.*

- $P\_FWBSGroup[condRad][fw\_blanket][outboard] = f\_rad\_blanket\_ob * P\_rad\_chamb + (1.0 - f\_rad\_edge) * P\_rad\_edge * f\_rad\_edge\_blanket\_ob;$
- $P\_FWBSGroup[condRad][fw\_blanket][inboard] = (1.0 - f\_alpha\_div) * P\_alpha\_loss + f\_rad\_blanket\_ib * P\_rad\_chamb + (1.0 - f\_rad\_edge) * P\_rad\_edge * f\_rad\_edge\_blanket\_ib;$
- $P\_FWBSGroup[condRad][shield][outboard] = 0.0;$
- $P\_FWBSGroup[condRad][shield][inboard] = 0.0;$

*// Split the IB & OB powers from neutrons into FW/blanket and shield fractions. Include the effect of neutron energy multiplication (1.1).*

$f\_fw\_blanket = 0.90;$  *// from powerflow.data input file*  
 $f\_shield = 0.10;$  *// from powerflow.data input file*

- $P\_FWBSGroup[neutron][fw\_blanket][outboard] = f\_fw\_blanket * f\_nmul\_fw\_outboard * P\_neutron\_FWBSGroup\_ob;$
- $P\_FWBSGroup[neutron][fw\_blanket][inboard] = f\_fw\_blanket * f\_nmul\_fw\_inboard * P\_neutron\_FWBSGroup\_ib;$
- $P\_FWBSGroup[neutron][shield][outboard] = f\_shield * f\_nmul\_shield\_outboard * P\_neutron\_FWBSGroup\_ob;$
- $P\_FWBSGroup[neutron][shield][inboard] = f\_shield * f\_nmul\_shield\_inboard * P\_neutron\_FWBSGroup\_ib;$

## **Power Absorbed**

### **B. Divertor:**

*Set up array:*

- $P\_DivGroup[3][4];$
- $divPlates = 0;$  *// j's*
- $divBlanket = 1;$
- $divShield = 2;$
- $double P\_he\_pump\_recovered = P\_pump\_divertor;$  *// He pump's thermal contribution*
- $double P\_neutron\_DivGroup\_in = f\_n\_div * P\_neutron;$  *// M applied below*
- $double P\_condRad\_DivGroup\_in = f\_alpha\_div * P\_alpha\_loss + f\_rad\_div * P\_rad\_chamb + f\_rad\_edge * P\_rad\_edge + P\_cond\_div;$

*// Conduction and radiation powers are only deposited in the divertor plates.*

- P\_DivGroup[condRad][divPlates] = P\_condRad\_DivGroup\_in;
- P\_DivGroup[condRad][divBlanket] = 0.0;
- P\_DivGroup[condRad][divShield] = 0.0;

*// Split the power from neutrons into fractions that go into divertor plates, blanket behind divertor and shield behind divertor. Include the effect of neutron energy multiplication. (TBD low temperature components, i.e. vacuum vessel, 1%?)*

- double f\_div\_plates = 0.50; *//from powerflow.data input file*
- double f\_div\_blanket = 0.40; *//from powerflow.data input file*
- double f\_div\_shield = 0.10; *//from powerflow.data input file*
- P\_DivGroup[neutron][divPlates] = f\_div\_plates \* f\_nmul\_divertor \* P\_neutron\_DivGroup\_in;
- P\_DivGroup[neutron][divBlanket] = f\_div\_blanket \* f\_nmul\_div\_blanket \* P\_neutron\_DivGroup\_in;
- P\_DivGroup[neutron][divShield] = f\_div\_shield \* f\_nmul\_div\_shield \* P\_neutron\_DivGroup\_in;

### **Thermal Power Summation**

#### **A. FWBS (First-wall, blanket, shield):**

- for(int i=0; i<2; i++)
- { for(int j=0; j<2; j++)
- { for(int k=0; k<2; k++)
- { P\_thermal = P\_thermal + P\_FWBSGroup[i][j][k];
- P\_thermal\_fwbs = P\_thermal\_fwbs + P\_FWBSGroup[i][j][k];

#### **B. Divertor:**

- for(int i=0; i<2; i++)
- { for(int j=0; j<3; j++)
- { P\_thermal = P\_thermal + P\_DivGroup[i][j]; *// continue to add to P\_thermal*
- P\_thermal\_div = P\_thermal\_div + P\_DivGroup[i][j]; } }
- P\_thermal\_fwbs = P\_thermal\_fwbs + P\_lm\_pump\_recovered; *// recovered power form above*
- P\_thermal\_div = P\_thermal\_div + P\_he\_pump\_recovered;
- P\_thermal = P\_thermal\_fwbs + P\_thermal\_div; *// total P\_thermal*

### **Power Flow Accountability:**

- P\_net\_electric = P\_gross\_electric –  
P\_plasma\_heating / eta\_plasma\_heating –  
P\_cd\_generic / eta\_cd\_generic –  
P\_aux\_func –  
P\_cryo –

- $$P_{\text{pump\_blanket}} / \eta_{\text{pump\_lm\_mech}} -$$

$$P_{\text{pump\_divertor}} / \eta_{\text{pump\_he}};$$
- $$P_{\text{generation}} = P_{\text{fusion}} * (0.8 * M + 0.2) / 1.0e6; // M = \text{neutron mult. factor, usually} = 1.1$$
- $$P_{\text{losses}} = P_{\text{plasma\_heating\_loss}} +$$

$$P_{\text{cd\_loss}} +$$

$$P_{\text{brayton\_loss}} +$$

$$P_{\text{pump\_blanket\_loss}} +$$

$$P_{\text{pump\_divertor\_loss}} +$$

$$P_{\text{aux\_func}} +$$

$$P_{\text{cryo}}; // + LT\_fwbs\_loss, + LT\_div\_loss // to be added$$
- $$P_{\text{leftover}} = P_{\text{generation}} - (P_{\text{net\_electric}} / 1.0e6) - (P_{\text{losses}} / 1.0e6);$$

*P\_leftover should be = 0 if all power flow is accounted for*

## 7. Gross Cycle Efficiency of the Brayton Power Conversion Cycle

**Reference:** R. Raffray, 8/2010 "Raffray - Pumping Power systems code input.xls"; polynomial fits added by L. Carlson 5/2011.

M. Tillack concluded 11/2010 that we can continue to use R. Raffray's NWL vs cycle efficiency for the near term since our NWL\_max and q" are low (~4.2 MW/m<sup>2</sup>, ~0.25 MW/m<sup>2</sup> for ~AT).

**Code Modules:** DesignPoint.cpp

**Notes:** This calculates the gross cycle efficiency of the Brayton power conversion cycle. It takes the maximum neutron wall loading in MW/m<sup>2</sup> at the first wall for different surface heat fluxes (q"max) in MW/m<sup>2</sup> and returns the gross cycle efficiency Q.

### 1. DCLL:

Choose polynomial fit depending on q"

- if (q2 < 0.250) { eta = 0; cout << "Maximum surface heat flux q" is too low for the Brayton cycle tabulated interval " << q2 << "\n"; }
- if (q2 >= 0.250 && q2 <= 0.500) { eta = 0.00013\*pow(nwl2,3) - 0.00190\*pow(nwl2,2) + 0.00087\*nwl2 + 0.44578;
- if (nwl2 < 2.0 && nwl2 > 7.5) cout << "Maximum neutron wall load (NWL\_max) at first wall is out of the Brayton cycle tabulated interval " << nwl2 << "\n"; }
- if (q2 > 0.500 && q2 <= 0.750) { eta = 0.00002\*pow(nwl2,3) - 0.00009\*pow(nwl2,2) - 0.00814\*nwl2 + 0.45770;
- if (nwl2 < 2.4 && nwl2 > 7.5) cout << "Maximum neutron wall load (NWL\_max) at first wall is out of the Brayton cycle tabulated interval " << nwl2 << "\n"; }
- if (q2 > 0.750 && q2 <= 1.000) { eta = 0.00018\*pow(nwl2,3) - 0.00275\*pow(nwl2,2) + 0.00599\*nwl2 + 0.43195;
- if (nwl2 < 4.5 && nwl2 > 7.5) cout << "Maximum neutron wall load (NWL\_max) at first wall is out of the Brayton cycle tabulated interval " << nwl2 << "\n"; }
- if (q2 > 1.000 && q2 <= 1.250) { eta = -0.00031\*pow(nwl2,2) + 0.00290\*nwl2 + 0.38929;
- if (nwl2 < 4.8 && nwl2 > 7.5) cout << "Maximum neutron wall load (NWL\_max) at first wall is out of the Brayton cycle tabulated interval " << nwl2 << "\n"; }
- if (q2 > 0.750) { eta = 0; cout << "Maximum surface heat flux q" (" << q2 << ") is too high for the Brayton cycle tabulated interval.\n"; }

## 2. SCLL:

Choose polynomial fit depending on q''

- if (q2 < 0.250) { eta = 0; cout << "Maximum surface heat flux q" is too low for the Brayton cycle tabulated interval " << q2 << "\n"; }
- if (q2 >= 0.250 && q2 <= 0.375) { eta = -0.00347\*nwl2 + 0.58859;
- if (nwl2 < 1.5 && nwl2 > 6.0) cout << "Maximum neutron wall load (NWL\_max) at first wall is out of the Brayton cycle tabulated interval " << nwl2 << "\n"; }
- if (q2 > 0.375 && q2 <= 0.500) { eta = -0.00361\*nwl2 + 0.56934;
- if (nwl2 < 1.8 && nwl2 > 6.0) cout << "Maximum neutron wall load (NWL\_max) at first wall is out of the Brayton cycle tabulated interval " << nwl2 << "\n"; }
- if (q2 > 0.500 && q2 <= 0.625) { eta = -0.00362\*nwl2 + 0.54828;
- if (nwl2 < 2.4 && nwl2 > 6.0) cout << "Maximum neutron wall load (NWL\_max) at first wall is out of the Brayton cycle tabulated interval " << nwl2 << "\n"; }
- if (q2 > 0.625 && q2 <= 0.750) { eta = -0.00363\*nwl2 + 0.52605;
- if (nwl2 < 3.0 && nwl2 > 6.0) cout << "Maximum neutron wall load (NWL\_max) at first wall is out of the Brayton cycle tabulated interval " << nwl2 << "\n"; }
- if (q2 > 0.750) { eta = 0; cout << "Maximum surface heat flux q" (" << q2 << ") is too high for the Brayton cycle tabulated interval.\n"; }

## 8. Current Drive

**Reference:** R. Raffray, "Power Flow.doc" 5/2008, implemented into code by Z. Dragojlovic; See also L. Waganer "Cost Basic Documentation (11-25-10).doc" for updated costing to 2009\$.

**Code Modules:** DesignPoint.cpp

**Notes:** The current drive parameters are called by reading part of the powerflow.data input file. This defines parameters such as efficiencies, frequencies, and unit costs of potential CD units such as lower hybrid (LH), neutral beam (NB), and fast wave (FW). The CD efficiencies are interpolated if the value is not directly found in the table. The frequency is set in the input file as well.

### CD Frequency, Efficiency, Cost Table

The following table gives the efficiency and cost at the frequency for an ICRF Fast Wave current drive design.

- frequency[6] = {80.0e6, 158.0e6, 250.0e6, 800.0e6, 2500.0e6, 8000.0e6};
- efficiency[6] = {0.84, 0.72, 0.65, 0.63, 0.48, 0.44 };
- ~~unit\_cost[6] = {1.15, 1.15, 2.30, 2.30, 2.30, 2.87 };~~ // 1992\$
- unit\_cost[6] = {1.65, 1.65, 3.29, 3.29, 3.29, 3.29 };

### Generic current drive efficiencies

- eta\_cd\_generic = generic current drive source efficiency
- eta\_cd[nb] = generic NB current drive source efficiency
- eta\_cd[lh] = generic LH current drive source efficiency
- eta\_cd[icrf] = generic ICRF current drive source efficiency
- eta\_cd[ec] = generic EC current drive source efficiency

## Current Drive Power

- $P_{cd\_generic} = inputdata[npt][25]*1.0e6; // \text{from physics input file}$
- $P_{cd\_loss} = (1.0 - \eta_{cd\_generic}) * P_{cd\_generic} / \eta_{cd\_generic};$

## 9. Parts

**Reference:** *Existing code*

**Code Modules:** *Geometry.cpp*

**Notes:** The following sections contain information on how to build the various parts of the tokamak power plant. Most dimensions are read into the code from the builds\_generic.data input files. More information about the inputs and outputs can be found in **ASC Inputs & Outputs.pdf** on the Documentation page. More information about what is contained in the Geometry.cpp module can be found in the document **ASC Geometry Module.pdf**.

## A. Plasma Surface Area

**Reference:** *C. Kessel 5/2011 "Plasma Contour2 – Kessel plasma geometry discretized formulas.pdf"*

**Code Modules:** *Geometry.cpp* "1/2 Plasma Geometry – at plasma surface" & "2/2 Plasma Geometry – with SOL attached."

**Notes:** Calculates the surface area of the plasma. The average neutron wall flux uses this by dividing the neutron power by the plasma surface area.

### A. Elliptical equations:

Good for estimation but not accurate when plasma elongation, squareness, and triangularity push the plasma shape beyond an ellipse.

- $A_{plasma\_surface} = 4.0 * \pi * \pi * R * (R / A + SOL) * \sqrt{(1.0 + \text{pow}(k, 2.0)) / 2.0};$ 
  - SOL = SOL thickness from input file
  - A = aspect ratio from physics module output
  - k = kappa, plasma elongation
  - R = plasma major radius

### B. Perimeter method:

Uses the perimeter function in Part.cpp to find the plasma's perimeter and then revolve it about  $2\pi R$ .

*[from ASC Part.cpp module]*

- $A_{plasma\_surface} =$   
 $(*Plasma).surface\_area("plasma\_perimeter") * 2.0 * \pi * plasma\_major\_R;$

### C. Discretized equations:

These take into account all the plasma factors that affect its size. This is the most accurate formula and is currently used in the code. To take advantage of symmetry, the contour shape is multiplied by two.

- $R = R_o + a \cos(\vartheta + \sin^{-1}(\delta) \sin(\vartheta))$
- $Z = a \kappa \sin(\vartheta + \zeta_o \sin(2\vartheta))$  for  $\vartheta < \pi/2$
- $Z = a \kappa \sin(\vartheta + \zeta_i \sin(2\vartheta))$  for  $\vartheta \geq \pi/2$ 
  - $R_o$  = plasma major radius
  - a = plasma minor radius (add SOL thickness here to find surface area incl. SOL)
  - $\kappa$  = plasma elongation

- $\delta$  = plasma triangularity
- $\zeta_o$  = plasma squareness on outboard side ( $\sim 0$ )
- $\zeta_i$  = plasma squareness on inboard side ( $\sim 0$ )
- $\vartheta$  = poloidal angle

## B. Last Closed Magnet Surface (Sepatrix)

**Reference:** Z. Dragojlovic, C. Kessel 2/2007 “X-Point.pdf”

**Code Modules:** *Geometry.cpp* “Last Closed Magnetic Surface”

**Notes:** Also known as “LCMSurf” or “Sepatrix.” The last closed magnetic surface follows the contour of the plasma shape offset by the thickness of the SOL. It separates from the plasma at the top and bottom and crosses in the middle of the divertor region.

**Reference Part:** plasma boundary

### How to Build

*The departure angles from following the plasma,  $\alpha_{in}$  and  $\alpha_{out}$ , are taken from CAD drawings of ARIES-AT.*

- $\alpha_{in} = 66.25 \cdot \pi / 180.0; \quad // = 1.156 \text{ rad}$
- $\alpha_{out} = \pi - 19.44 \cdot \pi / 180; \quad // = 2.802 \text{ rad}$
- $\delta = (*Machine).inboard\_generic\_build(0)/2.0; \quad // \text{ distance between plasma surface and separatrix} = 1/2 * SOL$

Build the separatrix from contour lines by following the plasma boundary. Space the separatrix at  $\frac{1}{2}$  the scrape-off-layer thickness away from the plasma (“delta”). Follow the plasma boundary up until the vector of the separatrix is equal to the above constants,  $\alpha_{in}$  and  $\alpha_{out}$ , which mimic the angles from ARIES-AT. Terminate the lines when then hit the replaceable HT shield. See Figure 1 for more details.

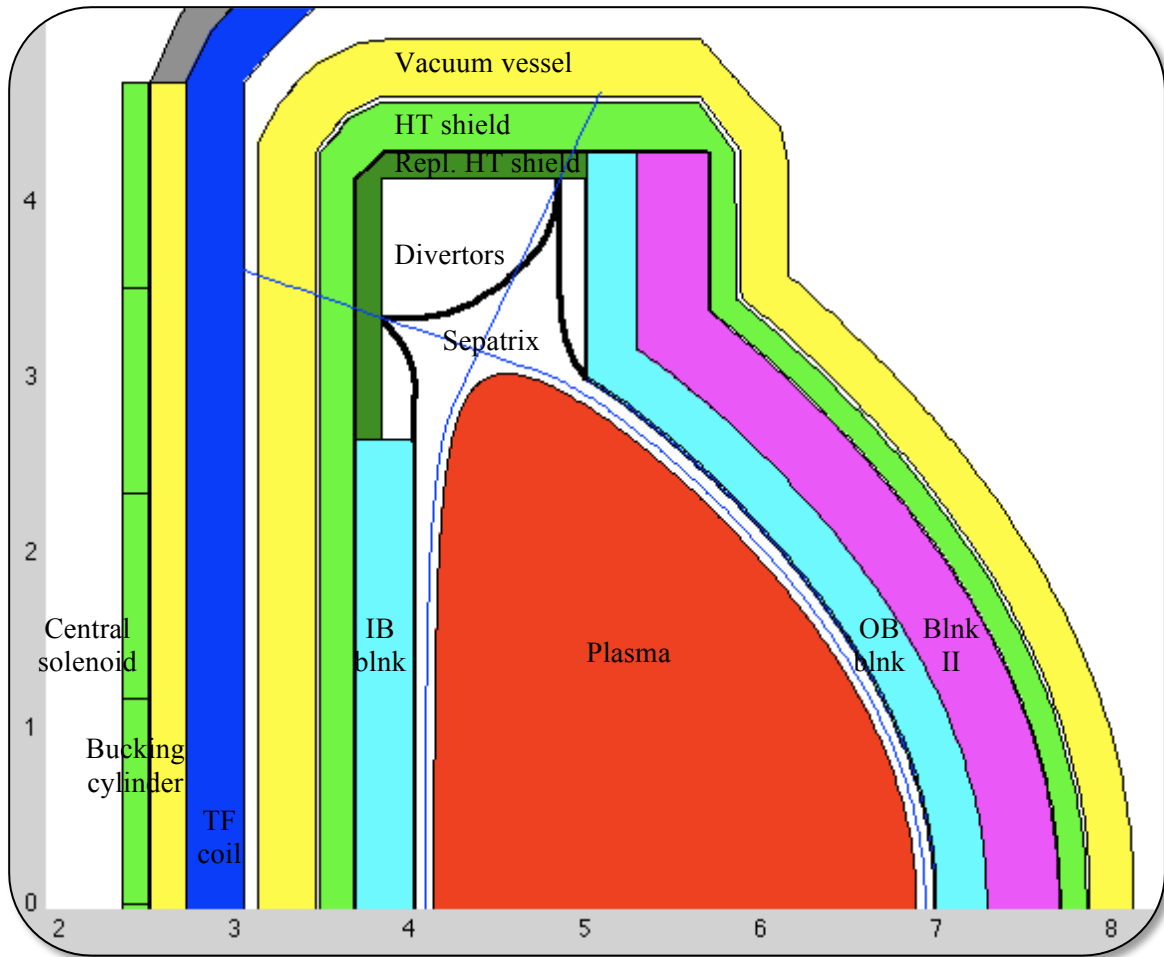


Figure 1. SCLL radial build with part labels (12/2011)

## C. First Wall IB Geometry

**Reference:** Z. Dragojlovic, 3/2007 "First Wall.pdf"

**Code Modules:** Geometry.cpp "First Wall IB Geometry"

**Notes:** The inboard plasma-facing first wall component that resides in front of the IB blanket and protects it. The majority of the cooling takes place in the first wall.

**Reference Part:** plasma boundary

### How to Build

- `dscr = (*Machine).inboard_generic_build(0); // SOL thickness`
- `dwall = (*Machine).inboard_generic_build(1); // FW thickness`
- `scale = 0.877; // scaling factor for FW height from ARIES-AT, constant in code`

Build the IB first wall at thickness "dwall" from input file and height scaled from ARIES-AT design,  $0.877 \times \text{plasma height}$ . Place the IB FW a distance equal to the SOL from the plasma.

## D. First Wall OB Geometry

**Reference:** Z. Dragojlovic, 3/2007 “First\_Wall.pdf”

**Code Modules:** Geometry.cpp “First Wall OB Geometry”

**Notes:** The outboard plasma-facing first wall component that resides in front of the OB blanket and protects it. The majority of the cooling takes place in the first wall.

**Reference Part:** plasma boundary

### How to Build

- `dscr = (*Machine).inboard_generic_build(0);` // SOL thickness
- `dwall = (*Machine).inboard_generic_build(1);` // FW thickness
- `scale = 0.34833;` // constant from First Wall.doc to scale geometrically to AT (ref?)
- `scale2 = 1.2577;` // constant from First Wall.doc to scale geometrically to AT (reference? The First\_Wall.pdf has this number in the document with an explanation that it is used to scale to the AT design, but how the specific number was attained.)

Build the OB first wall at thickness “dwall,” which is taken from input file. Follow the plasma upward on the OB side a distance equal to the SOL from the plasma and end at the height of the plasma reference. From here, go vertical upward from the height of the plasma + 1.2577, which is a scaling factor from the ARIES-AT design. This allows room for the divertors on the top and bottom of the double-null plasma.

## E. Blanket IB Geometry

**Reference:** Z. Dragojlovic, 3/2007 “Blanket.pdf”

**Code Modules:** Geometry.cpp “Blanket IB Geometry”

**Notes:** Extends nearly the length of the plasma behind the first wall and provides bulk cooling and neutron shielding with either PbLi and/or helium.

**Reference Part:** inboard first wall

### How to Build

- `gap = 0.0;` // gap between the FW and blanket
- `dwall = (*Machine).inboard_generic_build(2);` // blanket thickness from radial build

Build the IB blanket at thickness “dwall” by referencing the IB first wall. Extend it to the same height as the IB first wall.

## F. Blanket OB Geometry

**Reference:** Z. Dragojlovic, 3/2007 “Blanket.pdf”

**Code Modules:** Geometry.cpp “Blanket OB Geometry”

**Notes:** Extends around the outboard portion of the plasma and provides bulk cooling and neutron shielding with either PbLi and/or helium.

**Reference Part:** outboard first wall

### How to Build

- `dfw = (*Machine).outboard_generic_build(1); // FW I thickness`
- `gap = 0.0; // FW I to blanket gap`
- `dwall = (*Machine).outboard_generic_build(2); // OB blanket I thickness`

Build the OB blanket at thickness “dwall” by referencing the OB first wall. Extend it to the same height as the OB first wall.

## G. Back Wall IB Geometry (DCLL)

**Reference:** *No specific documentation*

**Code Modules:** *Geometry.cpp* “Back Wall IB Geometry (DCLL)”

**Notes:** Essentially a thinner “IB First Wall” behind the IB blanket that follows the same contour but with a different composition.

**Reference Part:** inboard blanket

### How to Build

- `gap = (*Machine).inboard_generic_build(5); // blanket and back wall gap`
- `dwall = (*Machine).inboard_generic_build(6); // back wall thickness`

Build the IB back wall at thickness “dwall” behind the IB blanket by referencing the IB blanket. Extend it to the same height as the IB blanket/first wall.

## H. Back Wall OB Geometry (DCLL)

**Reference:** *No specific documentation*

**Code Modules:** *Geometry.cpp* “Back Wall OB Geometry (DCLL)”

**Notes:** Essentially a thinner “OB First Wall” behind the OB blanket that follows the same contour but with a different composition.

**Reference Part:** outboard blanket

### How to Build

- `dblanket = (*Machine).outboard_generic_build(2); // blanket I thickness`
- `gap = 0.0; // blanket I to back wall II gap`
- `dwall = (*Machine).outboard_generic_build(9); // back wall II thickness`

Build the OB back wall at thickness “dwall” behind the OB blanket of thickness “dblanket” by referencing the IB blanket. Extend it to the same height as the OB blanket/first wall.

## I. Blanket II Geometry (SCLL)

**Reference:** *No specific documentation*

**Code Modules:** *Geometry.cpp* “Blanket OB Geometry”

**Notes:** Second blanket behind the main OB blanket in the SCLL design. Provides additional bulk cooling and neutron shielding with either PbLi and/or helium. Not used in DCLL design.

**Reference Part:** outboard blanket

### **How to Build**

- `dwref = (*Machine).outboard_generic_build(2); // OB blanket thickness`
- `dwgap = (*Machine).outboard_generic_build(6); // OB blanket and blanket II gap (stabilizing shell?)`
- `dwall = (*Machine).outboard_generic_build(8); // blanket II thickness`

Build the OB blanket II at thickness “dwall” by referencing the OB blanket. Extend it to the same height as the OB blanket. Blanket II has a different composition than blanket I.

## **J. Divertor Plates (SCLL)**

**Reference:** Z. Dragojlovic, 3/2007 “Divertor\_Plates.pdf”

**Code Modules:** *Geometry.cpp* “Divertor Plates (SCLL)”

**Notes:** This part design builds the six divertor plates in the upper and lower divertor region. Note that the reference document uses scaling factors to scale the divertors to the ARIES-AT design, whereas the code has implemented the scaling of normalized vectors instead.

**Reference Parts:**

- LCMSurf = Sepatrix
- RefLeft = Inboard blanket
- RefRight = Outboard blanket

### **How to Build**

- `delta = (*Machine).inboard_generic_build(0)/10.0; // 1/10th SOL`
- `dwall = (*Machine).inboard_generic_build(2); // IB blanket thickness`
- `drephs = (*Machine).vertical_generic_build(3); // repl HT shield thickness`

*Implement the curvature identical to the ARIES-AT divertor plates by stretching the normalized vectors which sample this curvature between the calculated end-points of the divertor plates generated by this code.*

**Normalized sample vectors: (same as DCLL below; no reference)**

- `xnorm0[10] = {0.0, 0.01, 0.01, 0.03, -0.02, -0.17, -0.35, -0.56, -0.76, -1.0};`
- `ynorm0[10] = {0.0, 0.17, 0.31, 0.45, 0.57, 0.69, 0.79, 0.86, 0.93, 1.0};`
- `xnorm1[10] = {0.0, 0.09, 0.24, 0.37, 0.51, 0.67, 0.81, 0.92, 0.98, 1.0};`
- `ynorm1[10] = {0.0, 0.00, 0.00, 0.03, 0.10, 0.23, 0.39, 0.56, 0.75, 1.0};`
- `xnorm2[10] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.01, 0.19, 0.49, 1.0};`
- `ynorm2[10] = {0.0, -0.09, -0.18, -0.31, -0.43, -0.55, -0.66, -0.79, -0.90, -1.0};`

Build the top three divertor plates as follows. The bottom plates will mirror the top ones.

- # 1. **IB:** Start on top-right corner of the IB blanket and scale the normalized sample IB vector “x,ynorm0” until it strikes the replaceable HT shield. See Figure 2.

- # 2. **Dome:** Start from end point of the IB divertor (strike point of divertor against the repl. HT shield) and scale the normalized sample dome vector “x,ynorm1” until it strikes the upper repl. HT shield.
- # 3. **OB:** Start from the end point of the dome divertor where it strikes the upper repl. HT shield and scale the normalized sample OB vector “x,ynorm2” until it touches the top-left corner of the OB first wall.
- To construct the lower divertor plates, simply mirror the top ones down across the  $y = 0$  line of symmetry.

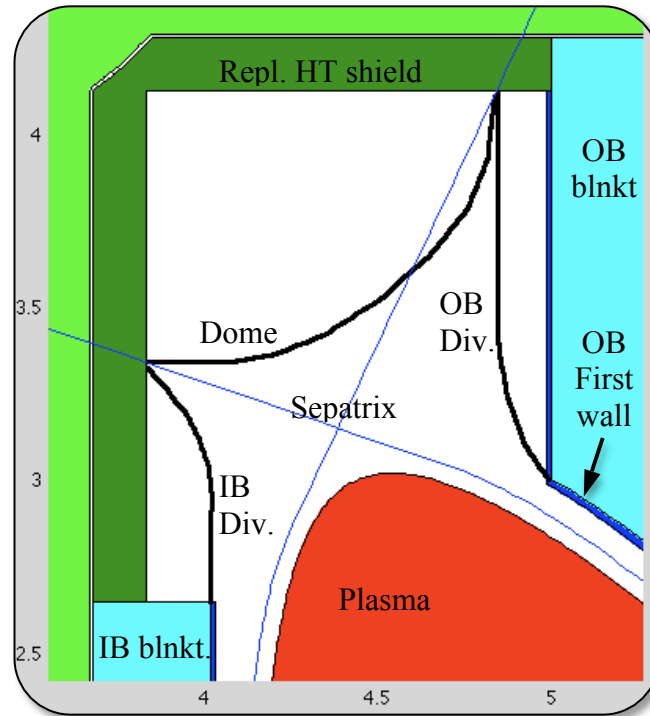


Figure 2. Upper three divertor plates scaled from ARIES-AT. IB (left), dome (top), OB (right) (12/2011)

## K. Divertor Plates (DCLL)

**Reference:** Z. Dragojlovic, 3/2007 “Divertor\_Plates.pdf”

**Code Modules:** *Geometry.cpp* “Divertor Plates (DCLL) - slightly different than SCLL, especially IB side”

**Notes:** This part design builds the six divertor plates in the upper and lower divertor region. Note that the reference document uses scaling factors to scale the divertors to the ARIES-AT design, whereas the code has implemented the scaling of normalized vectors instead.

### Reference Parts:

- LCMSurf = Sepatrix
- RefLeft = Inboard back wall
- RefRight = Outboard first wall

### How to Build

- delta = (\*Machine).inboard\_generic\_build(0)/10.0; // 1/10th SOL
- dfw = (\*Machine).inboard\_generic\_build(1); // FW thickness
- dgfwb = 0.0; // FW to blanket gap

- dblkt = (\*Machine).inboard\_generic\_build(2); // blanket thickness
- dgbbw = (\*Machine).inboard\_generic\_build(5); // blanket to back wall gap thickness
- dbw = (\*Machine).inboard\_generic\_build(6); // back wall thickness
- drephs = (\*Machine).vertical\_generic\_build(3); // repl HT shield thickness

*Implement the curvature identical to the ARIES-AT divertor plates by stretching the normalized vectors which sample this curvature between the calculated end-points of the divertor plates generated by this code.*

**Normalized sample vectors: (same as SCLL above; no reference – supposedly to mimic AT)**

- xnorm0[10] = {0.0, 0.01, 0.01, 0.03, -0.02, -0.17, -0.35, -0.56, -0.76, -1.0};
- ynorm0[10] = {0.0, 0.17, 0.31, 0.45, 0.57, 0.69, 0.79, 0.86, 0.93, 1.0};
- xnorm1[10] = {0.0, 0.09, 0.24, 0.37, 0.51, 0.67, 0.81, 0.92, 0.98, 1.0};
- ynorm1[10] = {0.0, 0.00, 0.00, 0.03, 0.10, 0.23, 0.39, 0.56, 0.75, 1.0};
- xnorm2[10] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.01, 0.19, 0.49, 1.0};
- ynorm2[10] = {0.0, -0.09, -0.18, -0.31, -0.43, -0.55, -0.66, -0.79, -0.90, -1.0};

Build the top three divertor plates as follows. The bottom plates will mirror the top ones.

- # 1. **IB:** Start on top-right corner of the IB blanket and scale the normalized sample IB vector “x,ynorm0” until it strikes the replaceable HT shield. The top-left corner is found by ksihiLeft (RefLeft = IB back wall) + dgbbw + dblkt + dgfbw + dfw. See also Figure 2.
- # 2. **Dome:** Start from end point of the IB divertor (strike point of divertor against the repl. HT shield) and scale the normalized sample dome vector “x,ynorm1” until it strikes the upper repl. HT shield.
- # 3. **OB:** Start from the end point of the dome divertor where it strikes the upper repl. HT shield and scale the normalized sample OB vector “x,ynorm2” until it touches the top-left corner of the OB first wall.  
To construct the lower divertor plates, simply mirror the top ones down across the y = 0 line of symmetry.

## L. Replaceable HT Shield

**Reference:** *No specific reference*

**Code Modules:** *Geometry.cpp* “Replaceable HT Shield, 2 parts: top and bottom”

**Notes:** A high-temperature shield that resides behind the upper and lower divertor regions and is replaceable. Cooled by PbLi or He.

**Reference Parts:**

- RefLo = IB back wall (DCLL), IB blanket (SCLL)
- RefHi = OB first wall (DCLL), OB blanket (SCLL)
- Plasma = plasma contours (both blankets)

### How to Build

- dwall = (\*Machine).inboard\_generic\_build(2); // IB blanket thickness
- dhtrepvrt = (\*Machine).vertical\_generic\_build(3); // repl HT shield thickness

Build the replaceable HT shield by starting at the top of the “RefLo” part and build upwards until the top of the “RefHi” is reached. Add a 45-degree angle to the rear of the repl. HT shield, then turn toward the outboard side and build until the “RefHi” is reached. Make the thickness per the input file. See Figure 2 for more clarification.

## M. HT Shield

### HT Shield Scaling

**Reference:** *L. El-Guebaly guidance 3/2011*

**Code Modules:** *Aries.cpp* “Scale HT Shield Thickness”

**Notes:** Scales the HT Shields with a factor dependent on the average neutron wall loading. This affects the inboard HT Shield, top/bottom replaceable HT Shields, and Skeleton Rings thicknesses. Note: in C++ coding, "log" is actually the natural log (ln base e). "Log base 10 is "log10()"

- DCLL blanket scaling factor =  $7.5e-2 * \log(nwl/3.3)$ ;
- SCLL blanket scaling factor =  $6.7e-2 * \log(nwl/3.3)$ ;
  - nwl = average neutron wall loading, read from physics module

### How to Build HT Shield

**Reference:** *Z. Dragojlovic, 3/2007 “HT\_Shield.pdf”*

**Code Modules:** *Geometry.cpp* “HT Shield”

**Notes:** - **SCLL** splits into IB and OB but builds it continuously. The HT shield resides behind the IB blanket and behind the OB blanket II and wraps around the entire vessel.  
 - **DCLL** maintains one continuous HT Shield only on IB side. The HT shield resides only behind the IB blanket and above/below the replaceable HT shield on the top/bottom of the vessel.

#### Reference Parts:

##### *SCLL reference parts for HT Shield:*

- RefLeft = BlanketInboard
- RefDiag[2] = RepHTShield, [0] top, [1] bottom
- RefTop = BlanketOutboard
- RefRight = BlanketII

##### *DCLL reference parts for HT Shield:*

- RefLeft = BackWallInboard
- RefDiag[2] = RepHTShield, [0] top, [1] bottom
- RefTop = BlanketOutboard
- RefRight = BackWallOutboard

- gapinb = (\*Machine).inboard\_generic\_build(7); // IB blanket and HT shield gap
- dhtinb = (\*Machine).inboard\_generic\_build(8); // IB HT shield thickness, scaled down slightly in Aries.cpp
- dbwoutb = (\*Machine).outboard\_generic\_build(8); // OB blanket II thickness
- dbwoutb = (\*Machine).outboard\_generic\_build(9); // OB back wall II thickness (DCLL)
- gapoutb = (\*Machine).outboard\_generic\_build(12); // OB blanket II to HT shield gap
- dhtoutb = (\*Machine).outboard\_generic\_build(13); // OB HT shield thickness, scaled down slightly in Aries.cpp
- dhtrepvert = (\*Machine).vertical\_generic\_build(3); // repl HT shield thickness
- gapvert = (\*Machine).vertical\_generic\_build(4); // repl HT shield and HT shield gap
- dhtvert = (\*Machine).vertical\_generic\_build(5); // vertical HT shield thickness

Build the HT shield from contour lines by following the referenced parts (RefLeft, Diag, Top, Right) at the thickness specified in the generic\_build.data input file. The HT shield generally

hugs the IB blanket, the replaceable HT shield, and the OB blanket II or back wall. There is a 45-degree notch at the top-left and top-right as the shield conforms to the replaceable HT shield on the inside and the vacuum vessel on the outside. The volume fractions of the different materials composing the part are specified in Cost.cpp

## N. Skeleton Ring (DCLL)

**Reference:** *No specific reference*

**Code Modules:** *Geometry.cpp* “Skeleton Ring (DCLL)”

**Notes:** This assumes the HT shield (“HTShieldDCLL”) is in one piece. The skeleton ring is essentially another shield for the DCLL design that wraps around the vessel behind the HT shield on the IB and vertical side and behind the OB back wall of the blanket.

**Reference Part:** DCLL HT shield

### How to Build

- dhtinb = (\*Machine).inboard\_generic\_build(8); // IB HT shield thickness
- gapinb = (\*Machine).inboard\_generic\_build(9); // IB HT shield and SkelRing gap
- dvinb = (\*Machine).inboard\_generic\_build(10); // IB SkelRing thickness
- dvoutb = (\*Machine).outboard\_generic\_build(15); // OB SkelRing thickness
- drephvert = (\*Machine).vertical\_generic\_build(3); // vert repl HT shield thickness
- dhtvert = (\*Machine).vertical\_generic\_build(5); // vert HT shield thickness
- gapvert = (\*Machine).vertical\_generic\_build(6); // vert HT shield and SkelRing gap
- dvvert = (\*Machine).vertical\_generic\_build(7); // vert SkelRing thickness

Build the skeleton ring by adding contour lines to follow the IB HT shield all the way around until the vertical. There is a notch in the top-left corner for the 45-degree angle in the HT shield. Then the skeleton ring follows the OB back wall as it continues around. The thickness of the ring is different for the IB/OB and the vertical dimensions.

## O. Manifold (DCLL OB and vert)

**Reference:** *No specific reference*

**Code Modules:** *Geometry.cpp* “Manifold (DCLL)”

**Notes:** From L. El-Guebaly’s new (July 2011) design, contains PbLi+He. Some space is needed for PbLi/He manifolds on the OB side just behind the OB and vertical skeleton ring on the DCLL blanket design. There are no manifolds on the IB side.

**Reference Part:** skeleton ring

### How to Build

TBD

## P. Manifolding (SCLL)

**Reference:** *New 2/2012 by L. Carlson, seen in ACT-SA (d)*

**Code Version:** ASC v.35

**Code Modules:** *Geometry.cpp* “Manifolding (SCLL)”

**Notes:** Requested by M. Tillack ~2/2012 to remove OB blanket FW, I & II extensions on the top and bottom and to add a manifolding region in preparation for future detailed design work in this area. This region is to the right of OB divertors and above/below OB blankets and will be used to combine and route He & PbLi into and out of the blankets/vessel. It will be composed of ~30% FS, ~50% PbLi, ~10% He and ~10% SiC-comp.

**Reference Part:** Replaceable HT shield [0], FW, blanket I, blanket II

### **How to Build**

- `dwall = (*Machine).inboard_generic_build(1); // FW thickness`

Build the manifolding area by reducing the OB FW and blanket I & II so that they stop and angle up toward the vacuum vessel at the inflection point of the FW as it turns upward. Fill this void between the replaceable HT shield to the OB HT shield (x) and the HT shield to the top of the blanket I & II with the manifolding.

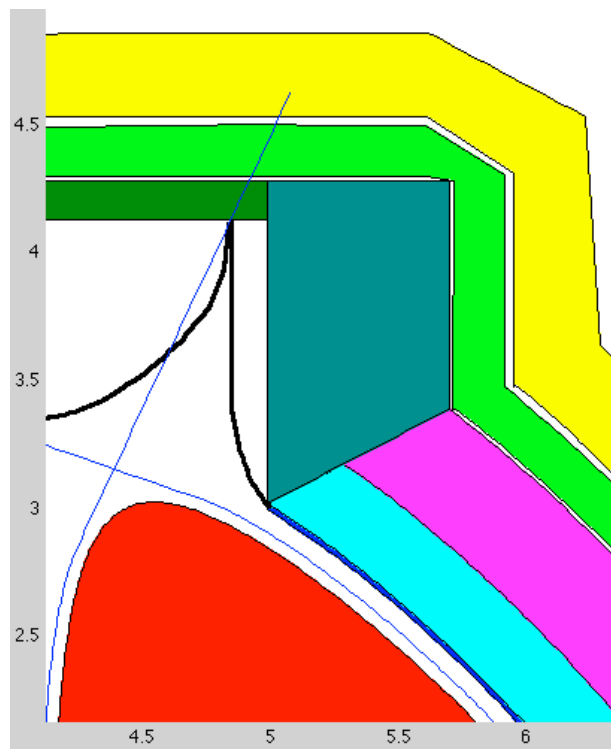


**Figure 3. Manifolding region above OB blanket I & II. (2/8/2012)**

## **Q. Vacuum Vessel (SCLL)**

**Reference:** Z. Dragojlovic, 3/2007 “*Vacuum\_Vessel.pdf*”

**Code Modules:** *Geometry.cpp* “Vacuum Vessel (SCLL)”

**Notes:** Treated as a single component in the SCLL blanket design and built around the entire HT shield. The vacuum vessel realistically has penetrations for maintenance ports and vacuum ducts. These are constructed in their own builds.

**Reference Part:** HT shield

### **How to Build**

- `dwgap = (*Machine).inboard_generic_build(13); // IB HT shield and VV gap`
- `dwall = (*Machine).inboard_generic_build(14); // IB VV thickness`
- `ibvv = (*Machine).inboard_generic_build(14); // IB VV thickness`
- `obvvgap = (*Machine).outboard_generic_build(18); // OB HT shield and VV gap`
- `obvv = (*Machine).outboard_generic_build(19); // OB VV thickness`

Build the vacuum vessel by following the HT shield all the way around. Mimic the top-left and top-right corners, which are notched. The IB/vertical builds are of different thickness than the OB build, as specified in the generic build input file. See Figure 1 for more details.

## **R. Vacuum Vessel (DCLL)**

**Reference:** Z. Dragojlovic, 3/2007 “*Vacuum\_Vessel.pdf*” No specific documentation as to why there are 4 components of the DCLL vacuum vessel.

**Code Modules:** *Geometry.cpp* “1/4, 2/4, 3/4, 4/4 Vacuum Vessel (DCLL)”

**Notes:** Treated as four separate components in the DCLL blanket design and built around the entire skeleton ring. The vacuum vessel realistically has penetrations for maintenance ports and vacuum ducts. These are constructed in their own builds.

**Reference Part:** skeleton ring

### **How to Build VV 1 of 4 Inboard**

- `gapinb = (*Machine).inboard_generic_build(13); // IB skel ring and VV gap`
- `dringvert = (*Machine).vertical_generic_build(7); // vert skel ring thickness`
- `dvinb = (*Machine).inboard_generic_build(14); // IB VV thickness`

Build the inboard portion of the vacuum vessel by adhering to the IB skeleton ring. Follow the ring all the way to the top until the bottom notch of the ring. See Figure 4 for more details. The thickness of the IB VV is specified in the radial build input file as well as any gaps.

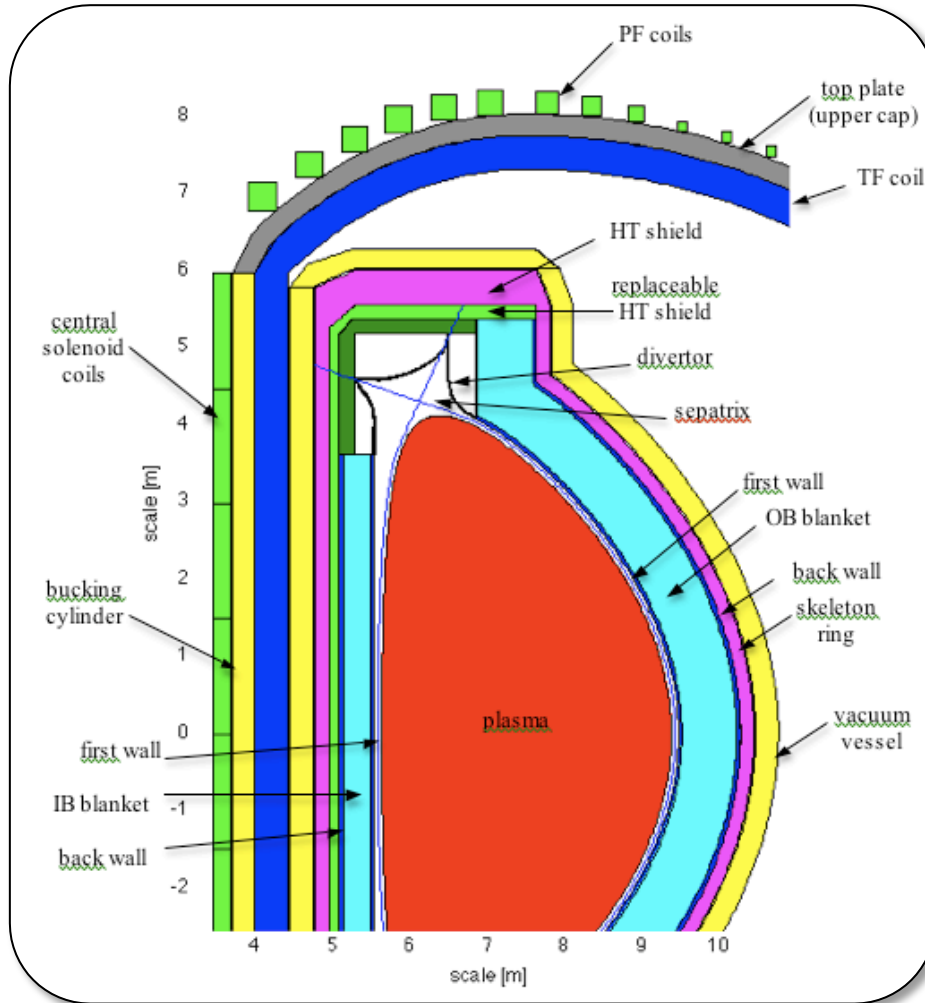


Figure 4. DCLL radial build with labels (12/2011)

### **How to Build VV 2 of 4 Outboard**

- gapoutb = (\*Machine).outboard\_generic\_build(18); // OB skel ring and VV gap
- dringvert = (\*Machine).vertical\_generic\_build(7); // vert skel ring thickness
- gapvert = (\*Machine).vertical\_generic\_build(10); // vert skel ring and VV gap
- dvoutb = (\*Machine).outboard\_generic\_build(19); // OB vacuum vessel thickness
- dvvert = (\*Machine).vertical\_generic\_build(11); // vert VV thickness

Build the outboard portion of the vacuum vessel by adhering to the OB skeleton ring. Follow the ring all the way to the top until the top notch of the ring. This will allow the OB VV to be removed for maintenance. See Figure 4 for more details. The thickness of the OB VV is specified in the radial build input file as well as any gaps.

### **How to Build VV 3 of 4 Bottom, by symmetry**

Build the bottom vacuum vessel by simply mirroring the top by symmetry.

### **How to Build VV 4 of 4 Top**

- dhtinb = (\*Machine).inboard\_generic\_build(8); // IB HT shield thickness
- dringinb = (\*Machine).inboard\_generic\_build(10); // IB skel ring thickness
- gapinb = (\*Machine).inboard\_generic\_build(13); // IB skel ring and VV gap
- dringvert = (\*Machine).vertical\_generic\_build(7); // vert skel ring thickness
- gapvert = (\*Machine).vertical\_generic\_build(10); // vert skel ring and VV gap
- dvinb = (\*Machine).inboard\_generic\_build(14); // IB VV thickness
- dvoutb = (\*Machine).outboard\_generic\_build(19); // OB vacuum vessel thickness
- dhtrepvert = (\*Machine).vertical\_generic\_build(3); // vert repl HT shield thickness
- dsrvert = (\*Machine).vertical\_generic\_build(7); // vert skel ring thickness
- dvvert = (\*Machine).vertical\_generic\_build(11); // vert VV thickness

Build the top portion of the vacuum vessel by adhering to the top of the skeleton ring. Follow the ring from the top of the IB VV to the top of the OB VV. The vacuum vessel components should fit together cleanly. See Figure 4 for more details. The thickness of the top VV is specified in the radial build input file as well as any gaps.

## S. TF Magnet

**Reference:** Z. Dragojlovic, L. Bromberg, 5/2007 “TF\_Magnet.pdf” and  
Z. Dragojlovic, 8/2010 “HowToBuildTFCoil.pdf”

**Code Modules:** Geometry.cpp “TF Magnet: r-theta plane, ‘Next Step’, conservative costing”

**Notes:** The seven TF coil designs are all similar to each other. The design in use currently is the third (iii) one below. The other algorithms won’t be detailed in this document.

- a. *r-theta plane*
  - i. PPPL Version, conservative Costing
  - ii. PPPL Version
  - iii. Next Step, conservative costing (currently in use)**
  - iv. Next Step
  - v. HTS
  - vi. Nb3Sn
  - vii. ARIES-AT coil design with fixed volume fractions
- b. *r-z plane – revolved*

**Reference Part:** plasma and vacuum vessel

### How to Build: TF Coil, r-theta plane – “Next Step, conservative costing”

- const pi = 4.0\*atan(1.0);
- const muo = 4.0\*pi\*1.0e-7;
- sigma\_sheath = *maximum stress in protective sheath around strands (defined from magnets.data input file)*
- sigmab = *maximum allowed stress in the coil casing (defined from magnets.data input file)*
- ksiloP = (\*Plasma).bounding\_box(0, "outer"); // Define the bounding box for the plasma
- ksihiP = (\*Plasma).bounding\_box(2, "outer"); // Define the bounding box for the plasma
- ksihiVV = (\*VVRefBody)[1].bounding\_box(2, "outer"); // Define the bounding box for VV
- ksiloVV = (\*VVRefBody)[0].bounding\_box(0, "outer"); // Define the bounding box for VV
- gapvvtf = (\*Machine).inboard\_generic\_build(17); // VV and IB TF coil gap
- rTFinner = ksiloVV - gapvvtf; // radius of inner TF coil

- **Maximum Magnetic Field:**
  - $R_o = (*Machine).plasma\_major\_radius();$
  - $B_{tmax} = B_t * R_o / r_{TFinner};$
  - $(*Machine).get\_maximum\_magnetic\_field\_at\_tf\_coil(B_{tmax});$  // puts  $B_{tmax}$  into this fcn
- **Current Density in Superconductor versus  $B_{tmax}$  function for Nb3Sn @ 4.2 K**
  - $b_{tmaxVec}[5] = \{6.0, 10.0, 14.0, 16.0, 18.0\};$
  - $j_{scVec}[5] = \{1.52e9, 1.35e9, 8.84e8, 5.43e8, 1.30e8\};$
  - [Super-conducting current table from "TF\_Magnet.pdf" The code interpolates to the nearest value in the table.]*
- Super-conducting current polynomial based on the table values (*this SUPERCEEDS the table above.*) *[C. Kessel also uses this in his code "sysengr.f" line ~364]*
  - $J_{sc} = -9.1226 * pow(B_{tmax}, 2) * 1.0e6 + 1.0301 * B_{tmax} * 1.0e8 + 1.2308 * 1.0e9;$
- **Cross section of the winding pack:**
  - $J_{cu} = \text{Current density in copper from magnets.data input file}$
  - $f_{he} = \text{Volume fraction of He coolant}$
  - $ds_{sheath} = pow(B_{tmax}, 2) / (2.0 * \mu_o * \sigma_{sheath});$  // Sheath thickness
  - $f_{sheath} = 1.0 - pow(1.0 - ds_{sheath}, 2);$  // Sheath fraction
  - $f_{ins} = \text{Volume fraction of the insulator}$
  - $I_{tf} = \text{Nominal current in TF coil}$
- **Cross-sectional areas:**
  - $A_{cu} = I_{tf} / J_{cu};$
  - $A_{sc} = I_{tf} / J_{sc};$
  - $A_{strands} = A_{cu} + A_{sc};$
  - $A_{turn} = A_{strands} / (1.0 - f_{he} - f_{sheath} - f_{ins});$  // C.K. sysengr.f line ~376
- **Cross-sectional area of a single turn in a winding pack**
  - $A_{ins} = A_{turn} * f_{ins};$
  - $A_{sheath} = A_{turn} * f_{sheath};$
- **Coil Casing Cross Section**
  - $N_{tf} = \text{Number of TF coils}$
  - $b = \text{Side thickness of the coil casing is } b/2$
  - $ATF = A_{turn} * B_t * 2.0 * \pi * R_o / (N_{tf} * I_{tf} * \mu_o);$  // Area of a TF coil single winding pack
  - $N_{wp} = \text{int}(ATF / A_{turn});$  // Number of turns per winding pack (WP)
  - $(*Machine).get\_number\_of\_tf\_cable\_turns(N_{wp});$  // puts  $N_{wp}$  into this function
- **Coil Casing Thickness Loop**
  - $a = b;$  //  $b = \text{side thickness of coil casing, initial guess for casing thicknesses}$
  - **for**(int j=0; j<5; j++) { // START LOOP
  - $R_1 = r_{TFinner} - 2.0 * a;$  // Calculate  $R_1$ , the inner radius of inner TF coil leg
  - **for**(int i=0; i<5; i++) {
  - $R_1 = r_{TFinner} - a - ATF / (2.0 * \pi * R_1 / N_{tf} - b);$  } // similar to CK sysengr.f~382

- $w_{TF} = 2.0 \cdot \pi \cdot R1 / N_{tf} - b$ ; *// width of the TF coil*
- $d_{TF} = A_{TF} / w_{TF}$ ; *// winding pack (conductors) thickness if lumped all together*
- Calculate R2 based on the clearance on the outboard side of the TF coils needed for removal of the power core module for maintenance purpose
- $c_{TFVV}$  = *Clearance between the Vacuum Vessel and the TF coil outmost edge*
- $R2 = R_{sihiVV} + R1 + c_{TFVV} / \sin(\pi / N_{tf})$ ; *// inner radius of outer TF coil leg*

#### Structure calculation based on bending:

- $G1 = \text{pow}(3.0 / (16.0 \cdot \sigma_{ab} \cdot \mu_0), 0.5)$ ;
- $G2 = B_t \cdot R_o / (R2 - R1)$ ;
- $G3 = \text{pow}((R2 \cdot R2 - R1 \cdot R1) \cdot (1 + \log(R2 / R1)) + R1 \cdot R2 \cdot (2 - (\log(R2 / R1)) \cdot (\log(R2 / R1))), 0.5)$ ; *// note extra log is for log^2, reference?*
- $\text{coeff} = 0.208$ ; *// OK for ACT-I, II. Coefficient is a casing thickness multiplier based on scaling FEA results to match AT casing.*
- $a = \text{coeff} \cdot G1 \cdot G2 \cdot G3$ ;
- *// "a" is the total coil casing, so a/2 + winding pack + a/2 = total TF thickness. This thickness includes coil case and/or all internal structure. There is a tradeoff between internal and coil casing structure – if one goes down, the other must go up.*
- } *// END LOOP*

#### Defining parameters of the coil cross section (the thickness is uniform along the TF coil perimeter for now)

- $\delta_{inner} = a + d_{TF}$ ; *// total thickness of inner leg equal to casing + magnet*
- $\delta_{outer} = a + d_{TF}$ ; *// total thickness of outer leg equal to casing + magnet*
- $\delta_{top} = a + d_{TF}$ ; *// total thickness of top/bottom part equal to casing + magnet*
- $R2 = R2 + \delta_{outer}$ ; *// Update the outer radius of the outer surface of the TF coil:*
- $(\text{*Machine}).\text{get\_tf\_coil\_radii}(R1, R2)$ ; *// Get the inner-most and outer-most radius of the TF Coil*
- $\delta = \max(\delta_{inner}, \max(\delta_{outer}, \delta_{top}))$ ;
- $\text{width} = w_{TF} + b$ ; *// width of TF coil*

#### Cross sectional area of the composition:

- $A_{comp} = \delta \cdot \text{width}$ ; *// cross sectional area of the composition*
- $A_{structure} = \delta \cdot \text{width} - d_{TF} \cdot w_{TF}$ ; *// steel (should be ~75% of WP+CC)*
- $v_{frac\_structure} = A_{structure} / A_{comp}$ ; *// volume fraction of SS316 structure*
- $v_{frac\_sc} = A_{sc} \cdot N_{wp} / A_{comp}$ ; *// superconductor (Nb3Sn)*
- $v_{frac\_cu} = A_{cu} \cdot N_{wp} / A_{comp}$ ; *// copper (Cu)*
- $v_{frac\_sheath} = A_{sheath} \cdot N_{wp} / A_{comp}$ ; *// sheath (Inconel)*
- $v_{frac\_ins} = A_{ins} \cdot N_{wp} / A_{comp}$ ; *// insulator (Polyamide)*
- $v_{frac\_he} = 1.0 - v_{frac\_structure} - v_{frac\_sc} - v_{frac\_cu} - v_{frac\_sheath} - v_{frac\_ins}$ ; *// He coolant*

#### Get TF Cross-Section Material Components:

- Gets the material costs, mass, etc from Part.cpp based on (1) material name, (2) volume fraction, (3) density fraction, and (4) the cost base.
  - (\*Magnet).get\_cross\_section(0, delta, width);
  - (\*Magnet).get\_material\_component("SS316", vfrac\_structure, 1.0, 0);
  - (\*Magnet).get\_material\_component("Nb3Sn", vfrac\_sc, 1.0, 1);
  - (\*Magnet).get\_material\_component("Cu", vfrac\_cu, 1.0, 2);
  - (\*Magnet).get\_material\_component("Inconel", vfrac\_sheath, 1.0, 3);
  - (\*Magnet).get\_material\_component("Polyamide", vfrac\_ins, 1.0, 4);
  - (\*Magnet).get\_material\_component("He", vfrac\_he, 1.0, 5);

### **How to Build: TF Coil, r-z plane – “revolved”**

This part design builds the TF coils by revolving their cross-sections in the r-z plane to create the coil’s shape about the “Princeton-D” ellipse.

- dw = (\*Magnet).cross\_section\_width(0); *// TF magnet wall thickness*
- hxp = (\*Machine).xpoint(1); *// X-point height*
- R1 = (\*Machine).tf\_magnet\_inner\_leg\_radius(); *// TF Coil inboard radii*
- R2 = (\*Machine).tf\_magnet\_outer\_leg\_radius(); *// TF Coil outboard radii*
- r1 = R1 + dw;
- r2 = R2 - dw;
- Ro = (\*Machine).plasma\_major\_radius(); *// Plasma major radius*
- ao = (\*Machine).plasma\_minor\_radius(); *// Plasma minor radius*

### **Define the parameters of the “Princeton-D” ellipse [1]:**

[1] Richard J. Thome, John M. Tarrh, “MHD and Fusion Magnets – Field and Force Design Concepts”, John Wiley & Sons, 1982.

- b = etahi + 1.043 ; *// radius on the vertical axis, same as cap below, reference?*
- a1 = (Ro-r1)/pow(1.0-pow( 0.95\*etahi/b, 2.0 ), 0.5 );
- a2 = r2 - Ro;

Build the TF coil by following the contour of the inboard vacuum vessel at the computed thickness. Once the top of the VV is cleared, the contour of the Princeton-D ellipse can be formed.

### **TF coil contour loop**

```
int len = 300; // Transform the plasma into the TF coil contour
double *x, *y;
x = new double [len];
y = new double [len];

x[0] = r1;
y[0] = 0.0;
double delx = 2*(r2-r1)/(len-3);
for(int i=1; i<len-1; i++) // loop over TF coil contour
{ if(i < (len+1)/2 ) // upper half of the coil
  { int c;
```

```

if(i == 1){c=0;}
else    {c=1;}
    x[i]=x[i-1]+c*delx;
if(x[i] < (r2-a2))           // inner ellipse
    { y[i] = b*pow(1.0-pow((x[i]-Ro)/a1, 2.0), 0.5); }
else                         // outer ellipse
    { y[i] = b*pow(1.0-pow((x[i]-Ro)/a2, 2.0), 0.5); } }
else                         // lower half of the coil
    { if(i == (len+1)/2)
      { x[i] = r2; } }
else
    { x[i] = x[len-i]; }
if(i == len-2) x[i] = x[0];
if(x[i] < (r2-a2))           // inner ellipse
    { y[i] = -b*pow(1.0-pow((x[i]-Ro)/a1, 2.0), 0.5); }
else                         // outer ellipse
    { y[i] = -b*pow(1.0-pow((x[i]-Ro)/a2, 2.0), 0.5); } }
} // end loop over TF coil contour

```

## T. Bucking Cylinder

**Reference:** *No specific documentation.*

**Code Modules:** *Geometry.cpp* “Bucking Cylinder”

**Notes:** This resides just around the central solenoid and gives it strength and support. Its thickness depends on the yield stress of the casing as well as the maximum toroidal field BTmax.

**Reference Part:** TF coil and vacuum vessel

### How to Build

- $\sigma_{Yield} = \text{casing yield stress for the coil case from magnets.data input file}$
- $\sigma = 2.0/3.0 * \sigma_{Yield};$
- $B_{tmax} = B_t * (*Machine).plasma\_major\_radius()/ksilo;$
- $dscr = (*Machine).inboard\_generic\_build(21); // TF coil to bucking cyl gap$
- $d_{wall} = \text{pow}(B_{tmax}, 2) * ksilo / (2.0 * \mu_0 * \sigma); // bucking cyl thickness$

Build the bucking cylinder at the computed thickness by referencing the inboard portion of the TF coil and following it up until the top of the vacuum vessel.

## U. Upper Toroidal Cap on the TF Coil

**Reference:** *No specific documentation.*

**Code Modules:** *Geometry.cpp* “Upper Toroidal Cap on the TF Coil”

**Notes:** This cap supports the PF coils that lie on top of the TF coils on the top of the machine. Its thickness depends on the yield stress of the casing and BTmax.

**Reference Part:** TF coil and vacuum vessel

### **How to Build**

- $\sigma_{Yield}$  = casing yield stress for the coil case from magnets.data input file
- $\sigma = 2.0/3.0 * \sigma_{Yield}$ ;
- $B_{tmax} = B_t * (*Machine).plasma\_major\_radius() / k_{silo}$ ;
- $dw_{gap} = (*Machine).vertical\_generic\_build(18)$ ; // TF coil to toroidal cap gap
- $d_{wall} = \text{pow}(B_{tmax}, 2) * k_{silo} / (2.0 * \mu_0 * \sigma)$ ; // toroidal cap thickness
- $d_{magnet} = (*RefBody).cross\_section\_width(0)$ ; // TF magnet thickness
  
- $R_o = (*Machine).plasma\_major\_radius()$ ;
- $R_1 = (*Machine).tf\_magnet\_inner\_leg\_radius()$ ;
- $R_2 = (*Machine).tf\_magnet\_outer\_leg\_radius()$ ;
- $r_1 = R_1 + d_{magnet}$ ;
- $r_2 = R_2 - d_{magnet}$ ;
- $b = \text{etahiV} + 1.043$ ; // TF ellipse radius on the vertical axis
- $a_2 = r_2 - R_o$ ; // TF outer ellipse radius on the horizontal axis

**Limiting point on the outer TF coil contour - this point is determined by the height of the vacuum vessel etahiV**

- $y_{tf} = \text{etahiV}$ ;
- $x_{tf} = R_o + a_2 * \sqrt{1.0 - \text{pow}(y_{tf}/b, 2.0)}$ ;
- $g = (x_{tf} - R_o) / a_2$ ;
- $\alpha_t = \text{atan}(-b/a_2 * g / \sqrt{1.0 - \text{pow}(g, 2.0)})$ ;
- $\alpha_n = \alpha_t + \pi/2.0$ ;

Build the upper toroidal cap at the computed thickness by starting at the top of the bucking cylinder and following the contour of the TF coil. End the cap when no more PF coils need support and before the top-most portion of the vacuum vessel is reached (cannot block maintenance door access).

## **V. Lower Toroidal Cap on the TF Coil**

**Reference:** No specific documentation.

**Code Modules:** Geometry.cpp “Lower Toroidal Cap on the TF Coil – By Design OR By Symmetry”

**Notes:** This cap supports the PF coils that lie below the TF coils on the bottom of the machine. Its thickness depends on the yield stress of the casing and  $B_{tmax}$ .

**Reference Part:** TF coil and vacuum vessel

### **How to Build**

- $\sigma_{Yield}$  = casing yield stress for the coil case from magnets.data input file
- $\sigma = 2.0/3.0 * \sigma_{Yield}$ ;
- $B_{tmax} = B_t * (*Machine).plasma\_major\_radius() / k_{silo}$ ;
- $dw_{gap} = (*Machine).vertical\_generic\_build(18)$ ; // TF coil to upper cap gap
- $d_{wall} = \text{pow}(B_{tmax}, 2) * k_{silo} / (2.0 * \mu_0 * \sigma)$ ; // toroidal cap thickness
- $d_{magnet} = (*RefBody).cross\_section\_width(0)$ ; // TF magnet thickness

- $Ro = (*Machine).plasma\_major\_radius();$
- $R1 = (*Machine).tf\_magnet\_inner\_leg\_radius();$
- $R2 = (*Machine).tf\_magnet\_outer\_leg\_radius();$
- $r1 = R1 + dmagnet;$
- $r2 = R2 - dmagnet;$
- $b = -etaloV + 1.043; // TF ellipse radius on the vertical axis$
- $a2 = r2 - Ro; // TF outer ellipse radius on the horizontal axis$

**Limiting point on the outer TF coil contour - this point is determined by the height of the vacuum vessel etahV**

- $y\_tf = etaloV;$
- $x\_tf = Ro + a2 * \sqrt{1.0 - \text{pow}(y\_tf/b, 2.0)};$
- $g = (x\_tf - Ro)/a2;$
- $\alpha\_t = \text{atan}(b/a2 * g / \sqrt{1.0 - \text{pow}(g, 2.0)}) ;$
- $\alpha\_n = \alpha\_t - \pi/2.0;$

Build the lower toroidal cap at the computed thickness by starting at the bottom of the bucking cylinder and following the contour of the TF coil. End the cap when no more PF coils need support and before the bottom-most portion of the vacuum vessel is reached (cannot block maintenance door access). Alternatively, the upper TF coil cap can be mirrored to the bottom by symmetry.

## W.PF Coil Design

**Reference:** Z. Dragojlovic, C. Kessel, 10/2007 "PF\_Magnet.pdf"

**Code Modules:** Geometry.cpp "PF Coil Design"

**Notes:** The PF coils run in the poloidal direction and lie on top of and bottom of the TF coil caps. PF coils #0-3 from top + #32-35 from bottom create the central solenoid; #4-17 lie along the top and #18-31 lie along the bottom (going clock-wise); #4-17 are duplicated as spares. See Figure 7 for an enumeration of the PF coils and their locations.

**Reference Part:** bucking cylinder, top and bottom TF coil caps

### How to Build

- **Coil currents at zero flux state:**
  - $ccoil1[18] = \{1.24, 0.47, 1.85, 7.08, 4.29, 4.05, 3.87, 4.20, 4.02, 3.64, 3.02, 2.27, 1.35, 0.15, -1.35, -3.11, -4.99, -6.83\};$
  - $ccoil2[18] = \{1.03, 1.76, 3.23, 5.85, 3.76, 3.78, 3.81, 4.36, 4.37, 4.14, 3.59, 2.86, 1.91, 0.57, -1.17, -3.32, -5.74, -8.15\};$
- **Rate of change of coil current with flux state:**
  - $dcdp[18] = \{1.40, 0.47, 0.54, 2.44, 0.87, 0.67, 0.52, 0.49, 0.31, 0.19, 0.10, 0.02, -0.03, -0.07, -0.06, 0.02, 0.14, 0.29\};$
- **Current limiters:**
  - $qmax = \text{upper limiter of the current}$

- $q_{min}$  = lower limiter of the current
- **Determine PF coil currents at given  $q_{95}$  and scale to plasma current:**
  - $q_{mhd} = (*Machine).plasma\_edge\_safety\_factor();$
  - $aip = (*Machine).plasma\_current();$
  - $for(int i=0; i<18; i++)$  // 0-17 are all PF coils above midplane
    - {  $if(q_{mhd} \geq q_{max})$
    - {  $ccoil[i] = ccoil2[i]*(aip/12.8);$  }
    - $if(q_{mhd} \leq q_{min})$
    - {  $ccoil[i] = ccoil1[i]*(aip/12.8);$  }
    - $if(q_{mhd} > q_{min} \ \&\& \ q_{mhd} < q_{max})$
    - {  $ccoil[i] = ccoil1[i] + ((ccoil2[i]-ccoil1[i])/(q_{max}-q_{min}))* (q_{max}-q_{mhd});$
    - $ccoil[i] = ccoil[i]*(aip/12.8);$  } }
- **Calculate flux swing required to rampup to  $I_p$ :**
  - $r = (*Machine).plasma\_major\_radius();$
  - $asp = (*Machine).plasma\_major\_radius() / (*Machine).plasma\_minor\_radius();$  // plasma aspect ratio
  - $ak = (*Machine).plasma\_elongation();$  // plasma elongation
  - $betap = (*Machine).plasma\_poloidal\_beta();$
  - $e = 1.0/asp;$
  - $ali = 0.50;$
  - $ce = 0.6;$
  - $a1 = (1.0 + 1.81*pow(e, 0.5) + 2.05*e)*log(8.0/e)-(2.0 + 9.25*pow(e, 0.5) - 1.21*e);$
  - $a2 = 0.73*pow(e, 0.5)*(1.0 + 2.0*pow(e, 4.0) - 6.0*pow(e, 5.0) + 3.7*pow(e, 6.0));$
  - $a3 = 1.0 + 0.98*pow(e, 2.0) + 0.49*pow(e, 4.0) + 1.47*pow(e, 6.0);$
  - $a4 = 0.25*e*(1.0 + 0.84*e - 1.44*pow(e, 2.0));$
  - $als = a1*(1.0 - e)/(1.0 - e + a2*ak);$
  - $am = pow(1.0 - e, 2.0)/(pow(1.0 - e, 2.0)*a3 + a4*pow(ak, 0.5));$
  - $alex2 = als - (am/4.0)*(log(8.0/(e*pow(ak, 0.5))) + betap + ali/2.0 - 3.0/2.0);$
  - $alex2 = als;$
  - $dfcsx = 1.2566*aip*r*(alex2 + (ali/2.0) + ce);$
  - $dfcsx2 = 1.2566*aip*r*(alex2 + (ali/2.0) + ce);$
- **Calculate forward and back bias coil currents:**
  - $for(int i=0; i<18; i++)$ 
    - {  $ccoilf[i] = ccoil[i] - (dcdp[i]*dfcsx2/2.0)/25.0;$
    - $ccoilb[i] = ccoil[i]/4.0 + (dcdp[i]*dfcsx2/2.0)/25.0;$  }
- **Determine maximum current in each PF coil:**
  - $ccoilmx[18];$
  - $for(int i=0; i<18; i++)$ 
    - {  $ccoilmx[i] = max( max(ccoilb[i], ccoilf[i]), ccoil[i] );$  }

- **Calculate coil thickness** based on jSC central solenoid (first 4 coils in array) outer radius of the central solenoid calculated as inner radius of the bucking cylinder minus the gap between central solenoid and bucking cylinder
  - `csbcgap = (*Machine).inboard_generic_build(23); // central solenoid to buck cyl gap`
  - `rpfout = ksiloBC - csbcgap;`
  - `Bmaxlimpf = Maximum allowed field on PF coils`
  - `ccoilsmx = max( max(ccoilmx[1], ccoilmx[2]), max(ccoilmx[3], ccoilmx[4]) );`
  - `ccoilsmx = fabs(ccoilsmx);`
  - `bmaxpf = dfcsx/(2.0*pi*pow(rpfout, 2.0));`
  - `if(bmaxpf > bmaxlimpf)`
    - `{ cout << "bmaxpf = " << bmaxpf << " which is greater than the limit" << bmaxlimpf << "\n"; (*Machine).get_data_point_filter(3, 3); return; }`
- **Maximum allowed current density in superconductor = xjsclimpf;**
  - `if(bmaxpf > 12.0)`
    - `{ xjsclimpf = -2.1e8*bmaxpf + 3.78e9; }`
  - `else`
    - `{ xjsclimpf = -4.0e7*bmaxpf + 1.74e9; }`
  - `hcs = 2.0*etahiBC; // bucking cylinder height`
- **Areas:**
  - `Asc = ccoilsmx*1.0e6/xjsclimpf; // total area of the superconductor`
  - `sigma_sheath = maximum stress in protective sheath around strands`
  - `Asheath = ccoilsmx*1.0e6*bmaxpf*rpfout/sigma_sheath; // total area of sheath`
  - `J2_tau = quench protection parameter`
  - `tau = quench time`
  - `Acu = ccoilsmx*1.0e6/pow(2.0*J2_tau/tau,0.5); // total area of the Cu`
  - `f_He = volume fraction of He coolant`
  - `Ahe = (Asc+Acu)*f_He; // total area of the coolant (He)`
  - `Astructure = ccoilsmx*1.0e6*bmaxpf*rpfout/sigma; // total area of the stainless steel structure`
  - `f_ins = volume fraction of the insulator`
  - `Ains = (Asc + Asheath + Acu + Ahe + Astructure)*f_ins/(1.0 - f_ins);`
  - `Atotal = Asc + Acu + Asheath + Astructure + Ahe + Ains; // total area of the insulator`
- **Area fractions**
  - `fstarea = (Asheath+Astructure)/Atotal;`
  - `fcuarea = Acu/Atotal;`
  - `fcoolarea = Ahe/Atotal;`
  - `finsarea = Ains/Atotal;`
  - `fsarea = 1.0 - fstarea - fcurea - fcoolarea - finsarea;`
  - `dr = 8.0*ccoilsmx*1.0e6/(hcs*fsarea*xjsclimpf);`
  - `rpfin = rpfout - dr;`
- **Locations of the PF coils**

Build the central solenoid by following the inside of the bucking cylinder up to its same height. The central solenoid is composed of 8 coils, #0-3 and #32-35. All the PF coils on the top and bottom toroidal cap are equally spaced. Realistically though, the PF coils will need to be shifted either up or down to make room for the vacuum vessel duct penetration. If the slope of the cap surface is going up, they are on the left side of the spacer. If the slope is going down, they are on the right side. The spacer `rpf[i]` is placed by dividing the width of the toroidal cap by 14, which is the number of coils on the top. Starting from the 12th coil, the spacer is shifted to the left by `dx[i]`. Mirror the top coils down to the bottom cap by symmetry.

- `rpf[i] = ksiloT + spacing*(i-3);`
- `if( i > 9 ) rpf[i] = rpf[i] - dx[i]/2.0;`  
`double dist = 1.0e6;`  
`int ipfc;`  
`for(int j=0; j<lttc; j++)`  
`{ double xttc = (*TopTCap).contour(j, 0, "outer");`  
`double yttc = (*TopTCap).contour(j, 1, "outer");`  
`if( dist > fabs(xttc-rpf[i]) &&`  
`xttc < (*TopTCap).contour(j+1, 0, "outer") )`  
`{ dist = fabs(xttc-rpf[i]);`  
`ipfc = j;`  
`if ( yttc >= (*TopTCap).contour(j-1, 1, "outer") )`  
`{ position[i] = "left"; }`  
`else`  
`{ position[i] = "right"; }`  
`}} }`  
`zpf[i] = (*TopTCap).contour(ipfc, 1, "outer");`  
`} } END`

## X. Cryogenic Dome

**Reference:** Z. Dragojlovic, 3/2007 "*Cryogenic\_Dome.pdf*"

**Code Modules:** *Geometry.cpp* "*Cryogenic Dome*"

**Notes:** The cryogenic dome is created around the entire tokamak using the TF coil as its reference. It is also sometimes called the "cryodome or cryostat."

**Reference Part:** TF coil

### How to Build

**Inner Contour:** outer contour is the inner + the cryodome thickness

- `ksilo = (*RefBody).bounding_box(0, "outer"); // min IB TF coil (x)`
- `etalo = (*RefBody).bounding_box(1, "outer"); // bottom TF coil (y)`
- `ksihi = (*RefBody).bounding_box(2, "outer"); // max OB TF coil (x)`
- `etahi = (*RefBody).bounding_box(3, "outer"); // top of TF coil (y)`
- `dwall = (*Machine).vertical_generic_build(21); // cryodome thickness`
- `xs[0] = 0.0;`
- `ys[0] = 1.5*etahi;`

- $xs[20] = ksilo + (ksihi - ksilo) * 1.1;$
- $ys[20] = etahi * 1.1;$
- $for(int i=1; i<20; i++)$
- $\{ xs[i] = xs[0] + i * xs[20] / 20;$
- $ys[i] = ys[20] + (ys[0] - ys[20]) * 2.0 / \pi * \text{atan}(2.0 * \text{fabs}(xs[20] - xs[i])); \}$
- $xs[21] = xs[20];$
- $ys[21] = 1.5 * etalo;$
- $xs[22] = 0.001;$
- $ys[22] = ys[21];$
- $xs[23] = 0.0;$
- $ys[23] = ys[21];$

Build the cryodome by following the contour lines and reference points as indicated above. Then, build the outer contour at the thickness given by the vertical build input file.

## Y. Maintenance Port Window

**Reference:** *No specific documentation.*

**Code Modules:** *Geometry.cpp “Maintenance Port Window in-between the TF Coils”*

**Notes:** A maintenance port window resides in each sector between TF coils and allows access to the OB vacuum vessel door for removal and maintenance of the power core components.

**Reference Part:** TF coil and vacuum vessel

### How to Build

The maintenance port window is simply the area enclosed by the width between the TF coils and the height of the outboard vacuum vessel. See also Maintenance Ports build.

## Z. Maintenance Ports

**Reference:** *No specific documentation.*

**Code Modules:** *Geometry.cpp “Maintenance Ports”*

**Notes:** The maintenance ports are the areas in each sector between TF coils. The ports allow access to the OB vacuum vessel door for removal and maintenance of the power core components.

**Reference Part:** vacuum vessel and cryodome

### How to Build

- $ductwall = (*Machine).vertical\_generic\_build(11)/4.0; // 1/4th VV thickness$

### **Volume and Surface Areas**

- $(*MPort).get\_volume(); // volume of one side wall of the maintenance port$
- $vSideWall = (*MPort).volume();$

#### Volume of the top and bottom walls of the maintenance port

- $r_{Top} = (*MPort).bounding\_box(0, "outer");$
- $l_{Top} = ksihi - r_{Top};$
- $v_{TopWall} = 2.0 * r_{Top} * \sin(\pi/16.0) * l_{Top} * ductwall;$

#### Volume of the outer door of the maintenance port

- $\eta_{MPhi} = (*MPort).bounding\_box(3, "outer");$
- $\eta_{MPlo} = (*MPort).bounding\_box(1, "outer");$
- $v_{OuterDoor} = 2.0 * r_{Top} * \sin(\pi/16.0) * (\eta_{MPhi} - \eta_{MPlo}) * ductwall;$

#### Amount of volume to add to already included side wall:

- $v_{Plus} = 2.0 * v_{TopWall} + v_{SideWall} + v_{OuterDoor};$
- $(*MPort).add\_to\_volume(v_{Plus});$

Build the maintenance port by coming directly off the top and bottom of the vacuum vessel components and extending out radially until the cryodome is reached. The width of the port is the distance between the TF coils specified at the radius of the outboard vacuum vessel. The volume of the maintenance port is quite substantial – see Figure 5 for a sense of scale. The volume includes all the space enclosed within the pie-shaped bounding box of the OB VV door to the cryodome at a height of the entire vacuum vessel.

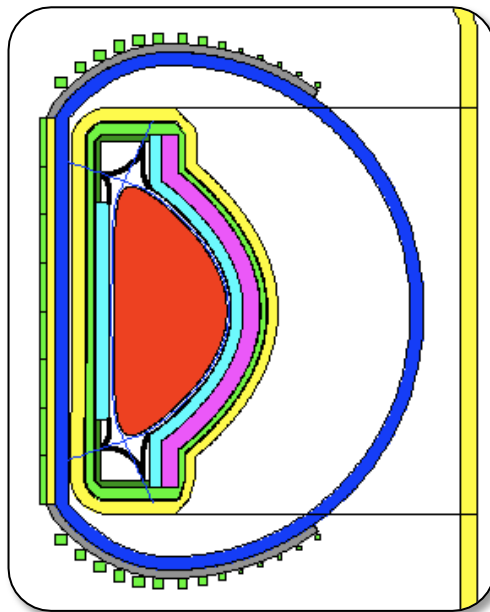


Figure 5. SCLL radial build showing maintenance port extending radially from OB VV out in-between the TF coil, then through the cryodome. (12/2011)

## AA. Upper Vacuum Vessel Duct

**Reference:** *No specific documentation.*

**Code Modules:** *Geometry.cpp “Upper Vacuum Vessel Duct – Bottom, Top and Side Wall”*

**Notes:** The vacuum ducts provide a way to access the inside of the vacuum vessel for pumping. The vacuum ducts penetrate the vacuum vessel, route in-between the TF coils, penetrate the upper toroidal cap, nudge the PF coils slightly so there is room to come between them and penetrate the cryostat for connection to vacuum pumps outside the tokamak. See Figure 6 for a visualization.

**Reference Part:** vacuum vessel, TF coil and cryodome

### **How to Build – Bottom Wall**

This design function builds the bottom wall of the upper vacuum duct.

- etahiVV = (\*Vessel)[0].bounding\_box(3, "outer"); // top of vert VV (y)
- ksihiVV = (\*Vessel)[0].bounding\_box(2, "outer"); // OB VV (x)
- ksiloVV = (\*Vessel)[0].bounding\_box(0, "outer"); // IB VV (x)
- ksimedVV = 0.4\*(ksiloVV + ksihiVV); // middle of VV
- ksihiCryo = (\*Cryo).bounding\_box(2, "outer"); // Relevant bounding box coordinates of the outer cylindrical surface of the cryostat
- etahiMagnet = (\*Magnet).bounding\_box(3, "inner"); // Ordinate of the highest point of the inner surface of the TF magnet
- hVVTF = etahiMagnet - etahiVV; // Largest vert gap between vacuum vessel and TF coil
- ductwidth = 0.15\*(ksihiVV - ksiloVV); // Width of the duct in the vertical plane
- ductwall = (\*Machine).vertical\_generic\_build(11)/4.0; // Duct wall thickness = 1/4th VV thickness

Build the bottom wall by referencing the top of the vertical VV at the location “ksimedVV+ductwidth.” Come up through the TF coil and coil cap at 45 degrees and go radially outward through the cryodome. The top wall build is similarly constructed at a distance “ductwidth” above it.

### **How to Build – Top Wall**

This design function builds the top wall of the upper vacuum duct.

- etahiVV = (\*Vessel)[0].bounding\_box(3, "outer"); // top of vert VV (y)
- ksihiVV = (\*Vessel)[0].bounding\_box(2, "outer"); // OB VV (x)
- ksiloVV = (\*Vessel)[0].bounding\_box(0, "outer"); // IB VV (x)
- ksimedVV = 0.4\*(ksiloVV + ksihiVV); // middle of VV
- ksihiCryo = (\*Cryo).bounding\_box(2, "outer"); // Relevant bounding box coordinates of the outer cylindrical surface of the cryostat
- etahiMagnet = (\*Magnet).bounding\_box(3, "inner"); // Ordinate of the highest point of the inner surface of the TF magnet
- hVVTF = etahiMagnet - etahiVV; // Largest vert gap between vacuum vessel and TF coil
- ductwidth = 0.15\*(ksihiVV - ksiloVV); // Width of the duct in the vertical plane
- ductwall = (\*Machine).vertical\_generic\_build(11)/4.0; // Duct wall thickness = 1/4th VV thickness

Build the top wall by referencing the top of the vertical VV at the location “ksimedVV.” Come up through the TF coil and coil cap at 45 degrees and go radially outward through the cryodome. The bottom wall build is similarly constructed at a distance “ductwidth” below it.

### How to Build – Side Wall

This design function builds the two side walls of the upper vacuum duct. The vacuum duct has a trapezoidal cross section. The top side length is denoted as "a", bottom side as "b", lateral sides as "c" and height as "h". The thickness of the vacuum duct wall is denoted as "d". The side walls are not viewable with the 2-dimensional views shown here.

### Sides of the Trapezoid

- `a = (*Duct)[1].cross_section_depth(0);` // top side of the trapezoidal cross section
- `b = (*Duct)[0].cross_section_depth(0);` // bottom side of the trapezoidal cross section
- `etahiTopWall = (*Duct)[1].bounding_box(3, "outer");`
- `etahiBottomWall = (*Duct)[0].bounding_box(3, "outer");`
- `h = (etahiTopWall - etahiBottomWall);` // height of the trapezoid
- `c = pow( pow(h, 2.0) + pow( (b-a)/2.0, 2.0), 0.5);` // lateral side of trapezoidal cross section

### Wall Thickness

- `dwall = (*Duct)[0].cross_section_width(0);`

### Total Lengths of the Top and Bottom Wall of the Vacuum Duct

- `ITW = (*Duct)[1].volume()/(dwall*a);`
- `IBW = (*Duct)[0].volume()/(dwall*b);`

### Volume of the Side Wall of the Vacuum Duct

- `vSW = 0.5*(ITW + IBW)*c*dwall;`
- `(*Duct)[2].add_to_volume(vSW);` // only place Duct[2] is used here

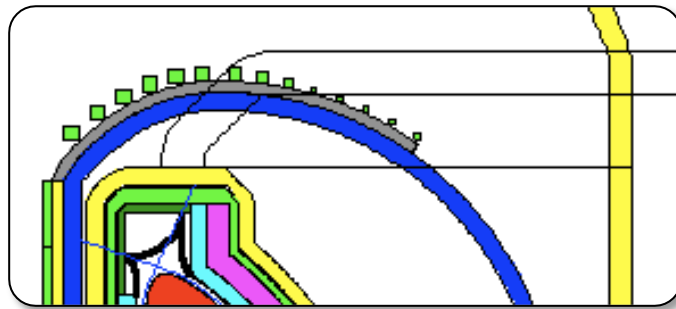


Figure 6. SCLL radial build with vacuum vessel ducting (12/2011)

...END of Parts...

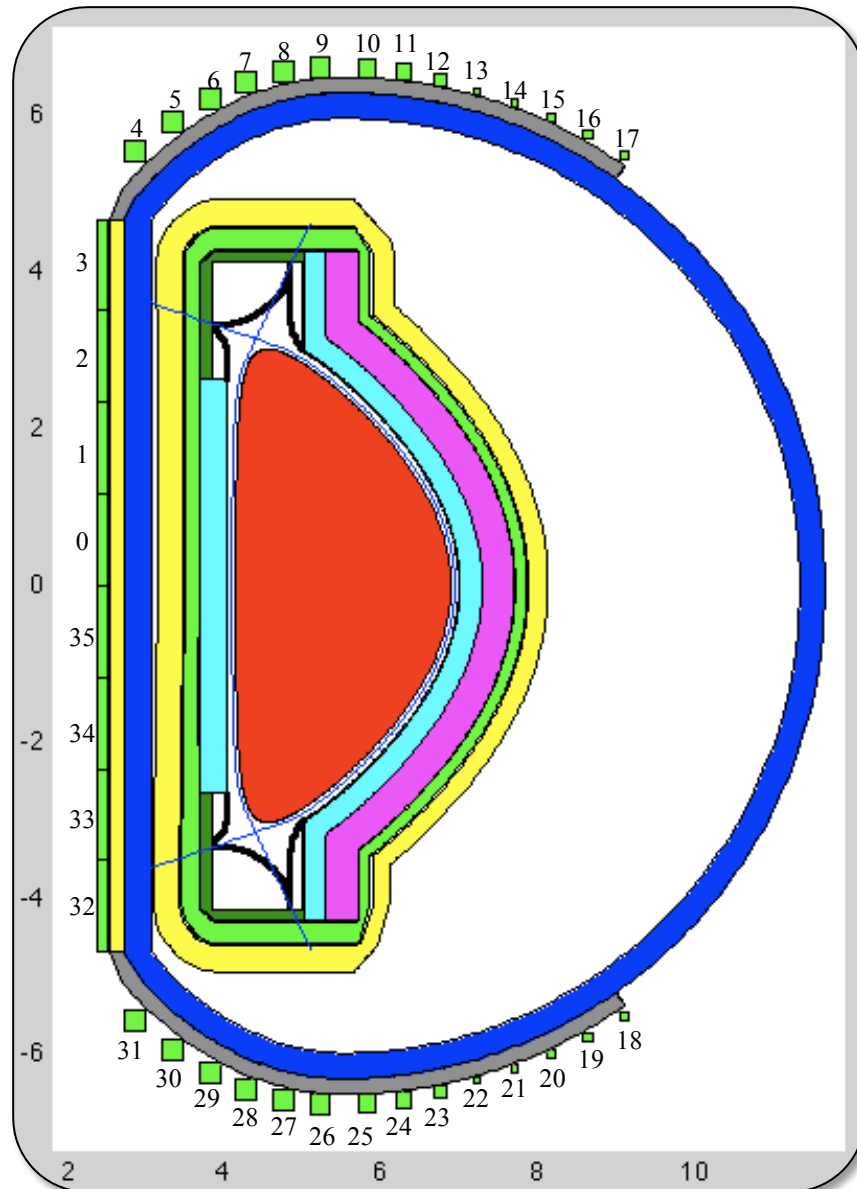


**Figure 7. SCLL radial build with PF coil enumeration (12/2011)**