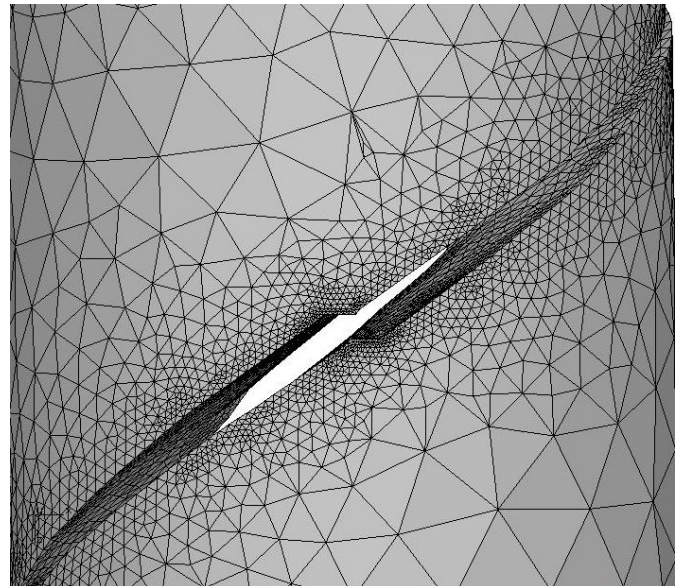


# FRANC3D

## Command Language & Python Extensions

Version 8.2



Fracture Analysis Consultants, Inc  
[www.fracanalysis.com](http://www.fracanalysis.com)

Revised: October 2022

# Table of Contents:

Table of Contents:	2
1. Introduction	4
2. Command File Syntax	4
2.1 Commands	8
2.1.1 AutoGrowth( )	8
2.1.1.1 sif_params	9
2.1.1.2 growth_params	9
2.1.1.3 growth_plan	12
2.1.1.4 growth_variables	12
2.1.1.5 template_params	12
2.1.1.6 front_fitting_params	13
2.1.1.7 general_analysis_options	13
2.1.2 CheckGrowthStatus( )	17
2.1.3 CloseModel( )	18
2.1.4 ComputeCOD( )	18
2.1.5 ComputeGrowthParams( )	18
2.1.6 ComputeSif( )	18
2.1.7 CrackTractConst( )	19
2.1.7.1 CFT index and load case	19
2.1.8 CrackTractDelete( )	19
2.1.9 CrackTractExternalDist( )	19
2.1.10 CrackTractSurface( )	20
2.1.11 CrackTract1DRad( )	20
2.1.12 CrackTract2DRad( )	21
2.1.13 FretModelImport( )	21
2.1.14 FretNucleationCycles( )	22
2.1.15 FretNucleationDataImport( )	22
2.1.16 GetBuildInfo( )	23
2.1.17 GetCrackData( )	23
2.1.18 GetIntegrationResults( )	23
2.1.19 GetGrowthStatus( )	23
2.1.20 GrowCrack( )	24
2.1.21 GrowCrackFromFile( )	24
2.1.22 GrowMergeCrack( )	24
2.1.23 Include( )	25
2.1.24 InsertFileFlaw( )	25
2.1.24.1 flaw_insert_params	26
2.1.25 InsertMultFileFlaw( )	26
2.1.26 InsertMultParamFlaw( )	27
2.1.27 InsertParamFlaw( )	27
2.1.28 InsertUserBdryFlaw( )	28
2.1.29 InsertUserMeshFlaw( )	29
2.1.30 IntegrateStep( )	29
2.1.31 OpenFdbModel( )	29

2.1.32	OpenMeshModel( )	30
2.1.33	ReadFullGrowthHist ( )	30
2.1.34	ReadGrowthParams ( )	31
2.1.35	ReadResponse( )	31
2.1.36	RunAnalysis( )	31
2.1.37	SaveFdbModel( )	32
2.1.38	SaveGrowthParams( )	32
2.1.39	SaveMeshModel( )	33
2.1.40	SetEdgeParameters( )	33
2.1.41	SetGrowthParams( )	34
2.1.42	SetLoadSchedule( )	34
2.1.43	SetMeshingParameters( )	34
2.1.44	SetStatusFile( )	35
2.1.45	SetUnits( )	36
2.1.46	SetUserExtensionsFile( )	36
2.1.47	SetWorkingDirectory( )	36
2.1.48	SifHistory( )	37
2.1.49	<i>StartRecording( )</i>	38
2.1.50	Submodeler( )	38
2.1.51	WriteCOD( )	38
2.1.52	WriteCrackData( )	39
2.1.53	WriteEPJ( )	39
2.1.54	WriteFatigueData( )	40
2.1.55	WriteGrowthData( )	41
2.1.56	WriteGrowthParams( )	41
2.1.57	WriteResolvedSif( )	42
2.1.58	WriteResolvedSifPath( )	43
2.1.59	WriteSERR( )	43
2.1.60	WriteSif( )	44
2.1.61	WriteSifPath( )	45
2.1.62	WriteStdTempData( )	45
2.2	Example Command File	46
3.	Python Module	49
3.1	Command Converter	49
3.2	Python Module	49
3.2.1	Setting the Path	50
3.2.2	class F3DApp	51
3.2.3	class CrackData	60
3.2.4	class CrackStep	61
3.2.5	class CrackFront	61
3.2.6	class FrontPoint	62
3.2.7	class IntegrationResults	63
4.	Python Crack Growth Extensions	64
4.1	Data Access Functions	65
4.2	Specifying a user extension file	67
4.3	Example user extension files	68

# 1. Introduction

This document describes the FRANC3D command language and the Python extensions.

The PyF3D module works with Python versions 3.6 - 3.9. If your system does not have one of these versions, you can download (and build) Python yourself (see [www.python.org](http://www.python.org)).

Note that the PATH and PYTHONPATH environment variables might need to be set for all the Python modules to work correctly. The folder that contains the franc3d executable and the PyF3D modules can be added to these environment variables. Additional information is provided in Section 3.

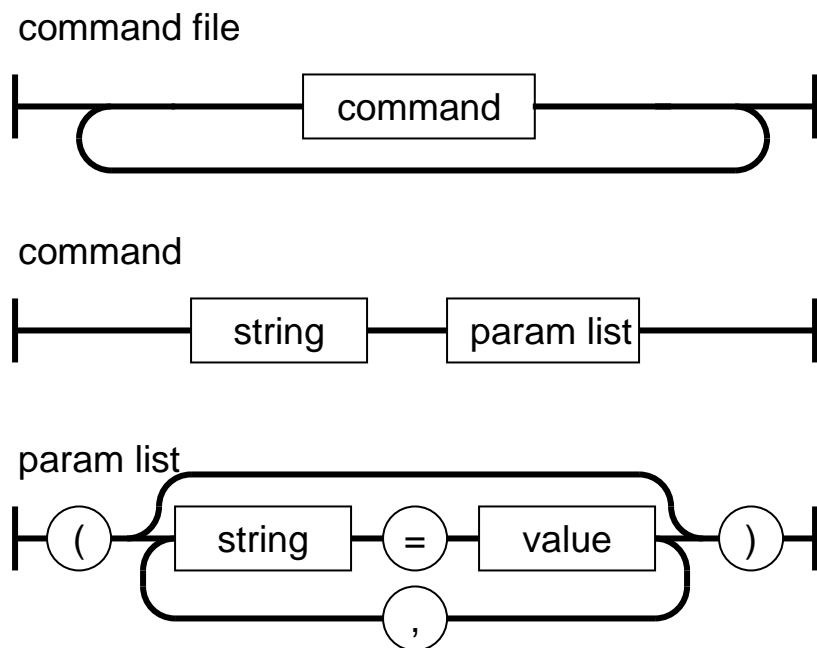
## 2. Command File Syntax

When running FRANC3D using the standard GUI menu and dialogs, a session file is saved that contains the commands executed through the GUI. A user can play-back these commands to reproduce their actions or edit the file to execute different or modified commands without using the GUI.

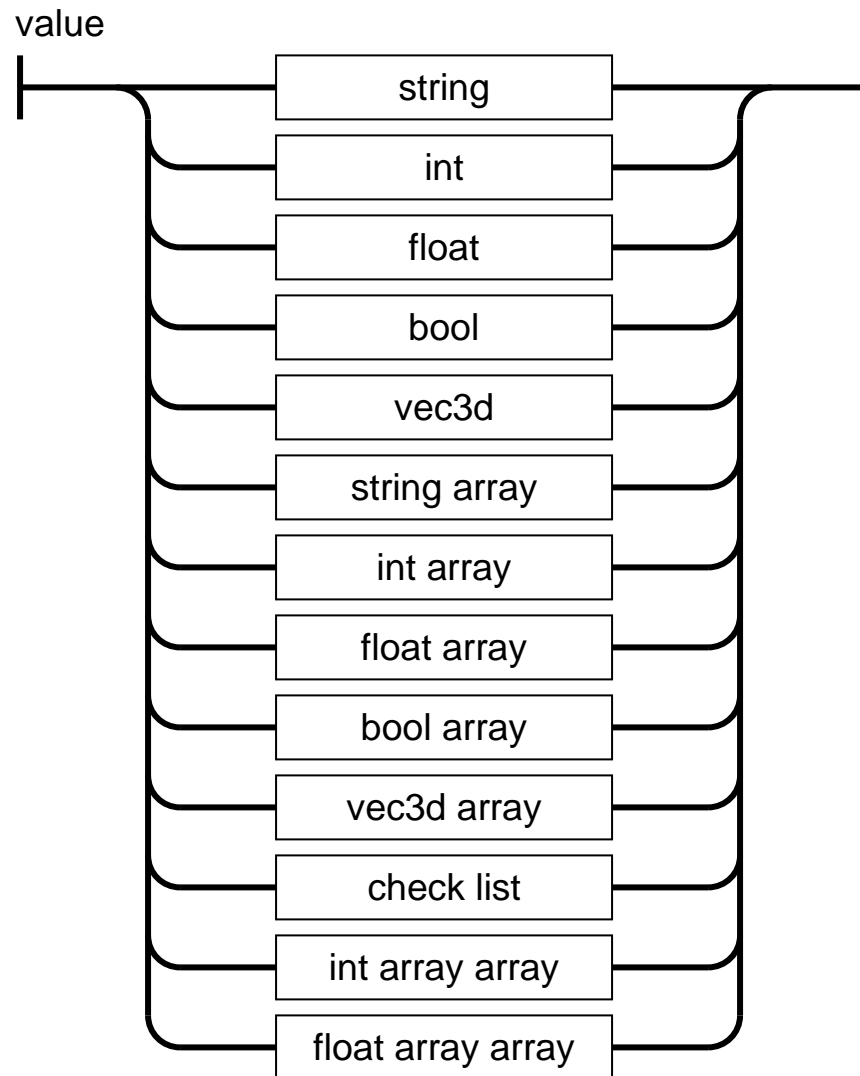
These session files contain a series of command statements. In command files, lines starting with '#' are comment lines. Informally, the command statements look something like:

```
command_a(param_1=value1,param_2="string param")
```

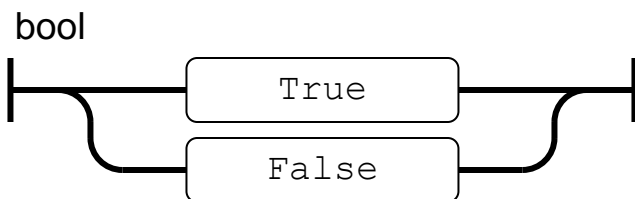
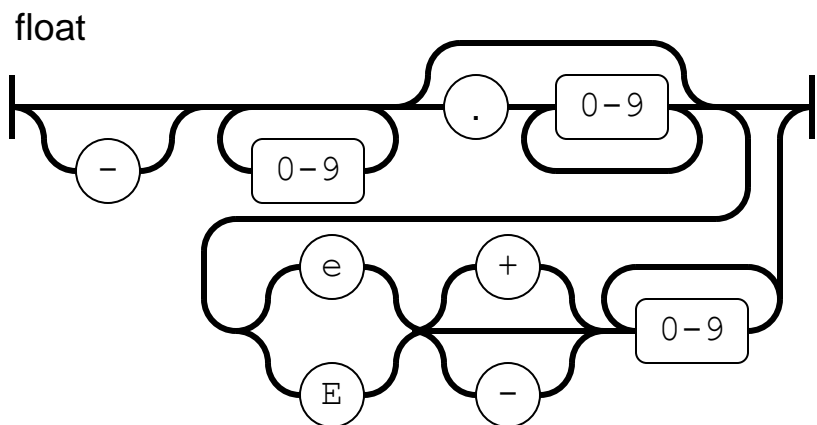
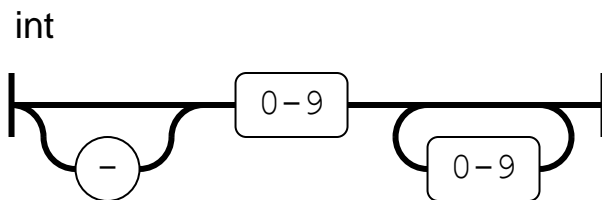
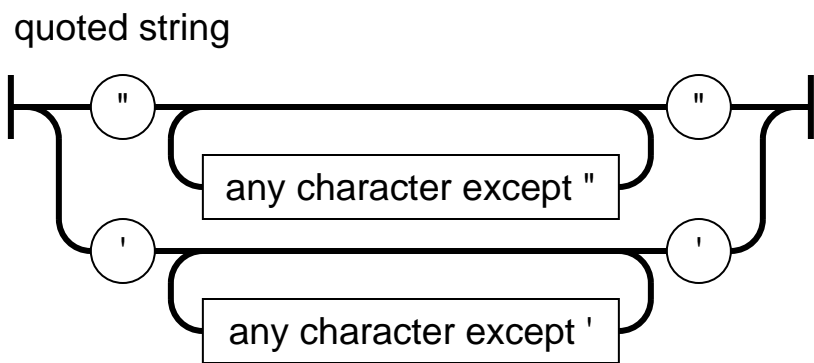
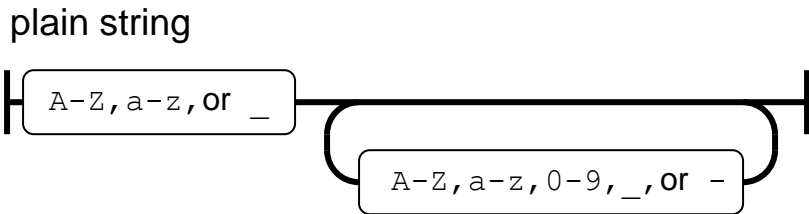
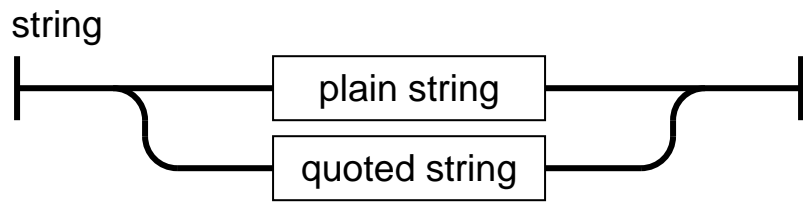
The syntax diagram for a command file is:



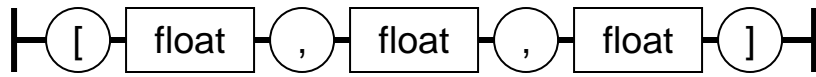
Each parameter value is one of the following types:



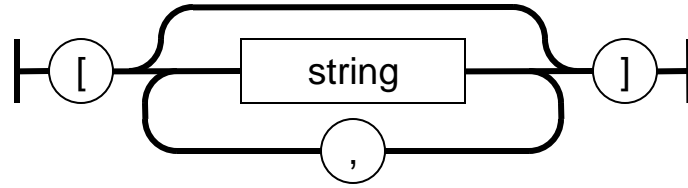
Where the types are defined as:



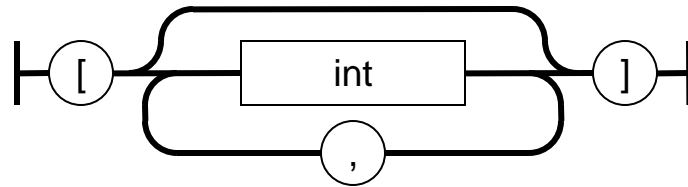
vec3d



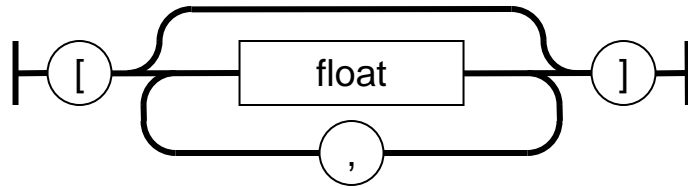
string array



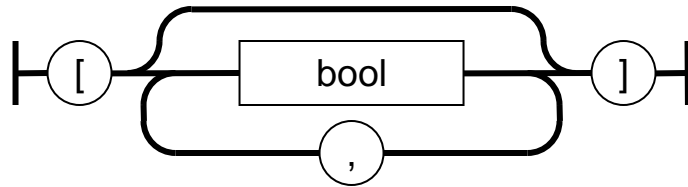
int array



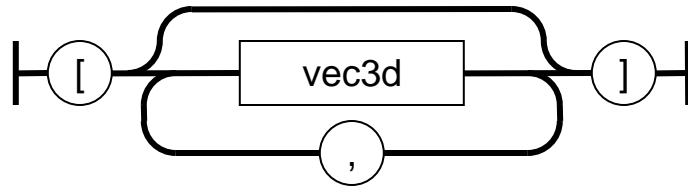
float array



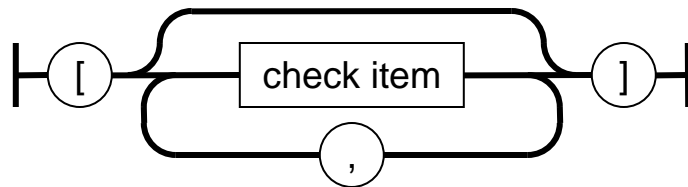
bool array



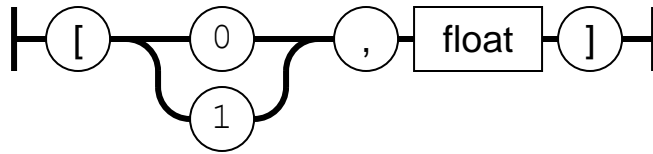
vec3d array



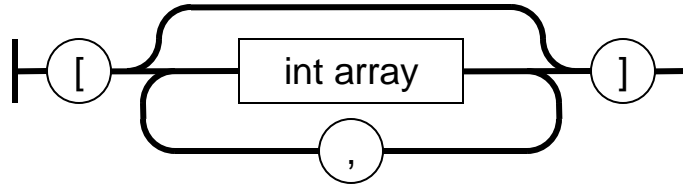
check list



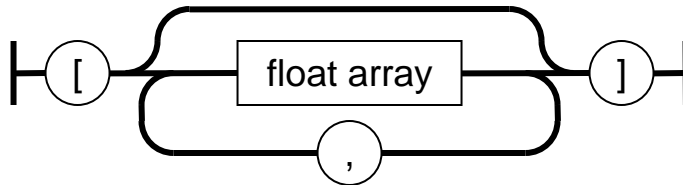
check item



int array array



float array array



## 2.1 Commands

The FRANC3D commands and their parameter lists are described here, and an example for each command is provided.

The command parameter type is given after the parameter name in *italics*. Note that (req) means that a parameter is required and (opt) means that a parameter is optional.

### 2.1.1 AutoGrowth()

Automatically grow crack(s).

*parameters:*

model\_type = *string* (req) – model type: ABAQUS, ANSYS or NASTRAN

cur\_step = *int* (req) – current crack growth step number

file\_name = *single quoted string* (req) - base file name

sif\_params (opt) – see 2.1.1.1

growth\_plan (opt) – see 2.1.1.3

template\_params (opt) – see 2.1.1.5

front\_fitting\_params (opt) – see 2.1.1.6

analysis\_options (req) – see 2.1.1.7

*example:*

```
AutoGrowth (  
    model_type=ABAQUS,
```



```

cur_step=1,
file_name='acube_init_crack',
num_steps=5,
step_type=SCONST,
const_step_size=0.1,
check_DKth=true,
maximum_steps=5,
temp_radius_type=RELATIVE,
temp_radius=65,
extrapolate=[[3,3]],
flags=[TRANSFER_BC,NO_CFACE_TRACT,NO_CFACE_CNTCT],
connection_type=MERGE,
command='"abaqus.bat" job=Abaqus_cube_crack_STEP_001
ask_delete=NO -interactive -analysis ')

```

### 2.1.1.1 *sif\_params*

*parameters:*

sif\_method = *string* (opt) - SIF computation method  
M\_INTEGRAL - use the *M*-integral (interaction integral) technique (default)  
DISP\_CORR - use the displacement correlation technique  
VCCT - use the virtual crack closure technique  
do\_therm\_terms = *bool* (opt) - include thermal terms  
ref\_temp = *float* (opt) - reference temperature, default = 0.0  
do\_press\_terms = *bool* (opt) - include crack-face pressure terms  
do\_crack\_face\_contact = *bool* (opt) - include contact pressure terms  
large\_rotations = *bool* (opt) - correction term large rigid body type rotation

*(initial stress options)*

initial\_model\_type = *string* (opt) – initial stress model type  
initial\_mesh\_file = *string* (opt) – initial stress mesh file name  
initial\_stress\_file = *string* (opt) – initial stress results file name  
initial\_stress\_step = *int* (opt) – initial stress results load step  
initial\_stress\_substep = *int* (opt) – initial stress results load substep  
initial\_stress\_scale = *float* (opt) – initial stress results scalar multiplier

do\_epj = *bool* (opt) – compute elasto-plastic J-integral values

### 2.1.1.2 *growth\_params*

*parameters:*

growth\_type = *string* (opt) - growth computation method:  
SUBCRITICAL, QUASI\_STATIC

*fem\_units* = *string* (opt) – US or SI units:  
 KSI\_IN, PSI\_IN, MPA\_MM, MPA\_M, PA\_MM, PA\_M  
*equiv\_type* = *string* (opt) – method for computing *K\_equiv*:  
 EQUIV\_KI, EQUIV\_SRR, EQUIV\_KRSS  
*SERR\_equiv\_sign* = *string* (opt) – option for setting sign of *K\_equiv*:  
 FROM\_KI, FROM\_KII, FROM\_KIII, ALWAYS\_POS, ALWAYS\_NEG  
*gamma\_II* = *float* (opt) – strain energy release rate mode II factor  
*gamma\_III* = *float* (opt) – strain energy release rate mode III factor  
*closure\_flag* = *bool* (opt) – closure  
*accelerate* = *bool* (opt) – accelerated integration  
*constant\_time\_k* = *bool* (opt) – use constant *K* value for specified time  
*extension\_type* = *string* (opt) – extension type:  
 MEDIAN\_EXT, CYCLES\_EXT, USER\_EXT  
*load\_schedule\_data* = *string list* (opt) – load schedule data; see Section 2.1.1.2.1  
*load\_schedule\_label* = *string* (opt) – load schedule description  
*growth\_model* = *string list* (opt) – crack growth rate model; see Section 2.1.1.2.2  
*growth\_model\_label* = *string* (opt) – crack growth rate description  
*eta\_II* = *float* (opt) – factor for Mode II of *K\_equiv*  
*eta\_III* = *float* (opt) – factor for Mode III of *K\_equiv*  
*load\_sequence\_alg* = *string* (opt) – load sequence retardation model  
 GEN\_WILLENBORG, MOD\_GEN\_WILLENBORG  
*willenborg\_shutoff* = *float* (opt) – Willenborg shutoff  
*willenborg\_phi0* = *float* (opt) – Willenborg  $\phi_0$   
*kink\_angle\_strategy* = *string* (opt) – crack growth kink angle model  
 MTS, MSS, GEN, SERR, PLANAR, USER\_ANG  
*quasi\_static\_n* = *float* (opt) – quasi-static crack growth power  
*quasi\_static\_loads* = *string list* (opt) – quasi-static crack growth load steps  
*user\_py\_file* = *string* (opt) – Python user defined crack growth  
*aniso\_tough\_params* = *float array* (opt) – anisotropic toughness values  
*saved\_file\_name* = *string* (opt) – file name for saving parameters  
*load\_schedule* = *string* (opt) – file name for load schedule  
*load\_step\_map* = *string* (opt) – FE load step map; see Section 2.1.1.2.3

### 2.1.1.2.1 *load\_schedule\_data*

*parameters:*

*version* = *int* (req) – load schedule data version  
*schedule* = *string list* (req) – load schedule

*example:*

```

load_schedule_data=[\+
"VERSION: 1",\+
"SCHEDULE (",\+
"REPEAT_COUNT: FOREVER",\+
"NUM_CHILDREN: 1",\+
"TRANSIENT (",\+

```

```
"REPEAT_COUNT: 1",\,+
"CASES:",\,+
"NUM_STEPS: 5",\,+
"1 NONE 1 1 0",\,+
"2 NONE 1 1 0",\,+
"3 NONE 1 1 0",\,+
"4 NONE 1 1 0",\,+
"5 NONE 1 1 0",\,+
")",\,+
")"]
```

### 2.1.1.2.2 *growth\_model*

*parameters:*

`version` = *int* (req) – crack growth rate data version  
`num_models` = *int* (req) – number of growth rate models

*example:*

```
growth_model=[\,+
"VERSION: 2",\,+
"NUM_MODELS: 1",\,+
"CYCLES_RATE_TYPE: PARIS",\,+
"CYCLES_RATIO_TYPE: NONE",\,+
"CYCLES_TEMP_INTERP: TEMP_NONE",\,+
"CYCLES_TITLE: ",\,+
"CYCLES_DESCRIPTION: 0",\,+
"CYCLES_PROP_UNITS: MM|MPA|C|SEC",\,+
"C: 1e-11 n: 3 DKth: 0.1 Kc: 100 ",\,+
"TIME_RATE_TYPE: NONE"]
```

### 2.1.1.2.3 *load\_step\_map*

*parameters:*

`version` = *int* (req) – load step map data version  
`sub_range` = *string list* (req) – load step map

*example:*

```
load_step_map=[\,+
"VERSION: 1",\,+
"MAX_SUB: 5",\,+
"0 0",\,+
"0 0",\,+
"0 0",\,+
"0 0",\,+
"0 0",\,+
"0 0",\,+
"LABELS: 5",\,+
"5 Load Step 5",\,+
```

"4 Load Step 4",\n+\n"3 Load Step 3",\n+\n"2 Load Step 2",\n+\n"1 Load Step 1"]],

### 2.1.1.3 *growth\_plan*

*num\_steps* = *int* (opt) – number of crack growth steps  
*step\_type* = *string* (opt) – type of crack growth increment model  
*const\_step\_size* = *float* (opt) – constant crack growth median increment  
*lin\_step\_start* = *float* (opt) – linear model crack growth start value  
*lin\_step\_inc* = *float* (opt) – linear model crack growth increment  
*user\_step* = *float array* (opt) – user-defined crack growth increment list  
*check\_Kc* = *bool* (opt) – check to see if  $K > K_c$   
*check\_DKth* = *bool* (opt) – check to see if  $DK < DK_{th}$   
*maximum\_steps* = *int* (opt) – maximum number of crack growth steps  
*maximum\_cycles* = *int* (opt) – maximum number of cycles  
*maximum\_time* = *float* (opt) – maximum time  
*maximum\_depth* = *float* (opt) – maximum crack depth  
*check\_HCF\_threshold* = *int* (opt) – check against HCF threshold  
*check\_growth\_fail* = *bool* (opt) – check to see if all fronts advance

### 2.1.1.4 *growth\_variables*

*median\_step* = *float* (opt) – median crack growth step size  
*dist\_step* = *float* (opt) – crack growth step size at *scale\_node*  
*scale\_node* = *float* (opt) – normalized SIF position; 0 is  $K_{min}$  , 1 is  $K_{max}$   
*cycles\_step* = *float* (opt) – crack growth cycles  
*time\_step* = *float* (opt) – time of crack growth  
*start\_cycle* = *float* (opt) – starting cycle count  
*start\_time* = *float* (opt) – starting time  
*front\_mult* = *float array* (opt) – crack front extension factors

### 2.1.1.5 *template\_params*

*use\_templates* = *bool* (opt) – use template flag  
*temp\_radius\_type* = *string* (opt) – template radius type  
ABSOLUTE, RELATIVE  
*temp\_radius* = *float* (opt) – template radius type  
*temp\_prog\_ratio* = *float* (opt) – template progression ratio  
*temp\_num\_rings* = *int* (opt) – template number of rings of elements  
*temp\_num\_circ* = *int* (opt) – template number of elements around front  
*temp\_max\_aspect* = *float* (opt) – template element aspect ratio  
*temp\_simple\_interserct* = *bool* (opt) – use simple intersections

### 2.1.1.6 *front\_fitting\_params*

smoothing\_method = *string array* (opt) – smoothing type for each crack front  
KINK\_EXTEN\_POLY - polynomial fit to kink angles and extension  
FIXED\_ORDER\_POLY – polynomial fit through front points  
MULTIPLE\_POLY – three polynomials fit through front points  
CUBIC\_SPLINE – cubic-spline fit through front points  
MOVING\_POLY – moving polynomial fit through front points  
NOFIT\_EXTRAP – no fitting but extrapolate ends  
HERMITIAN – Hermitian polynomial fit through closed-front points  
polynomial\_order = *int array* (opt) – polynomial order for each front  
discard = *int array* (opt) – discard end points for each front  
extrapolate = *float array* (opt) – extrapolate curve ends for each front  
mult\_poly\_ratio = *int array* (opt) – multiple polynomial ratio  
mv\_poly\_range = *int array* (opt) – moving polynomial range  
retain\_all\_nodes = *bool array* (opt) – retains nodes as geometric points  
on the new crack front; only applies to GrowCrackFromFile  
auto\_adjust\_fit = *bool* (opt) – flag to turn on/off automatic fit adjustment  
is\_partial\_fit = *bool array* (opt) – flag for partial extension fitting  
partial\_extension\_tol = *float array* (opt) – tolerance to determine partial extension

### 2.1.1.7 *general\_analysis\_options*

flags = *string* (req) - list of analysis options:  
CFACE\_CNTCT - enforce crack face contact  
NO\_CFACE\_CNTCT - do not enforce crack face contact (default)  
CFACE\_TRACT - write crack face tractions (default)  
NO\_CFACE\_TRACT - do not write crack face tractions  
FILE\_ONLY - write analysis files only  
CONTOUR\_INTEGRAL - perform contour integral calculation  
(applies to ABAQUS or ANSYS)  
NL\_CONTOUR\_INTEGRAL - perform FRANC3D non-linear integral calculation  
(applies to ABAQUS)  
ALL\_FRAMES - write output for all substeps/iterations of load step  
FULL\_OUTPUT - write output for all nodes of full model  
TEMPLATE\_OUTPUT - write output for just template nodes  
  
front\_elem\_type = *string* (opt) - type of elements to place at the crack front:  
WEDGE - natural wedge elements (default)  
COLLAPSED - collapsed brick, front nodes constrained  
BLUNTED - collapsed brick, front nodes unconstrained  
connection\_type = *string* (opt) - method to join the submodel and global model:  
MERGE - combine coincident nodes (default)

CONSTRAIN - join using constraint equations  
 CONTACT - insert contact conditions between models  
 merge\_tol = *float* (opt) – tolerance for merging nodes for local/global connection  
 global\_model = *single quoted string* (opt) - name of the global model  
 merge\_surf\_labels = *string array* (opt) - labels for the local connection surface  
 global\_surf\_labels = *string array* (opt) - labels for the global connect surface  
 global\_is\_master = *bool* (opt) – global connection surface is master if true  
 command = *single quoted string* (opt) - analysis command  
 python\_exe = *single quoted string* (opt) - Python executable  
 python\_script = *single quoted string* (opt) – file name of the Python script  
  
 crack\_face\_contact = *bool* (opt) – define crack face contact if true  
  
 contact\_masters = *string list* (opt) – list of contact-master surfaces  
 contact\_slaves = *string list* (opt) – list of contact-slave surfaces  
  
 constraint\_masters = *string list* (opt) – list of constraint-master surfaces  
 constraint\_slaves = *string list* (opt) – list of constraint-slave surfaces  
 del\_file\_list = *string list* (opt) – list of file extensions to be deleted

For ANSYS, there are solver-specific options:

license = *string* (opt) – license string  
 use\_mpi = *bool* (opt) – use MPI solver  
 mpi\_lib = *int* (opt) – MPI library type  
 set\_jobname = *int* (opt) – jobname setting  
 jobname = *string* (opt) – jobname  
 add\_usercmd = *bool* (opt) – flag for user-option added to command  
 usercmd = *string* (opt) – user command  
 connection\_modify = *int* (opt) – modify the local+global merge

Contact and constraint connections have additional parameters and data that are different for each analysis code.

#### **2.1.1.7.1 ABAQUS Contact or Constraint**

For local+global base connections:

locglob\_contact\_type = *int* (opt) – contact type; 0=general, 1=surface to surface  
 locglob\_contact\_surf\_interact = *string* (opt) – surface interaction name  
 locglob\_contact\_surf\_behavior = *int* (opt) – surface behavior  
 locglob\_contact\_friction = *float* (opt) – surface friction coefficient  
 locglob\_contact\_small\_sliding = *bool* (opt) – small sliding flag  
 locglob\_contact\_tied = *bool* (opt) – tied (bonded/glued) contact flag

logglob\_contact\_adjust = *float* (opt) – amount node positions can be adjusted

logglob\_constraint\_adjust = *float* (opt) – amount node positions can be adjusted

logglob\_constraint\_pos\_tol = *float* (opt) – tolerance

For crack face contact:

crack\_contact\_type = *string* (opt) – contact type; 0=general, 1=surface to surface

crack\_contact\_surf\_interact = *string* (opt) – surface interaction name

crack\_contact\_surf\_behavior = *int* (opt) – surface behavior

crack\_contact\_friction = *float* (opt) – surface friction coefficient

crack\_contact\_small\_sliding = *bool* (opt) – small sliding flag

crack\_contact\_tied = *bool* (opt) – tied (bonded/glued) contact flag

crack\_contact\_adjust = *float* (opt) – amount node positions can be adjusted

crack\_contact\_nlgeom = *bool* (opt) – turn on/off solver nlgeom flag

For extra connections:

contact\_connection = *string* (opt) – connection type; PAIRED, SINGLE

contact\_type = *string* (opt) – contact type; 0=general, 1=surface to surface

contact\_surf\_interact = *string* (opt) – surface interaction name

contact\_surf\_behavior = *int* (opt) – surface behavior

contact\_friction = *float* (opt) – surface friction coefficient

contact\_small\_sliding = *bool* (opt) – small sliding flag

contact\_tied = *bool* (opt) – tied (bonded/glued) contact flag

contact\_adjust = *float* (opt) – amount node positions can be adjusted

constraint\_adjust = *float* (opt) – amount node positions can be adjusted

constraint\_pos\_tol = *float* (opt) – tolerance

The extra connections are defined as lists. For example, for two extra contact connections, the contact\_masters=[[m\_surf\_1],[m\_surf\_2]] provides the main contact surfaces.

#### **2.1.1.7.2 ANSYS Contact or Constraint**

For local+global base connections:

logglob\_contact\_symmetric\_pair = *bool* (opt) – create symmetric pair

logglob\_contact\_as\_target = *bool* (opt) –

logglob\_contact\_mat\_id = *int* (opt) – material ID

logglob\_contact\_mu = *float* (opt) – coefficient of friction

logglob\_contact\_real\_id = *int* (opt) – real constant ID

logglob\_contact\_real\_mod\_3 = *float* (opt) – real property

logglob\_contact\_real\_mod\_4 = *float* (opt) – real property

logglob\_contact\_et170\_id = *int* (opt) – element type 170 ID

locglob\_contact\_et174\_id = *int* (opt) – element type 174 ID  
locglob\_contact\_keyopt\_2 = *int* (opt) – key option  
locglob\_contact\_keyopt\_5 = *int* (opt) – key option  
locglob\_contact\_keyopt\_8 = *int* (opt) – key option  
locglob\_contact\_keyopt\_9 = *int* (opt) – key option  
locglob\_contact\_keyopt\_10 = *int* (opt) – key option  
locglob\_contact\_keyopt\_12 = *int* (opt) – key option

locglob\_constraint\_dofx = *bool* (opt) – include X dof in constraint  
locglob\_constraint\_dofy = *bool* (opt) – include Y dof in constraint  
locglob\_constraint\_dofz = *bool* (opt) – include Z dof in constraint  
locglob\_constraint\_tol = *float* (opt) – tolerance  
locglob\_constraint\_mov\_tol = *float* (opt) – move tolerance

For crack face contact:

crack\_contact\_symmetric\_pair = *bool* (opt) – create symmetric pair  
crack\_contact\_add\_front\_nodes = *bool* (opt) – include crack front nodes

crack\_contact\_as\_target = *bool* (opt) –  
crack\_contact\_mat\_id = *int* (opt) – material ID  
crack\_contact\_mu = *float* (opt) – coefficient of friction  
crack\_contact\_real\_id = *int* (opt) – real constant ID  
crack\_contact\_real\_mod\_3 = *float* (opt) – real property  
crack\_contact\_real\_mod\_4 = *float* (opt) – real property  
crack\_contact\_et170\_id = *int* (opt) – element type 170 ID  
crack\_contact\_et174\_id = *int* (opt) – element type 174 ID  
crack\_contact\_keyopt\_2 = *int* (opt) – key option  
crack\_contact\_keyopt\_5 = *int* (opt) – key option  
crack\_contact\_keyopt\_8 = *int* (opt) – key option  
crack\_contact\_keyopt\_9 = *int* (opt) – key option  
crack\_contact\_keyopt\_10 = *int* (opt) – key option  
crack\_contact\_keyopt\_12 = *int* (opt) – key option

For extra connections:

contact\_connection = *string* (opt) – connection name  
contact\_symmetric\_pair = *bool* (opt) – create symmetric pair  
contact\_as\_target = *bool* (opt) –  
contact\_mat\_id = *int* (opt) – material ID  
contact\_mu = *float* (opt) – coefficient of friction  
contact\_real\_id = *int* (opt) – real constant ID  
contact\_real\_mod\_3 = *float* (opt) – real property  
contact\_real\_mod\_4 = *float* (opt) – real property



contact\_et170\_id = *int* (opt) – element type 170 ID  
contact\_et174\_id = *int* (opt) – element type 174 ID  
contact\_keyopt\_2 = *int* (opt) – key option  
contact\_keyopt\_5 = *int* (opt) – key option  
contact\_keyopt\_8 = *int* (opt) – key option  
contact\_keyopt\_9 = *int* (opt) – key option  
contact\_keyopt\_10 = *int* (opt) – key option  
contact\_keyopt\_12 = *int* (opt) – key option

constraint\_dofx = *bool* (opt) – include X dof in constraint  
constraint\_dofy = *bool* (opt) – include Y dof in constraint  
constraint\_dofz = *bool* (opt) – include Z dof in constraint  
constraint\_tol = *float* (opt) – tolerance  
constraint\_mov\_tol = *float* (opt) – move tolerance

The extra connections are defined as lists, same as for the ABAQUS extra connections.

### **2.1.1.7.3 NASTRAN Contact or Constraint**

For crack face contact:

crack\_contact\_BCTSetId = *int* (opt) – set ID  
crack\_contact\_BSurfMainId = *int* (opt) – main side surface ID  
crack\_contact\_BSurfMatId = *int* (opt) – mate side surface ID  
crack\_contact\_BCTSetFric = *float* (opt) – coefficient of friction  
crack\_contact\_BCTSetMinD = *float* (opt) – minimum distance  
crack\_contact\_BCTSetMaxD = *float* (opt) – maximum distance  
crack\_contact\_BCOffset = *float* (opt) – offset

### **2.1.2 CheckGrowthStatus( )**

Checks the current crack growth status and sets the growth status variable as GS\_NORMAL, GS\_CRITICAL, GS\_THRESHOLD or GS\_FAILED, which can be retrieved using GetGrowthStatus.

*parameters:*

file\_name = *single quoted string* (opt) – data file name  
step = *int* (opt) – crack growth step number

*example:*

CheckGrowthStatus (file\_name='C:\temp\gs.txt')

### 2.1.3 CloseModel()

Close the current model.

*example:*

```
CloseModel()
```

### 2.1.4 ComputeCOD()

Compute crack-front stress intensity factors.

*parameters:*

**distance** = *float* (req) - distance from the crack front

**at\_nodes** = *bool* (opt) - compute at nodes rather than geometric points (default = true)

*example:*

```
ComputeCOD(distance=0.1,  
at_nodes=false)
```

### 2.1.5 ComputeGrowthParams()

Compute the crack growth.

*parameters:*

**sif\_params** => see Section 2.1.1.1: sif\_params

**growth\_params** => see Section 2.1.1.2: growth\_params

**growth\_vars** => see Section 2.1.1.4: growth\_variables

*example:*

```
ComputeGrowthParams(sif_method=M_INTEGRAL,  
growth_type=QUASI_STATIC,  
median_step=0.01)
```

### 2.1.6 ComputeSif()

Compute crack-front stress intensity factors.

*parameters:*

**sif\_params** => see Section 2.1.1.1: sif\_params

*example:*

```
ComputeSif(sif_method=M_INTEGRAL,  
do_therm_terms=true)
```

### 2.1.7 CrackTractConst( )

Define constant crack-face traction.

*parameters:*

`index = int` (req) - crack traction index  
`press = float` (req) - constant pressure magnitude  
`load_case = int` (req) - traction load case  
`set_temp = int` (opt) – set temperature for traction load step  
`constant_temp = float` (opt) – constant temperature value for traction load step  
`loadstep_temp = int` (opt) – temperature from prior load step for traction load step

*example:*

```
CrackTractConst (index=1,  
  pressure=10.0,  
  load_case=1)
```

#### 2.1.7.1 CFT index and load case

All crack surface tractions (CFT) have an internal index. Indexes start at 0; each CFT has its own index. If CFTs are applied in their own load step; the first CFT load\_case id should start at a number that is one higher than the last FE model load step id.

### 2.1.8 CrackTractDelete( )

Delete a crack-face traction.

*parameters:*

`index = int` (req) - crack traction index

*example:*

```
CrackTractDelete (index=1)
```

### 2.1.9 CrackTractExternalDist( )

Define external distribution crack-face tractions.

*parameters:*

`model_type = string` (opt) – external mesh type  
`index = int` (req) - crack traction index  
`mesh_file = single quoted string` (req) - external mesh file name

*stress\_file* = *single quoted string* (req) - external stress file name  
*external\_step* = *int* (opt) - external load step ID  
*external\_substep* = *int* (opt) - external load substep ID  
*stress\_scale* = *float* (opt) - external stress factor  
*load\_case* = *int* (req) - traction load case  
*set\_temp* = *int* (opt) – set temperature for traction load step  
*constant\_temp* = *float* (opt) – constant temperature value for traction load step  
*loadstep\_temp* = *int* (opt) – temperature from prior load step for traction load step

*example:*

```

CrackTractExternalDist(index=1,
  mesh_file='my_dist.fem',
  stress_file='my_dist.str',
  external_step=1,
  load_case=1)
  
```

### 2.1.10 CrackTractSurface( )

Define surface treatment crack-face tractions.

*parameters:*

*index* = *int* (req) - crack traction index  
*dist* = *float array* (req) - distribution distance/traction pairs  
*load\_case* = *int* (req) - traction load case  
*file\_name* = *string* (opt) - text file with element face ids  
*set\_temp* = *int* (opt) – set temperature for traction load step  
*constant\_temp* = *float* (opt) – constant temperature value for traction load step  
*loadstep\_temp* = *int* (opt) – temperature from prior load step for traction load step

*example:*

```

CrackTractSurface(index=1,
  dist=[[0,10],[0.25,0]],
  load_case=2)
  
```

### 2.1.11 CrackTract1DRad( )

Define a 1D radial crack-face traction distribution.

*parameters:*

*index* = *int* (req) - crack traction index  
*axis* = *int* (req) - axis of rotation (x = 1, y = 2, z = 3)  
*offset* = *vec3d* (req) - axis offset from origin  
*dist* = *float array* (req) - distribution distance/traction pairs  
*load\_case* = *int* (req) - traction load case

`set_temp = int (opt)` – set temperature for traction load step  
`constant_temp = float (opt)` – constant temperature value for traction load step  
`loadstep_temp = int (opt)` – temperature from prior load step for traction load step

*example:*

```
CrackTract1DRad(index=1,  
axis=1,  
offset=[0,0,1],  
dist=[[0,10],[0.25,0]],  
load_case=2)
```

### 2.1.12 CrackTract2DRad( )

Define a 2D radial crack-face traction distribution.

*parameters:*

`index = int (req)` - crack traction index  
`axis = int (req)` - axis of rotation (x = 1, y = 2, z = 3)  
`axial = float array (req)` - list of axial locations  
`radial = float array (req)` - list of radial locations  
`dist = float array (req)` - table of traction values for all radial and axial locations  
`load_case = int (req)` - traction load case  
`set_temp = int (opt)` – set temperature for traction load step  
`constant_temp = float (opt)` – constant temperature value for traction load step  
`loadstep_temp = int (opt)` – temperature from prior load step for traction load step

*example:*

```
CrackTract2DRad(index=0,  
axis=1,  
axial=[-0.5,0.0,0.5],  
radial=[0.0,1.0],  
dist=[[0,1,0],[1,1.25,1]],  
load_case=2)
```

### 2.1.13 FretModelImport( )

Import fretting data model files.

*parameters:*

`model_type = string (req)` – model file type  
ABAQUS - ABAQUS .inp file  
ANSYS - ANSYS .cdb file  
`file_name = single quoted string (req)` – model file name  
`results_files = string array (req)` – list of results file names

`results_lcs` = *int array* (req) – list of load case ids in results  
`contact_pair` = *string array* (req) – contact pair name  
`results_option` = *int* (req) – read results-pair file or single .dtp file

*example:*

```
FretModelImport (model_type=ANSYS,  
file_name='C:\temp\uncracked.cdb',  
results_files=['C:\fretting test rig\fretting_rig_ls1.str',  
               'C:\fretting test rig\fretting_rig_ls2.str'],  
results_lcs=[0,1],  
contact_pair=[contact_6_6_contact_5_6],  
results_option=0)
```

### 2.1.14 FretNucleationCycles( )

Compute fretting nucleation cycles.

*parameters:*

`fretting_model_type` = *string* (req) – model file type  
FRET\_SEQ - equivalent stress model  
FRET\_GCRIT - critical shear stress model  
FRET\_SWT - Smith-Watson-Topper model  
FRET\_RUIZ – Ruiz-Chen model  
FRET\_MSMT – modified Smith-Watson-Topper (Fatemi-Socie) model  
`fretting_param_names` = *string array* (req) – fretting model parameters  
`fretting_param_vals` = *float array* (req) – fretting model parameters  
`do_averaging` = *string* (opt) – do volume averaging of stress / strain  
`do_add_residual` = *bool* (opt) – add residual stress  
`rdata` = *float array* (opt) – residual stress data  
`rfile_name` = *string* (opt) – residual stress data file name  
`save_file` = *string* (opt) – file name to save fretting cycles  
`max_load_step` = *int array* (req) – load step id for maximum load  
`min_load_step` = *int array* (req) – load step id for minimum load

*example:*

```
FretNucleationCycles (fretting_model_type=FRET_SEQ,  
fretting_param_names=[scale,ef,c,b,a,rotate,C,B,sf,E,d],  
fretting_param_vals=[0.001, 32.4, 7170., 0.0004, 1.125, 0, -0.77, 0.00018,  
                    376.6, 126100,-0.77],  
do_averaging=FRET_AVRG_NONE)
```

### 2.1.15 FretNucleationDataImport( )

Import fretting parameter versus cycle test data.

*parameters:*

fretting\_raw\_data\_file = *single quoted string* (req) – file name for raw fretting nucleation data

fretting\_function\_type = *int* (req) – function type for fitting raw data

POLY\_FIT=0; polynomial fit

LM\_FIT=1; nonlinear exponential fit

fretting\_function\_id = *int* (req) – function id

linear=0; polynomial fit

quadratic=1; polynomial fit

cubic=2; polynomial fit

power law=0; nonlinear exponential fit

power1 law=1; nonlinear exponential fit

power2 law=2; nonlinear exponential fit

exponential=3; nonlinear exponential fit

*example:*

```
FretNucleationDataImport (fretting_raw_data_file='fret_data.txt',  
fretting_function_type=1,  
fretting_function_id=2)
```

### 2.1.16 GetBuildInfo( )

Get build number and date. This information is written to the current playback log file. If there is no log file, the information is written to stdout.

*example:*

```
GetBuildInfo( )
```

### 2.1.17 GetCrackData( )

Get all the crack growth data; command in only available from Python. See Section 3.2.2.

### 2.1.18 GetIntegrationResults( )

Get the integration results; command in only available from Python. IntegrateStep() should be called before this. See Section 3.2.7.

### 2.1.19 GetGrowthStatus( )

Get the growth status; command is only available from Python.

It returns Normal, Threshold, Critical or Front\_Failed. See Section 3.2.

### 2.1.20 GrowCrack( )

Grow crack front(s).

*parameters:*

sif\_params => see Section 2.1.1.1: sif\_params  
growth\_vars => see Section 2.1.1.3: growth\_variables  
template\_params => see Section 2.1.1.5: template\_params  
fit\_params => see Section 2.1.1.6: front\_fitting\_params  
file\_name = *string (opt)* - file name  
check\_fit = *bool (opt)* – true if front fitting and extension automatically adjusted

*example:*

```
GrowCrack(sif_method=M_INTEGRAL,  
          median_step=0.1,  
          temp_radius_type=ABSOLUTE,  
          temp_radius=0.05,  
          discard=[[0,0]],  
          extrapolate=[[3,3]])
```

### 2.1.21 GrowCrackFromFile( )

Grow crack from a file.

*parameters:*

read\_file = *single quoted string (req)* - file name of new front points  
template\_params => see Section 2.1.1.5: template\_params  
fit\_params => see Section 2.1.1.6: front\_fitting\_params  
file\_name = *single quoted string (opt)* - file name

*example:*

```
GrowCrackFromFile(temp_radius_type=ABSOLUTE,  
                 temp_radius=0.05,  
                 read_file='abaqus_cube/small_step.frt'))
```

### 2.1.22 GrowMergeCrack ( )

Grow and merge coplanar crack fronts.

*parameters:*

sif\_params => see Section 2.1.1.1: sif\_params  
growth\_vars => see Section 2.1.1.3: growth\_variables



template\_params => see Section 2.1.1.5: template\_params  
 fit\_params => see Section 2.1.1.6: front\_fitting\_params  
 file\_name = *string (opt)* - file name  
 fit\_points = *vec3d array (opt)* – list of new fitted crack front points  
 old\_points = *vec3d array (opt)* – list of current crack front points  
 norms = *vec3d array (opt)* – list of crack front normals  
 frt\_idx = *int array (opt)* – list of crack front ids  
 frt\_ord = *bool array (opt)* – list of crack front ordering

*example:*

```

GrowMergeCrack(sif_method=M_INTEGRAL,
  median_step=0.1,
  temp_radius_type=ABSOLUTE,
  temp_radius=0.05,
  fit_points=[[1,0,0],[2,0,0],[3,0,0]],
  old_points=[[.1,0,0],[.2,0,0],[.3,0,0]],
  norms=[[0,0,1],[0,0,1],[0,0,1]],
  frt_idx=[1,0],
  frt_ord[true,true])
  
```

### 2.1.23 Include( )

Include a file; only available in NO\_GUI mode.

*parameters:*

flags = *string array (opt)* – INTERACTIVE, NATIVE, ABAQUS, ANSYS, NASTRAN  
 file\_name = *single quoted string (req)* - file name

### 2.1.24 InsertFileFlaw( )

Insert a flaw from a .crk file. Note that flaw orientation and template options are defined in .crk files. Parameters for this command, if specified, will override the values in the file.

*parameters:*

flaw\_insert\_params => see Section 2.1.24.1  
 file\_name = *single quoted string (req)* - name of a flaw (.crk) definition file

*example:*

```

InsertFileFlaw(
  file_name='my_flaw.crk',
  translation=[0,0.1,0],
  radius=0.025)
  
```

### 2.1.24.1 *flaw\_insert\_params*

*parameters:*

*rotation\_axes* = *int array* (opt) - ordered list of up to three rotation axes (x = 1, y = 2, z = 3)  
*rotation\_mag* = *float array* (opt) - ordered list of up to three rotation magnitudes  
*translation* = *vec3d* (opt) - translation vector  
*local\_x\_axis* = *vec3d* (opt) – unit x-vector to define rotation  
*local\_y\_axis* = *vec3d* (opt) - unit y-vector to define rotation  
*refinement\_level* = *int* (opt) - number of times the flaw patches will be recursively subdivided  
*do\_template* = *bool* (opt) – true if template used (default = true)  
*radius* = *float* (opt) - crack-front template radius  
*progression\_ratio* = *float* (opt) - crack-front template element size progression ratio (default = 1)  
*num\_rings* = *int* (opt) - number of element rings in the crack-front template (default = 3)  
*num\_circ* = *int* (opt) - number of elements inserted circumferentially about a crack-front (default = 8)  
*max\_aspect\_ratio* = *float* (opt) - maximum allowable aspect ratio for crack-front elements (default = 3.0)  
*simple\_ints\_only* = *bool* (opt) - if true, crack-front templates terminate within the body (default = false)  
*ref\_node* = *int* (opt) – node ID for translation  
*bi\_material\_flag* = *bool* (opt) – indicates crack is in bi-material interface  
*bi\_material\_angle* = *float* (opt) – angle where crack pops out of interface  
*symmetry\_flag* = *bool* (opt) – indicates crack is on a symmetry surface  
*symmetry\_surf\_name* = *string* (opt) – symmetry surface name  
*symmetry\_surf\_type* = *string* (opt) – symmetry surface type  
“NODE\_SET or “SURF\_SET”

### 2.1.25 **InsertMultFileFlaw()**

Insert multiple flaws from .crk files. The .crk files are read and combined into a single .crk file that is then inserted using the `InsertFileFlaw()` command.

*parameters:*

*file\_names* = *quoted string array* (req) - names of a flaw (.crk) definition files

*example:*

```
InsertMultFileFlaw(  
    file_names=['crack_1.crk', 'crack_2.crk'])
```

### 2.1.26 InsertMultParamFlaw( )

Insert multiple parameterized flaws.

*parameters:*

flaw\_type = *string array* (req) - list of flaw types:  
    CRACK - zero volume crack  
    VOID - finite volume void  
crack\_type = *string array* (req if flaw\_type=CRACK) - list of crack types:  
    ELLIPSE - elliptical crack  
    THRU - through the thickness crack  
    CENTER - center crack  
    LONG - long shallow crack  
    NONE - place holder for void types  
void\_type = *string array* (req if flaw\_type=VOID) - list of void types:  
    ELLIPSOID - ellipsoidal flaw  
    NONE - place holder for crack types  
flaw\_params = *float array* (req) - list of lists of flaw size parameters  
flaw\_insert\_params *array* => see Section 2.1.24.1

*example:*

```
InsertMultParamFlaw(  
    flaw_type=[CRACK, CRACK],  
    crack_type=[EMESH, EMESH],  
    flaw_params=[[0.1, 0.1], [0.15, 0.15]],  
    rotation_axes=[[1], [1]],  
    rotation_mag=[[90], [90]],  
    translation=[[0, 0.1, 1], [0, -0.15, 1]],  
    radius=[0.02, 0.03])
```

### 2.1.27 InsertParamFlaw( )

Insert a parameterized flaw.

*parameters:*

flaw\_type = *string* (req) - flaw type:  
    CRACK - zero volume crack  
    VOID - finite volume void  
crack\_type = *string* (req if flaw\_type=CRACK) - crack types:  
    EMESH - elliptical crack  
    THRU - through the thickness crack  
    CENTER - center crack

LONG - long shallow crack  
 RING - double front ring crack  
 USER - user-defined boundary points crack  
 UMESH - user-defined mesh crack  
 void\_type = *string* (req if flaw\_type=VOID) - void types:  
     ELLIPSOID - ellipsoidal flaw  
 flaw\_params = *float array* (req) - list of flaw size parameters  
 flaw\_insert\_params => see Section 2.1.24.1

*example:*

```

InsertParamFlaw(
  flaw_type=CRACK,
  crack_type=EMESH,
  flaw_params=[0.15,0.15],
  rotation_axes=[1],
  rotation_mag=[90],
  translation=[0,-0.15,1],
  radius=0.03)
  
```

### 2.1.28 InsertUserBdryFlaw()

Insert a user-defined-boundary flaw.

*parameters:*

points = *Vec3D array* (opt) – boundary points  
 front\_flags = *int array* (opt) – boundary point front flag  
 file\_name = *quoted string* (opt) – file with boundary points  
 num\_smooth = *int* (opt) – number of points if smoothing and reparametrizing  
 flaw\_insert\_params => see Section 2.1.24.1

*example:*

```

InsertUserBdryFlaw(
  points=[[3.642,5.0,10.054],\+
          [3.645,5.0,9.984],\+
          [3.650,5.0,9.910],\+
          [3.9,5.0,10.14]],
  front_flags=[1,1,1,0],
  radius=0.078)
  
```

### 2.1.29 InsertUserMeshFlaw()

Insert a user-defined-mesh flaw.

*parameters:*

`flaw_type` = *string array* (opt) – default is CRACK  
`mesh_file` = *quoted string* (req) - name of a surface mesh file  
`front_groups` = *string array* (opt) – crack front node sets in mesh file  
`front_points` = *Vec3D array* (opt) – crack front points  
`scaling` = *float* (opt) – scale factor  
`do_reverse` = *bool* (opt) – reverse surface mesh normal if true  
(only required for symmetry cracks)  
`flaw_insert_params` => see Section 2.1.24.1

*example:*

```
InsertUserMeshFlaw(  
    flaw_type=CRACK,  
    mesh_file='surf_mesh_half_penny.cdb',  
    front_groups=[CRACK_FRONT],  
    rotation_axes=[1],  
    rotation_mag=[-90],  
    translation=[4, 5, 10.05],  
    radius=0.06)
```

### 2.1.30 IntegrateStep()

Integrate cycles for a step of crack growth and sets the growth status variable as GS\_NORMAL, GS\_CRITICAL or GS\_THRESHOLD, which can be retrieved using GetGrowthStatus().

*parameters:*

`step` = *int* (opt) – step of crack growth

*example:*

```
IntegrateStep()
```

### 2.1.31 OpenFdbModel()

Open a FRANC3D database (.fdb) file.

*parameters:*

`file_name` = *single quoted string* (req) - .fdb file name

*orig\_mesh\_file* = *single quoted string* (opt) - name of original, uncracked, mesh model  
*extra\_file* = *single quoted string array* (opt) - list of names of extra (load case) files  
*mesh\_file* = *single quoted string* (req) - name of current crack step mesh model  
*resp\_file* = *single quoted string* (opt) - name of current crack step FEM results  
*global\_file* = *single quoted string* (opt) - name of original global mesh portion

*example:*

```
OpenFdbModel (  
    file_name='my_cracked_model.fdb' ,  
    orig_mesh_file='uncracked_mesh.cdb' ,  
    mesh_file='my_cracked_model.cdb' ,  
    resp_file='my_cracked_model.dtp' )
```

### 2.1.32 OpenMeshModel( )

Open an FEM mesh file.

*parameters:*

*model\_type* = *string* (req) - type of file to read:  
    ABAQUS - ABAQUS .inp file  
    ANSYS - ANSYS .cdb file  
    NASTRAN - NASTRAN .bdf file  
*file\_name* = *single quoted string* (req) - mesh file name  
*global\_name* = *single quoted string* (opt) - mesh file name  
*extra\_files* = *single quoted string array* (opt) - list of names of extra load case files  
*retained\_nodes* = *int array* (opt) – list of retained node ids  
*retained\_nodes\_file* = *single quoted string* (opt) – retained nodes list file name  
*retain\_old\_fronts* = *bool* (opt) – retain crack front nodes from step to step  
*ansys\_exe* = *single quoted string* (opt) – ANSYS executable  
*ansys\_lic* = *string* (opt) – ANSYS license type string  
*retain\_auto\_cut\_surf* = *string* (opt) – retain the cut-surface nodes and facets

*example:*

```
OpenMeshModel (  
    model_type=ANSYS ,  
    file_name='my_mesh.cdb' )
```

### 2.1.33 ReadFullGrowthHist ( )

Read a full crack growth history file.

*parameters:*

`file_name = single quoted string (req) – crack growth history file name`

*example:*

```
ReadFullGrowthHist (file_name='history.fcg')
```

### **2.1.34 ReadGrowthParams ( )**

Read a crack growth parameters file; this command is deprecated.  
The SetGrowthParams command should be used instead.

*parameters:*

`file_name = single quoted string (req) – crack growth parameters file name`

*example:*

```
ReadGrowthParams (file_name='my.cgp')
```

### **2.1.35 ReadResponse( )**

Read a response file.

*parameters:*

`file_name = single quoted string (req) - response file name`

*example:*

```
ReadResponse (  
  file_name='my_results.dtp')
```

### **2.1.36 RunAnalysis( )**

Run an analysis

*parameters:*

`model_type = string (req) – model type: ABAQUS, ANSYS or NASTRAN`  
`file_name = single quoted string (req) - response file name`  
`general_analysis_options (req) – see Section 2.1.1.7`

*example:*

```
RunAnalysis(model_type="ABAQUS",
  file_name='abaqus_cube/my_crack_model',
  flags=[TRANSFER_BC,NO_CFACE_TRACT,NO_CFACE_CNTCT],
  merge_tol=0.0001,
  executable='abaqus',
  command='abaqus job=junk_full -interactive -analysis ',
  global_model='abaqus_cube/my_cube_global.inp',
  merge_surf_labels=[CUT_SURF],
  global_surf_labels=[C_SURF])
```

### 2.1.37 SaveFdbModel()

Save a FRANC3D database (.fdb) file.

*parameters:*

`file_name` = *single quoted string* (req) - .fdb file name  
`mesh_file_name` = *single quoted string* (req) - file name for the mesh model;  
this is the local cracked portion mesh file  
`rslt_file_name` = *single quoted string* (req) - file name for the FEM results  
this is the cracked model results  
`global_file_name` = *single quoted string* (opt) - file name for the global mesh  
`analysis_code` = *string* (req) - analysis code:  
ABAQUS, ANSYS, or NASTRAN  
`flags` = *string array* (opt) - list of analysis option flags (see Section 2.1.1.7)

*example:*

```
SaveFdbModel(
  file_name='my_model.fdb',
  mesh_file_name='my_model.cdb',
  rslt_file_name='my_model.dtp',
  analysis_code=ANSYS)
```

### 2.1.38 SaveGrowthParams()

Save a crack growth parameters file.

*parameters:*

`file_name` = *single quoted string* (req) – crack growth parameters file name



*example:*

```
SaveGrowthParams (file_name='my.cgp')
```

### 2.1.39 SaveMeshModel( )

Save a FEM mesh file.

*parameters:*

`file_name` = *single quoted string* (req) - mesh file name  
`model_type` = *string* (req) - analysis code:  
ABAQUS, ANSYS, or NASTRAN  
`flags` = *string array* (opt) - list of analysis option flags (see Section 2.1.1.7)

*example:*

```
SaveMeshModel(  
    file_name='my_model.cdb',  
    model_type=ANSYS)
```

### 2.1.40 SetEdgeParameters( )

Set parameters for edge extraction.

*parameters:*

`kink_angle` = *float* (opt) - kink edge angle threshold  
`do_planar_seed` = *bool* (opt) - use a seed growth algorithm to find planar regions  
`planar_angle` = *float* (opt) - angle threshold for planar regions  
`planar_facets` = *int* (opt) - minimum number of facets in planar regions  
`retain_lines` = *Vec3D array* (opt) – end points of geometry lines to retain  
`retain_infinite_extent` = *bool array* (opt) – flags indicate geometry lines infinitely long  
`retain_tolerance` = *float array* (opt) – tolerance for each line

*example:*

```
SetEdgeParameters(  
    do_planar_seed=True,  
    planar_angle=178,  
    planar_facets=5)
```

### 2.1.41 SetGrowthParams()

Set parameters for crack growth. Set the growth parameters or specify a file name. The file\_name will override the growth\_params if both are included.

*parameters:*

growth\_params => see Section 2.1.1.2  
growth\_vars => see Section 2.1.1.4  
file\_name => *single quoted string* (opt) – crack growth parameters file name

*example:*

```
SetGrowthParams (file_name='my.cgp')

SetGrowthParams (
  growth_type=QUASI_STATIC,
  load_step_map=["VERSION: 1", "MAX_SUB: 2", "0 0",
                "0 0", "LABELS: 2", "2 Load Step 2",
                "1 Load Step 1"],
  kink_angle_strategy=MTS,
  quasi_static_n=2,
  quasi_static_loads=["NUM_STEPS: 2", "1 FINAL 1 1 0",
                     "2 FINAL 1 1 0"])
```

### 2.1.42 SetLoadSchedule()

Set a load schedule from a file: DARWIN mission format file.

*parameters:*

file\_name = *single quoted string* (req) – load schedule file name

*example:*

```
SetLoadSchedule (file_name='my.txt')
```

### 2.1.43 SetMeshingParameters()

Modify meshing parameters.

*parameters:*

max\_gen\_elems = *int* (opt) - maximum number of elements that will be generated  
max\_vol\_restarts = *int* (opt) - maximum number of volume meshing restarts

do\_coarsen\_crack\_mouth = *bool* (opt) - coarsen crack mouth flag  
do\_crack\_proximity\_refinement = *bool* (opt) - surface refinement flag  
do\_not\_coarsen\_uncracked = *bool* (opt) – do not coarsen surface flag

volume\_meshing\_method = *string* (opt) - meshing code:

FRANC3D, ABAQUS, or ANSYS

ansys\_executable = *string* (opt) - ANSYS executable for meshing

ansys\_license = *string* (opt) - ANSYS license type

abaqus\_executable = *string* (opt) - ABAQUS executable for meshing

surf\_max\_internal\_cel\_ratio = *float* (opt) - Ratio of the maximum element size allowed in the interior of a surface mesh to the maximum element size on the boundary.

surf\_mesh\_density\_decay\_ratio = *float* (opt) - Nominally the maximum size ratio between two adjacent elements in a surface mesh.

surf\_curvature\_refinement\_factor = *float* (opt) - If *r* is the local minimum principal radius of curvature, then the local maximum ideal element size will be *r* \* SurfCurvatureRefineFactor, or this is the maximum secant angle an element will span for 'r'.

surf\_curvature\_refinement\_limit = *float* (opt) - The maximum ratio between the nominal local element size and a reduced size set due to local surface curvature.

surf\_crack\_front\_decay\_ratio = *float* (opt) - The ratio at which adjacent element sizes can increase as one moves from a crack front to a nearby surface.

optimal\_sphere\_factor = *float* (opt) - Controls the size of the spherical region that the volume mesher uses to look for existing nodes on the advancing front.

optimal\_size\_factor = *float* (opt) - Factor applied to the background oct-tree cell size to determine the local optimal element size.

volume\_refine\_factor = *float* (opt) - Factor applied to control local oct-tree refinement.

*example:*

```
SetMeshingParameters (  
    do_coarsen_crack_mouth=True,  
    max_vol_restarts=10)
```

#### 2.1.44 SetStatusFile( )

Set a status file. File records success or error status.

*parameters:*

file\_name = *single quoted string* (req) – status file name

*example:*

```
SetStatusFile (file_name='my_status.txt')
```

### 2.1.45 SetUnits( )

Specify the FEM units.

*parameters:*

model\_length = *string* (opt) – unit for model length: MM, M, INCH  
model\_stress = *string* (opt) – unit for model stress (modulus): MPA, PA, PSI, KSI  
temperature = *string* (opt) – unit for model temperature: C, F  
DARWIN\_units = *string* (opt) – unit for DARWIN: US, SI

*example:*

```
SetUnits(  
    model_length=INCH,  
    model_stress=PSI,  
    temperature=F)
```

### 2.1.46 SetUserExtensionsFile( )

Set user defined Python extensions file.

*parameters:*

file\_name = *single quoted string* (req) – Python extensions file name  
flags = *string array* (opt) – flags:  
    USER\_INITIALIZE,  
    USER\_NEW\_POINT,  
    USER\_KINK\_ANG,  
    USER\_CYCLES\_RATE,  
    USER\_TIME\_RATE,  
    USER\_START\_STEP,  
    USER\_END\_STEP

*example:*

```
SetUserExtensionsFile (file_name='my_user_growth.py',  
    flags=[USER_INITIALIZE,USER_NEW_POINT,USER_KINK_ANG])
```

### 2.1.47 SetWorkingDirectory( )

Set the work directory. The path separator for Windows and Linux are different.

*parameters:*

`directory = single quoted string (req) – folder path`

*example:*

```
SetWorkingDirectory (directory='/home/work')
SetWorkingDirectory (directory='C:\user\ansys_models')
```

## 2.1.48 SifHistory()

Compute the SIF history.

*parameters:*

`flags = string array (opt) – SIF history flags: PLANE, INTERACTIVE`

`path_type = string (opt) – SIF history path type`

`const_n_dist = float (opt) – normalized distance to compute SIFs`

`near_n_dist = float (opt) -`

`do_smooth_Ks = bool (opt) -`

`Ks_poly_order = int (opt) -`

`do_least_squares = bool (opt) -`

`poly_order = int (opt) -`

`min_n_dist = float (opt) -`

`max_n_dist = float (opt) -`

`min_smooth_n_dist = float (opt) -`

`max_smooth_n_dist = float (opt) -`

`plane_normal = Vec3D (opt) -`

`plane_point = Vec3D (opt) -`

`start_type = string (opt) – set start type (point or length)`

`start_point = Vec3D (opt) – set starting crack point (origin)`

`start_length = float (opt) – set starting crack length`

`start_step = int (opt) – set starting crack step id`

`start_front = int (opt) – set starting crack front id`

`growth_data_file = string (opt) – crack growth data file name`

`sif_files = string array (opt) – list of SIF history files`

*example:*

```
SifHistory(start_front=1)
```

### 2.1.49 *StartRecording()*

Starts recording the commands to a log file; this command is only available from the Python interface.

### 2.1.50 *Submodeler()*

Divide the FE model into local and global portions.

*parameters:*

flags = *string array* (opt) – submodeler flags:  
INTERACTIVE,  
NATIVE,  
ABAQUS,  
ANSYS,  
GLOBAL\_ONLY,  
SUBMODEL\_ONLY,  
DARWIN

model\_type = *string* (req) – model type: ABAQUS, ANSYS or NASTRAN  
orig\_file\_name = *single quoted string* (req) – input FE model  
elem\_file\_name = *single quoted string* (req) – element id list file  
submodel\_file\_name = *single quoted string* (opt) – local submodel file name  
global\_file\_name = *single quoted string* (opt) – global portion file name  
extra\_files = *single quoted string list* (opt) – extra load case file names  
elem\_groups = *string list* (opt) – list of element groups

*example:*

```
Submodeler(  
    model_type=ABAQUS,  
    orig_file_name='Abaqus-Cube.inp',  
    submodel_file_name='Abaqus-Cube_LOCAL.inp',  
    global_file_name='Abaqus-Cube_GLOBAL.inp',  
    elem_file_name='Abaqus-Cube_RETAINED_ELEMS.txt',  
    elem_groups=["local_elems_a", "local_elems_b"])
```

### 2.1.51 *WriteCOD()*

Generate a file containing crack-front crack opening displacement data. This must be preceded by *ComputeCOD*.

*parameters:*

*file\_name* = *single quoted string* (req) - output SIF file name  
*crack\_step* = *int* (opt) - ID (index) of the crack step  
*front\_id* = *int* (opt) - ID (index) of the crack front (0 .. n-1), default = 0  
*load\_step* = *int* (opt) - ID (index) of the load step  
*load\_substep* = *int* (opt) - ID (index) of the load substep  
*flags* = *string array* (opt) - array of output options:
 

- SPACE - use spaces as delimiters
- TAB - use tabs as delimiters (default)
- COMMA - use commas as delimiters
- COD - include crack opening displacements
- CSD - include crack sliding displacements
- CTD - include crack tearing displacements
- KI - include mode one stress intensity factors
- KII - include mode two stress intensity factors
- KIII - include mode three stress intensity factors
- CRD - include crack-front Cartesian coordinates
- DCCRD - include COD evaluation point coordinates
- AXES - include crack-front local coordinate axes
- MODULI - include the elastic modulus at the evaluation points

*example:*

```

WriteCOD(

file_name='sif_data.sif', flags=[TAB, COD, CSD, CTD, CRD, DCCRD])

```

### 2.1.52 WriteCrackData( )

Generate a file containing predicted crack growth data.

*parameters:*

*file\_name* = *single quoted string* (req) - output file name

*example:*

```

WriteCrackData(
    file_name='crack_info.dat')

```

### 2.1.53 WriteEPJ( )

Generate a file containing elasto-plastic J-integral data.

*parameters:*

`file_name` = *single quoted string* (req) - output file name

*example:*

```
WriteEPJ(  
    file_name='epj_info.dat')
```

### 2.1.54 WriteFatigueData()

Generate a file containing fatigue data.

*parameters:*

`file_name` = *single quoted string* (req) - output file name  
`growth_params` = (req) => see Section 2.1.1.2  
`step` = *int* (opt) - ID (index) of the crack step  
`front` = *int* (opt) - ID (index) of the crack front (0 .. n-1), default = 0  
`point` = *int* (opt) - ID (index) of the load step  
`flags` = *string array* (opt) - array of output options:  
    SPACE - use spaces as delimiters  
    TAB - use tabs as delimiters (default)  
    COMMA - use commas as delimiters  
    CYCLES - include cycles  
    TIME - include time  
    SIFS - include SIFs  
    SEQUENCE - include load sequence  
    STEPS - include steps  
    SURFACE - include surface data  
    PATH - include path data  
    KMAX - include Kmax  
    KMIN - include Kmin  
    DK - include delta K

*example:*

```
WriteFatigueData(  
    file_name='ftg_data.fcg'  
    flags=[TAB,CYCLES],  
    growth_type="SUBCRITICAL",  
    fem_units="MM|MPA|C|SEC",  
    load_schedule_data=["VERSION: 1",...],  
    growth_model=["VERSION: 2",...],  
    load_step_map=["VERSION: 1",...],
```



```
kink_angle_strategy="MTS")
```

### 2.1.55 WriteGrowthData()

Generate a file containing crack-growth data.

*parameters:*

`file_name` = *single quoted string* (req) - output file name  
`growth_params` = (req) => see Section 2.1.1.2  
`step` = *int* (opt) - ID (index) of the crack step  
`front` = *int* (opt) - ID (index) of the crack front (0 .. n-1), default = 0  
`event` = *int* (opt) - ID (index) of the load event (1 .. n)  
`flags` = *string array* (opt) - array of output options:  
    SPACE - use spaces as delimiters  
    TAB - use tabs as delimiters (default)  
    COMMA - use commas as delimiters  
    KMAX - include Kmax  
    KMIN - include Kmin  
    DK - include delta K  
    KEQ - include Kequiv  
    KSTAT - include Kstatic  
    KHOLD - include Khold  
    R - include R (ratio)  
    TEMP - include Temperature  
    CRD - include front point coordinates  
    AXES - include front local axes  
    REVERSE - output in reverse order

*example:*

```
WriteGrowthData(  
  file_name='growth_data.fcg'  
  flags=[TAB,KMAX],  
  growth_type="SUBCRITICAL",  
  fem_units="MM|MPA|C|SEC",  
  load_schedule_data=["VERSION: 1",...],  
  growth_model=["VERSION: 2",...],  
  load_step_map=["VERSION: 1",...],  
  kink_angle_strategy="MTS")
```

### 2.1.56 WriteGrowthParams()

Generate a file containing crack-growth parameters.

*parameters:*

`file_name` = *single quoted string* (req) - output file name  
`step` = *int* (opt) - ID (index) of the crack step  
`front_id` = *int* (opt) - crack front id (0 .. n-1), default = 0  
`flags` = *string array* (opt) - array of output options:  
    SPACE - use spaces as delimiters  
    TAB - use tabs as delimiters (default)  
    COMMA - use commas as delimiters  
    ANGLE - include kink angle  
    EXT - include extension  
    CRD - include front point coordinates  
    AXES - include front local axes  
    DIR - include growth direction  
    REVERSE - output in reverse order

*example:*

```
WriteGrowthParams (  
    file_name='growth_params.dat',  
    step=3,  
    front_id=0,  
    flags=[TAB,EXT])
```

### 2.1.57 WriteResolvedSif()

Generate a file containing resolved stress intensity factors.

*parameters:*

`file_name` = *single quoted string* (req) - output SIF file name  
`crack_step` = *int* (opt) - ID (index) of the crack step  
`front_id` = *int* (opt) - ID (index) of the crack front (0 .. n-1), default = 0  
`load_step` = *int* (opt) - ID (index) of the load step  
`load_substep` = *int* (opt) - ID (index) of the load substep  
`flags` = *string array* (opt) - array of output options:  
    SPACE - use spaces as delimiters  
    TAB - use tabs as delimiters (default)  
    COMMA - use commas as delimiters  
    KRSS - include Krss  
    KRNS - include Krns  
    RRAT - include R ratio  
    SPPP - include slip plane 111  
    SPNN - include slip plane 1-1-1

SNPN - include slip plane -11-1  
 SNNP - include slip plane -1-11  
 SSMAX - include slip plane max  
 MAX - include max slip  
 FWD - include forward slip  
 REV - include reverse slip  
 CRD - include crack-front Cartesian coordinates  
 AXES - include crack-front local coordinate axes  
 REVERSE - reverse the ordering of the values

*example:*

```

WriteResolvedSif(
  file_name=sif_data.sif
  crack_step=4,
  load_step=2,
  flags=[TAB, KRSS])
  
```

### **2.1.58 WriteResolvedSifPath()**

Generate a file containing resolved stress intensity factors along a path.

*parameters:*

file\_name = *single quoted string* (req) - output SIF file name  
 front\_id = *int* (opt) - ID (index) of the crack front (0 .. n-1), default = 0  
 load\_step = *int* (opt) - ID (index) of the load step  
 load\_substep = *int* (opt) - ID (index) of the load substep  
 path\_type = *string* (opt) - path type  
 flags = see flags in 2.1.57

*example:*

```

WriteResolvedSifPath(
  file_name=sif_path.sif
  load_step=3,
  path_type=CLOSEST,
  flags=[TAB, XYYY, KI, KII, KIII])
  
```

### **2.1.59 WriteSERR()**

Generate a file containing strain energy release rate data. This must be preceded by a ComputeSif call using a method that will compute G; *i.e.*, VCCT.

*parameters:*

`file_name` = *single quoted string* (req) - output SIF file name  
`crack_step` = *int* (opt) - ID (index) of the crack step  
`front_id` = *int* (opt) - ID (index) of the crack front (0 .. n-1), default = 0  
`load_step` = *int* (opt) - ID (index) of the load step  
`load_substep` = *int* (opt) - ID (index) of the load substep  
`flags` = *string array* (opt) - array of output options:  
    SPACE - use spaces as delimiters  
    TAB - use tabs as delimiters (default)  
    COMMA - use commas as delimiters

*example:*

```
WriteSERR(  
  file_name=sif_data.sif  
  flags=[TAB,XYYY,GI])
```

## 2.1.60 WriteSif()

Generate a file containing crack-front SIF data.

*parameters:*

`file_name` = *single quoted string* (req) - output SIF file name  
`crack_step` = *int* (opt) - ID (index) of the crack step  
`front_id` = *int* (opt) - ID (index) of the crack front (0 .. n-1), default = 0  
`load_step` = *int* (opt) - ID (index) of the load step  
`load_substep` = *int* (opt) - ID (index) of the load substep  
`sif_params` => see Section 2.1.1.1: `sif_params`  
`flags` = *string array* (opt) - array of output options:  
    SPACE - use spaces as delimiters  
    TAB - use tabs as delimiters (default)  
    COMMA - use commas as delimiters  
    KI - include mode one stress intensity factors  
    KII - include mode two stress intensity factors  
    KIII - include mode three stress intensity factors  
    J - include J-integral values  
    T - include T-stress values  
    TEMP - include temperature values  
    CRD - include crack-front Cartesian coordinates  
    AXES - include crack-front local coordinate axes  
    REVERSE - reverse the ordering of the values  
    GI - include mode one energy release rate  
    GII - include mode two energy release rate

GIII - include mode three energy release rate

*example:*

```
WriteSif(  
  file_name=sif_data.sif  
  crack_step=4,  
  load_step=2,  
  flags=[TAB,KI,KII,KIII,CRD],  
  do_therm_terms=true,  
  do_press_terms=true)
```

### 2.1.61 WriteSifPath()

Generate a file containing SIF path history data.

*parameters:*

file\_name = *single quoted string* (req) - output SIF file name  
front\_id = *int* (opt) - ID (index) of the crack front (0 .. n-1), default = 0  
load\_step = *int* (opt) - ID (index) of the load step  
load\_substep = *int* (opt) - ID (index) of the load substep  
path\_type = *string* (opt) - path type  
sif\_params => see Section 2.1.1.1: sif\_params  
flags = see flags in 2.1.60

*example:*

```
WriteSifPath(  
  file_name=sif_path.sif  
  load_step=3,  
  path_type=CLOSEST,  
  flags=[TAB,XYYY,KI,KII,KIII])
```

### 2.1.62 WriteStdTempData()

Generate a file containing the ID's and coordinates of nodes in the crack-front template.

*parameters:*

file\_name = *string* (req) - output file name

*example:*

```
WriteStdTempData(  
    file_name=template_info.dat)
```

## 2.2 Example Command File

Command files are typically given a .f3d or .log extension. They can be used within the GUI using the **File - Playback** menu option. They can also be used in batch mode from the command line as in:

```
C:\f3d\franc3d.exe -batch input_commands.f3d
```

A command file might look like this:

```
-----  
# FRANC3D Version 8.0  
  
SetWorkingDirectory(  
    directory='C:\Abaqus_base')  
  
OpenMeshModel(  
    model_type=ABAQUS,  
    file_name='Abaqus-Cube.inp',  
    retained_nodes_file='Abaqus-Cube_RETAINED.txt')  
  
SetUnits(  
    model_length=MM,  
    model_stress=MPA,  
    temperature=C)  
  
InsertFileFlaw(  
    file_name='Cube_Crack.crk')  
  
RunAnalysis(  
    model_type=ABAQUS,  
    file_name='Abaqus_cube_crack.fdb',  
    flags=[TRANSFER_BC,NO_CFACE_TRACT,NO_CFACE_CNTCT],  
    connection_type=MERGE,  
    command=""abaqus.bat" job=Abaqus_cube_crack ask_delete=NO -interactive -analysis ")  
  
ComputeSif()  
  
SetGrowthParams(  
    growth_type=SUBCRITICAL,  
    fem_units="MM|MPA|C|SEC",  
    load_schedule_data=[\+
```

```

"VERSION: 1",\,+
"SCHEDULE (" ,\,+
"REPEAT_COUNT: FOREVER",\,+
"NUM_CHILDREN: 1",\,+
"SIMPLE_CYCLIC (" ,\,+
"REPEAT_COUNT: 1",\,+
"R: 0",\,+
"KMAX:",\,+
"ONE_STEP: 1 0 1 1 0",\,+
")",\,+
")",
growth_model=[\,+
"VERSION: 2",\,+
"NUM_MODELS: 1",\,+
"CYCLES_RATE_TYPE: PARIS",\,+
"CYCLES_RATIO_TYPE: NONE",\,+
"CYCLES_TEMP_INTERP: TEMP_NONE",\,+
"CYCLES_TITLE: ",\,+
"CYCLES_DESCRIPTION: 0",\,+
"CYCLES_PROP_UNITS: MM|MPA|C|SEC",\,+
"C: 1e-010 n: 3 DKth: 0.1 Kc: 100 ",\,+
"TIME_RATE_TYPE: NONE"],
load_step_map=["VERSION: 1","MAX_SUB: 1","0 0","LABELS: 1","1 Load Step 1"],
kink_angle_strategy=MTS)

```

```

GrowCrack(
  median_step=0.1,
  cycles_step=1000,
  front_mult=[1],
  temp_radius_type=RELATIVE,
  temp_radius=65,
  extrapolate=[[3,3]])

```

```

AutoGrowth(
  model_type=ABAQUS,
  cur_step=1,
  file_name='Abaqus_cube_crack',
  num_steps=5,
  step_type=SCONST,
  const_step_size=0.1,
  check_Kc=true,
  check_DKth=true,
  maximum_steps=5,
  temp_radius_type=RELATIVE,
  temp_radius=65,
  extrapolate=[[3,3]],

```

```
flags=[TRANSFER_BC,NO_CFACE_TRACT,NO_CFACE_CNTCT],  
connection_type=MERGE,  
command="abaqus.bat" job=Abaqus_cube_crack_STEP_001 ask_delete=NO -interactive -  
analysis ')
```

```
WriteSifPath(  
file_name='k_vs_a.sif',  
load_step=1,  
flags=[TAB,A,KI,KII,KIII,CRD])
```



## 3. Python Module

The Python module has extensions that mirror the commands described in Section 2. It allows the user to write more elaborate Python scripts that include these commands.

### 3.1 Command Converter

The commands described in Section 2 can be converted to Python commands using the Fcl2Py executable. This program requires one argument, which is the command file name. It reads the commands from the file, converts them to the equivalent Python commands, and then writes this information to stdout, which can be piped to a file.

For example:

```
C:/examples/Fcl2Py.exe session01.log > ses01.py
```

### 3.2 Python Module

The PyF3D.dll (or .pyd or .so) file must be imported into Python. The module requires the Vec3D module, which is distributed with the PyF3D module. Two additional modules are required for Ver 8; these are CrackData and Maximize.

Note that MS Windows will have .dll or .pyd extensions, while Linux uses .so files.

The PyF3D module is linked against Python libraries, and Version 8.0 is linked with Python 3.6+. Python versions 3.7 and 3.8 have been tested on MSWindows and Linux; version 3.9 should work but has not been tested yet.

A typical Python script starts by importing the “sys” module and appending the path to the PyF3D.dll file before importing the PyF3D module:

```
import sys
sys.path.append("C:\\FRANC3D_Folder\\")

import PyF3D
#import Vec3D – should be imported automatically
```

Note that file names must be provided using single quoted strings. For MSWindows, the file path separator is the ‘\’ character. This must be defined using two of these characters, as ‘\\’ so that Python will process the path correctly.

### 3.2.1 Setting the Path

The PATH and PYTHONPATH might need to be set.

In MSWindows, environment variables can be set for a user or for the system. Fig 3.1 shows the system-variable PATH and PYTHONPATH setting. Some software programs will only use the system settings, but these typically require ADMIN privileges to set or change.

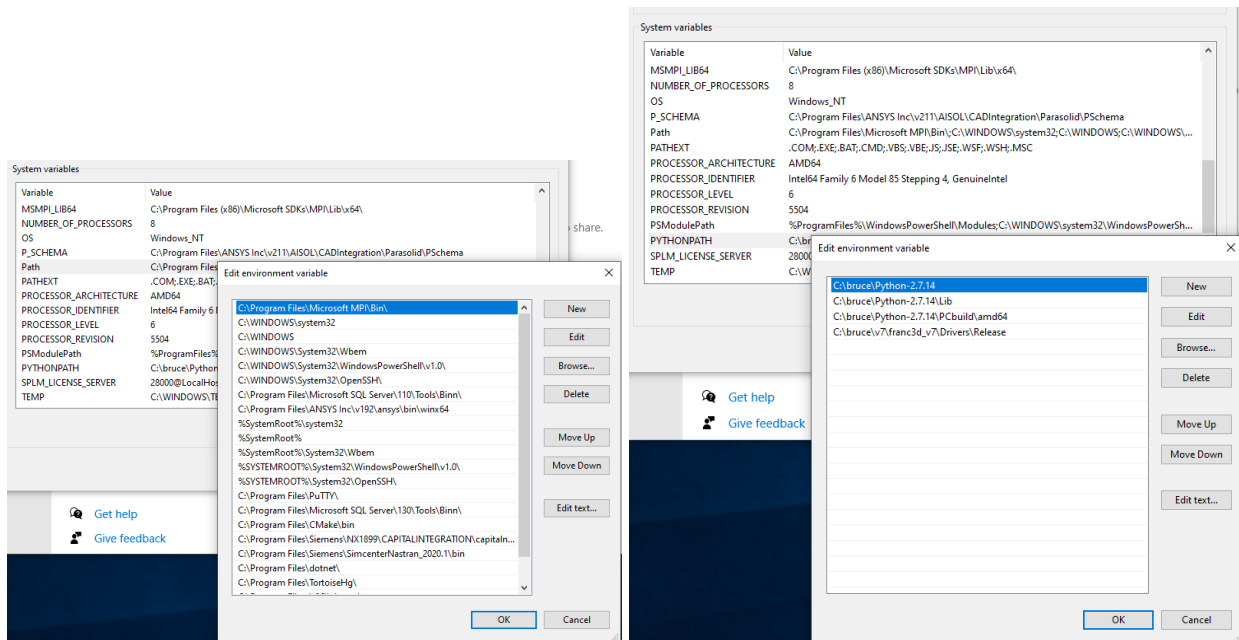


Figure 3.1: PATH and PYTHONPATH system variables.

It might be simpler for users to set the variables on the command line in the MS Windows CMD terminal. Fig 3.2 shows the command to set the PYTHONPATH prior to running a FRANC3D Python script (and prior to running the regular FRANC3D executable if the Python extension for crack growth is used). Substitute your Python-3 folder name in place of Python-2.7.14.

```
D:\tmp\Abaqus_py>set PYTHONPATH=C:\bruce\Python-2.7.14\Lib;C:\bruce\v7\franc3d_v7\Drivers\Release
D:\tmp\Abaqus_py>python run.py
FRANC3D license checkout successful
FRANC3D license checkin successful

D:\tmp\Abaqus_py>set PYTHONPATH=C:\bruce\Python-2.7.14\Lib\
D:\tmp\Abaqus_py>C:\bruce\v7\franc3d_v7\Drivers\Release\franc3d.exe
```

Figure 3.2: PYTHONPATH set from the command line in a CMD window.

It should be noted that ABAQUS 2020 (and older versions) uses Python 2.7, and the system variable setting for the PYTHONPATH can affect ABAQUS. Thus, we recommend that the user set the PATH on the command line (as in Fig 3.2) or use a *.bat* file to first set the variables and then start FRANC3D. In a *.bat* file, the command to set the PYTHONPATH is:

```
set PYTHONPATH=%PYTHONPATH%;C:\bruce\Python-3.8\Lib
```

When ABAQUS runs, it will see this setting and give a warning, but continue running:

```
executing: C:\SIMULIA\Commands\abaqus.bat  
ABAQUS Warning: Recursive loop detected when expanding environment variables:  
PYTHONPATH = %PYTHONPATH%;C:\bruce\Python-3.8\Lib  
Variable will be skipped.
```

This option for setting the PYTHONPATH is important when using Python 3.x with FRANC3D, as the user will be able to set the PYTHONPATH to their Python3.x folder and ABAQUS will ignore that setting and continue using its own Python 2.7 version.

The PATH and PYTHONPATH settings in Linux are usually defined in the user's shell resource file. For example, when using the bash, the file *.bashrc* or *.bash\_profile* in the user's home folder contains lines such as:

```
export PYTHONPATH=$PYTHONPATH:/home/bruce/Python-3.8/
```

### 3.2.2 class F3DApp

There are six classes defined in the PyF3D module: 1) F3DApp, 2) CrackGrowthData, 3) CrackStep, 4) CrackFront, 5) FrontPoint, and 6) IntegrationResults.

The F3DApp class is described here; the other classes are described in the following subsections.

The F3DApp is a FRANC3D application object. This is the Python analog to the main window in the GUI version. Most scripts will require an instance of this object.

constructor:

**F3DApp** - No arguments.

```
example:  
f3d = PyF3D.F3DApp()
```

methods:

All method *examples* below begin with “f3d”, which is the F3DApp object defined above.

**AutoGrowth** – see Section 2.1.1

*example:*

```
f3d.AutoGrowth(  
    model_type="ANSYS",  
    cur_step=1,  
    file_name='my_model',  
    num_steps=4,  
    step_type="SCONST",  
    const_step_size=0.075,  
    maximum_steps=4,  
    flags=["TRANSFER_BC", "CFACE_TRACT"],  
    merge_tol=0.0001,  
    connection_type="MERGE",  
    executable='ANSYS172.exe',  
    command='"ANSYS172.exe" -b -p ansys -np 2  
-i "my_model_STEP_001_full.cdb"  
-o "my_model_STEP_001_full.out" ',  
    global_model='my_GLOBAL.cdb',  
    merge_surf_labels=["AUTO_CUT_SURF"],  
    global_surf_labels=["GLOBAL_CONNECT_SURF"],  
    license="ansys",  
    crack_face_contact=False)
```

**CheckGrowthStatus** – see Section 2.1.2

*example:*

```
f3d.CheckGrowthStatus()
```

**CloseModel** – see Section 2.1.3

*example:*

```
f3d.CloseModel()
```

**ComputeCOD** – see Section 2.1.4

*example:*

```
f3d.ComputeCOD(distance=0.1)
```

**ComputeGrowthParams** – see Section 2.1.5

*example:*

```
f3d.ComputeGrowthParams(sif_method="M_INTEGRAL")
```

**ComputeSif** – see Section 2.1.6

*example:*

```
f3d.ComputeSif(sif_method="M_INTEGRAL",
```

```
do_therm_terms=True)
```

**CrackTractConst** – see Section 2.1.7

*example:*

```
f3d.CrackTractConst(index=1,  
                    pressure=10.0,  
                    load_case=1)
```

**CrackTractDelete** – see Section 2.1.8

*example:*

```
f3d.CrackTractDelete(index=1)
```

**CrackTractExternalDist** – see Section 2.1.9

*example:*

```
f3d.CrackTractExternalDist(index=1,  
                            mesh_file='C:\\temp\\my_dist.cdb',  
                            stress_file='C:\\temp\\my_dist.str',  
                            load_case=1)
```

**CrackTractSurface** – see Section 2.1.10

*example:*

```
f3d.CrackTractSurface(index=0,  
                      dist=[[0,1],[0.5,4],[1,5],[1.5,2],[2,1.5]],  
                      load_case=2,  
                      file_name='my_RS_SURF_0.txt'))
```

**CrackTract1DRad** – see Section 2.1.11

*example:*

```
f3d.CrackTract1DRad(index=0,  
                    axis=1,  
                    offset=[0,0,0],  
                    dist=[[0,1],[0.5,3],[1,0]],  
                    load_case=2))
```

**CrackTract2DRad** – see Section 2.1.12

*example:*

```
f3d.CrackTract2DRad(index=0,  
                    axis=1,  
                    axial=[-0.5,0.0,0.5],  
                    radial=[0.0,1.0],  
                    dist=[[0,1,0],[1,1.25,1]],  
                    load_case=2)
```

**FretModelImport** – see Section 2.1.13

*example:*

```
f3d.FretModelImport(model_type=ANSYS,
```

```
file_name='C:\\temp\\uncracked.cdb',
results_files=['C:\\fretting\\fretting_rig_ls1.str',
               'C:\\fretting\\fretting_rig_ls2.str'],
results_load_cases=[0,1],
contact_pair=[contact_6_6_contact_5_6],
results_option=0)
```

**FretNucleationCycles** – see Section 2.1.14

*example:*

```
f3d.FretNucleationCycles()
```

**FretNucleationDataImport** – see Section 2.1.15

*example:*

```
f3d.FretNucleationDataImport()
```

**GetBuildInfo** – see Section 2.1.16

*example:*

```
print(f3d.GetBuildInfo())
```

**GetCrackData** – see Section 2.1.17; returns CrackData object

*example:*

```
cgd = f3d.GetCrackData()
```

**GetIntegrationResults** – see Section 2.1.18; returns IntegrationResults object

*example:*

```
ir = f3d.GetIntegrationResults()
```

**GetGrowthStatus** – see Section 2.1.19; returns the GrowthStatus

*example:*

```
cgstat = f3d.GetGrowthStatus()
```

**GrowCrack** – see Section 2.1.20

*example:*

```
f3d.GrowCrack(
    do_therm_terms=True,
    median_step=0.1,
    cycles_step=1000,
    front_mult=[1],
    temp_radius_type="ABSOLUTE",
    temp_radius=0.05,
    extrapolate=[[3,3]])
```

**GrowCrackFromFile** – see Section 2.1.21

*example:*

```
f3d.GrowCrackFromFile(file='C:\\temp\\new_front.txt')
```

**GrowMergeCrack** – see Section 2.1.22

*example:*

```
f3d.GrowMergeCrack()
```

**Include** – see Section 2.1.23

*example:*

```
f3d.Include(file='C:\\temp\\include_file.ext')
```

**InsertFileFlaw** – see Section 2.1.24

*example:*

```
f3d.InsertFileFlaw(file='C:\\temp\\init_crack.crk')
```

**InsertMultiFileFlaw** – see Section 2.1.25

*example:*

```
f3d.InsertMultiFileFlaw(  
    file_names='crack_1.crk','crack_2.crk')
```

**InsertMultiParamFlaw** – see Section 2.1.26

*example:*

```
f3d.InsertMultiParamFlaw(  
    flaw_type=["CRACK","CRACK"],  
    crack_type=["EMESH","EMESH"],  
    flaw_params=[[0.1,0.1],[0.15,0.15]],  
    rotation_axes=[[1],[1]],  
    rotation_mag=[[90],[90]],  
    translation=[[0,0.1,1],[0,-0.15,1]],  
    radius=[0.02,0.03])
```

**InsertParamFlaw** – see Section 2.1.27

*example:*

```
f3d.InsertParamFlaw(  
    flaw_type="CRACK",  
    crack_type="EMESH",  
    flaw_params=[0.5,0.5],  
    rotation_axes=[1],  
    rotation_mag=[90],  
    translation=[0,0,10],  
    radius=0.05)
```

**InsertUserBdryFlaw** – see Section 2.1.28

*example:*

```
f3d.InsertUserBdryFlaw(  
    points=[[3.642,5.0,10.054],\+  
           [3.645,5.0,9.984],\+  
           [3.650,5.0,9.910],\+  
           [3.9,5.0,10.14]],  
    front_flags=[1,1,1,0],  
    radius=0.078)
```

**InsertUserMeshFlaw** – see Section 2.1.29

*example:*

```
f3d.InsertUserMeshFlaw(  
    flaw_type="CRACK",  
    mesh_file='surf_mesh_half_penny.cdb',  
    front_groups=["CRACK_FRONT"],  
    rotation_axes=[1],  
    rotation_mag=[-90],  
    translation=[4,5,10.05],  
    radius=0.06)
```

**IntegrateStep** – see Section 2.1.30

*example:*

```
f3d.IntegrateStep()
```

**OpenFdbModel** – see Section 2.1.31

*example:*

```
f3d.OpenFdbModel(file_name='C:\\cube\\crack_cube.fdb')
```

**OpenMeshModel** – see Section 2.1.32

*example:*

```
f3d.OpenMeshModel(model_type="ANSYS",  
    file_name='C:\\cube\\cube_cutout.cdb',  
    global_name='C:\\cube\\cube_GLOBAL.cdb')
```

**ReadFullGrowthHist** – see Section 2.1.33

*example:*

```
f3d.ReadFullGrowthHist(  
    file_name='C:\\cube\\cracked_cube_history.fcg')
```

**ReadGrowthParams** – see Section 2.1.34

*example:*

```
f3d.ReadGrowthParams(  
    file_name='C:\\cube\\cracked_cube_history.fcg')
```

**ReadResponse** – see Section 2.1.35

*example:*

```
f3d.ReadResponse(  
    file_name='C:\\cube\\cracked_cube.dtp')
```

**RunAnalysis** – see Section 2.1.36

*example:*

```
f3d.RunAnalysis(  
    model_type="ANSYS",  
    file_name='static.fdb',  
    flags=["TRANSFER_BC",
```



```

        "CFACE_TRACT", "NO_CFACE_CNTCT"],
merge_tol=0.0001,
connection_type="MERGE",
executable='ANSYS172.exe',
command='"ANSYS172.exe" -b -p ansys -np 2
        -i "static_full.cdb"
        -o "static_full.out"',
global_model='cube_GLOBAL.cdb',
merge_surf_labels=["AUTO_CUT_SURF"],
global_surf_labels=["GLOBAL_CONNECT_SURF"],
license="ansys",
crack_face_contact=False)

```

**SaveFdbModel** – see Section 2.1.37

*example:*

```

f3d.SaveFdbModel(file_name='C:\\temp\\my_model.fdb',
mesh_file_name='C:\\temp\\my_model.cdb',
rslt_file_name='C:\\temp\\my_model.dsp',
analysis_code="ANSYS")

```

**SaveGrowthParams** – see Section 2.1.38

*example:*

```

f3d.SaveGrowthParams(file_name='C:\\temp\\my.cgp)

```

**SaveMeshModel** – see Section 2.1.39

*example:*

```

f3d.SaveMeshModel(file_name='C:\\temp\\my_model.cdb',
model_type="ANSYS")

```

**SetEdgeParameters** – see Section 2.1.40

*example:*

```

f3d.SetEdgeParameters(do_planar_seed=True,
planar_angle=178,
planar_facets=5)

```

**SetGrowthParams** – see Section 2.1.41

*example:*

```

f3d.SetGrowthParams(
growth_type="QUASI_STATIC",
load_step_map=["VERSION: 1", "MAX_SUB: 2",
               "0 0", "0 0", "LABELS: 2",
               "2 Load Step 2",
               "1 Load Step 1"],
kink_angle_strategy="MTS",
quasi_static_n=2,

```

```
quasi_static_loads=["NUM_STEPS: 2",  
                    "1 FINAL 1 1 0",  
                    "2 FINAL 1 1 0"])
```

**SetLoadSchedule** – see Section 2.1.42

**SetMeshingParameters** – see Sections 2.1.43

*example:*

```
f3d.SetMeshingParameters(  
    do_surface_refinement=True,  
    max_vol_restarts=3)
```

**SetStatusFile** – see Sections 2.1.44; command not available in Python.

**SetUnits** – see Sections 2.1.45

*example:*

```
f3d.SetUnits(  
    model_length="MM",  
    model_stress="MPA",  
    temperature="C")
```

**SetUserExtensionsFile** – see Sections 2.1.46

*example:*

```
f3d.SetUserExtensionsFile(  
    file_name='user_python.py',  
    flags=["USER_INITIALIZE",  
          "USER_NEW_POINT",  
          "USER_KINK_ANG"])
```

**SetWorkingDirectory** – see Sections 2.1.47

Note for MSWindows the path separator is \\ and for Linux it is /

*example:*

```
f3d.SetWorkingDirectory(  
    directory='C:\\bruce\\ansys\\ansys_with_temp_mat')
```

**SifHistory** – see Sections 2.1.48

*example:*

```
f3d.SifHistory ()
```

**StartRecording** – writes the subsequent Python commands to a py\_session.log file;  
see Sections 2.1.49

*example:*

```
f3d.StartRecording ()
```

- this could be used if a user writes a complicated Python script and wants to record the commands in a session log to play back in the GUI later

**Submodeler** – see Sections 2.1.50

*example:*

```
f3d.Submodeler(
    model_type="ANSYS",
    orig_file_name='cube.cdb',
    submodel_file_name='cube_LOCAL.cdb',
    global_file_name='cube_GLOBAL.cdb',
    elem_file_name='cube_RETAINED_ELEMS.txt')
```

**WriteCOD** – see Sections 2.1.51

*example:*

```
f3d.WriteCOD(file_name='sif_data.sif'
             flags=["TAB", "COD", "CSD", "CRD", "PNT"])
```

**WriteCrackData** – see Sections 2.1.52

*example:*

```
f3d. WriteCrackData(file_name='crack_info.dat')
```

**WriteEPJ** – see Sections 2.1.53

*example:*

```
f3d. WriteEPJ(file_name='crack_epj.dat')
```

**WriteFatigueData** – see Sections 2.1.54

**WriteGrowthData** – see Sections 2.1.55

*example:*

```
f3d.WriteGrowthData()
```

**WriteGrowthParams** – see Sections 2.1.56

*example:*

```
f3d.WriteGrowthParams()
```

**WriteResolvedSif** – see Sections 2.1.57

**WriteResolvedSifPath** – see Sections 2.1.58

**WriteSERR** – see Sections 2.1.59

**WriteSif** – see Sections 2.1.60

*example:*

```
f3d.WriteSifs(file_name='sif_data.sif'
             flags=["TAB", "XYYY", "KI", "KII", "KIII"])
```

**WriteSifPath** – see Sections 2.1.61

**WriteStdTempData** – see Sections 2.1.62

*example:*

```
f3d.WriteStdTempData (file_name='template_info.dat')
```

### 3.2.3 class CrackData

The following method does not have a command line equivalent.

**GetCrackData** – returns CrackData object

*example:*

```
cgd = f3d.GetCrackData ()
```

The CrackData class stores the crack growth data. The object will contain the following data:

**Step** – contains the list of crack growth steps

*example:*

```
num_steps = len(cgd.Step)
```

**StartPoint** – returns the crack start point or origin (=None if not defined).

*example:*

```
sp = cgd.StartPoint
```

**StartLength** – returns the initial crack length (=None if not defined).

*example:*

```
ilen = cgd.Startlength
```

The CrackData class also has methods to retrieve the number of load steps and the substeps for a load step.

**NumLoadSteps()** – returns the number of load steps

**NumSubSteps(int ls)** – returns the number of substeps for the given load step (ls)

*example:*

```
n_ls = cgd.NumLoadSteps ()  
for i in range (1, n_ls+1):  
    n_ss = cgd.NumSubSteps (i)
```

### 3.2.4 class CrackStep

The CrackStep class stores the crack growth data per growth step. Access the data using the CrackData object:

```
cgs = cgd.Step[0]
```

**Name** – returns the name for the step

*example:*

```
name = cgs . Name
```

**UserIdx** – returns the index for the step

**ExtensionType** – returns the type of crack growth extension for the step

**ExtensionCycles** – returns the number of fatigue cycles for the step

**ExtensionTime** – returns the amount of hold-time for the step

**ExtensionFraction** – returns the extension fractions

**Front** – returns the list of crack fronts

*example:*

```
num_fronts = len(cgs . Front)
```

### 3.2.5 class CrackFront

The CrackFront class stores the crack growth data per crack front. Access the data using the CrackData object:

```
cgf = cgd.Step[0].Front[0]
```

**Id** – returns the crack front ID

*example:*

```
id = cgf . Id
```

**StartPoint** – returns the start point coordinates of the crack front

**StopPoint** – returns the end point coordinates of the crack front

**FitFront** – returns the list of fitted points to the new front

**FitOldFront** – returns the current front points corresponding to the new fitted points

**FitExtensions** – returns the list of extensions between the current and new front points

**ExtensionMult** – returns the extension multiplier

**Point** – returns list of front points

*example:*

```
num_points = len(cgf.Point)
```

### 3.2.6 class FrontPoint

The FrontPoint class stores the crack growth data per crack front point. Access the data using the CrackData object:

```
cgfp = cgd.Step[0].Front[0].Point[0]
```

**K** – returns the SIF values at the point for the given load step and substep

*example:*

```
cgfp.K(1, 1)
```

**J** – returns the J-integral values at the point for the given load step and substep

**T** – returns the T-stress value at the point for the given load step and substep

**G** – returns the G (strain energy release rate) at the point for the given load step and substep; see example for **K**

**COD** – returns the crack opening displacements at the point for the given load step and substep; see example for **K**

**Temp** – returns the temperature at the point for the given load step and substep

**NPos** – returns the normalized position at the point along the crack front

*example:*

```
npos = cgfp.NPos
```

**Coord** – returns the coordinate of the point along the crack front

**Axes** – returns the local axes (three unit-vectors) at the point along the crack front

**KinkAngle** – returns the crack growth kink angle at the point along the crack front

**Extension** – returns the crack extension at the point along the crack front

**NodeId** – returns the node ID of the point along the crack front

**SectId** – returns the element section ID for the point along the crack front

### 3.2.7 class **IntegrationResults**

The following method does not have a command line equivalent.

**GetIntegrationResults** – returns IntegrationResults object

*example:*

```
f3d.IntegrateStep()  
ir = f3d.GetIntegrationResults()
```

The IntegrationResults class stores the subcritical crack growth integration data.

*example:*

ir.reason – reason integrations stopped

ir.step\_cycles – cycles for the current crack growth step

ir.total\_cycles – total cycles for all growth steps

ir.step\_time – time for current crack growth step

ir.total\_time – total time for all growth steps

ir.step\_fraction – fraction of HCF growth in step

ir.step\_passes – number of passes through the schedule for the crack growth step

ir.total\_passes – total number of passes through the schedule for all growth steps

## 4. Python Crack Growth Extensions

The FRANC3D options for crack growth can be extended by using user-supplied subroutines written in the Python programming language.

FRANC3D predefines the names of the user-supplied subroutines. To simplify managing user defined subroutines, they should all be placed in one .py file. FRANC3D reads the .py file and searches for the predefined routine names.

The user defined Python function names, which FRANC3D recognizes, are listed here:

**def on\_initialize()** – This function is called once before any other user extensions are called. It primarily is used to initialize any global variables used by other functions.

**def on\_kink\_angle()** – During crack growth, this function is called once for each crack-front point on each crack front. The purpose of the function is to compute the “kink” angle (deviation from planar growth), which determines the direction of crack growth. The extension for this direction is computed using one of the algorithms built into FRANC3D. The function is not passed any arguments; data needed to compute a new crack-front coordinate is obtained using the predefined data access routines described below. The function should return a scalar value, which is the kink angle *in radians*.

**def on\_new\_point()** – During crack growth, this function is called once for each crack-front point on each crack front. The purpose of the function is to compute the polar coordinates of a corresponding point on a new (extended) crack front. The new point lies in the plane that is perpendicular to the crack front at the current crack front point. The function is not passed any arguments; data needed to compute a new crack-front coordinate is obtained using the predefined data access routines described below. The function should return two scalar values, the crack extension, and the kink angle in radians, in that order. The algorithms built into FRANC3D for computing kink angle can be accessed using routines described below.

**def on\_cycles\_growth\_rate(DK,R,temp)** – This function allows a user to define a crack growth rate for a material due to cyclic fatigue loading (e.g.,  $da/dN$ ). It is passed the stress intensity factor range ( $DK = K_{max} - K_{min}$ ), the stress ratio ( $R = K_{min} / K_{max}$ ), and the local temperature. The function should return the crack extension per load cycle.

**def on\_time\_growth\_rate(K,temp)** – This function allows a user to define a crack growth rate for a material as a function of time (e.g.,  $da/dt$ ). It is passed the stress intensity factor and the local temperature. The function should return the crack extension per unit time increment.

**def on\_start\_step()** – This function is called at the beginning of each crack growth step and allows one to set values for global variables used to compute crack extensions for the step. It is not passed any arguments and does not return any values.

**def on\_end\_step()** – This function is called at the end of each crack growth step. It is not passed any arguments and does not return any values.



## 4.1 Data Access Functions

To implement the functions described above, it might be necessary to access data in the FRANC3D crack database. The following functions are predefined in the user Python environment and can be called to access the crack data.

`NumCrackSteps()` – returns the number of crack growth steps in the current model. Use the `SetCrackStepNum` function to set the current crack step to a specified step.

`NumCrackFronts()` – returns the number of crack fronts for the current crack step. Use `SetCrackFrontNum` function to set the current crack front to a specified front for the current crack step.

`NumCrackPoints()` – returns the number of crack front points for the given step and crack front. Use the `SetCrackIndexNum` function to set the crack front point ID for the current crack front and crack step.

`NumLoadSteps()` – returns the number of load steps.

`NumLoadSubsteps(load_step)` – returns the number of sub steps for the given load step.

`GetCrackStepNum()` – returns the current crack step number.

`GetCrackFrontNum()` – returns the current crack front number.

`GetCrackPointNum()` – returns the index of the current point on the crack front.

`GetNPos()` – returns the normalized crack front coordinate of the current crack front point.

`GetCoord()` – returns the Cartesian coordinates of the current crack front point as a `Vec3D`.

`GetXAxis()` – returns the direction cosines of the x-axis of the local crack front coordinate system relative to the global Cartesian coordinates as a `Vec3D`.

`GetYAxis()` – returns the direction cosines of the y-axis of the local crack front coordinate system relative to the global Cartesian coordinates as a `Vec3D`.

`GetZAxis()` – returns the direction cosines of the z-axis of the local crack front coordinate system relative to the global Cartesian coordinates as a `Vec3D`.

`GetK([step[,sub step]])` – returns stress intensity factors (modes I, II, and III) as a `Vec3D`. If no arguments are given, the `Ks` for load step 1 are returned. If no sub step argument is given, the `Ks` for the final (or only) sub step are returned.

GetT([step[,sub step]]) – returns a T-stress (if available). If no arguments are given, the T for load step 1 is returned. If no sub step argument is given, the T for the final (or only) sub step is returned. None is returned if the T-stress is not available.

GetJ([step[,sub step]]) – returns a J-Integral value. If no arguments are given, the J for load step 1 is returned. If no sub step argument is given, the J for the final (or only) sub step is returned.

GetG([step[,sub step]]) – returns energy release rates (modes I, II, and III) as a Vec3D. If no arguments are given, the Gs for load step 1 are returned. If no sub step argument is given, the Gs for the final (or only) sub step are returned. None is returned if Gs are not available.

GetCOD([step[,sub step]]) – returns crack opening displacements (modes I, II, and III). If no arguments are given, the CODs for load step 1 are returned. If no sub step argument is given, the CODs for the final (or only) sub step are returned. None is returned if CODs are not available.

GetTemp([step[,sub step]]) – returns a crack-front temperature. If no arguments are given, the temperature for load step 1 are returned. If no sub step argument is given, the temperature for the final (or only) sub step are returned. None is returned if temperatures are not available.

GetDcPoint() – returns the coordinates of the points used to evaluate displacement correlation stress intensity factors as a Vec3D.

Maximize(x\_start,x\_stop,func,data) – Finds the value of  $x$  in the range  $x\_start - x\_stop$  that maximizes the user supplied function  $func$ . The function is passed the current  $x$  and the  $data$  (i.e.,  $my\_func(x,data)$ ).

SetCrackStepNum(step) – used to set the crack growth step number. This is used to access data (K's, coordinates, etc) for steps other than the current crack step.

SetCrackFrontNum(front) – used to set the crack front number. This is used to access data (K's, coordinates, etc) for fronts other than the current crack front.

SetCrackPointNum(index) – used to set the crack front point index. This is used to access data (K's, coordinates, etc) for crack front points other than the current point.

MtsKinkAngle(Kvec) – returns a kink angle computed by the maximum tensile stress (max hoop stress) criterion. The argument is the  $K_I$ ,  $K_{II}$ , and  $K_{III}$  values either as Vec3D or a sequence of three floats

MssKinkAngle(Kvec,eta\_II,eta\_III) – returns a kink angle computed by the maximum shear stress criterion. The first argument is the  $K_I$ ,  $K_{II}$ , and  $K_{III}$  values either as Vec3D or a sequence of three floats. The second and third arguments are parameters used to weight the mode II and mode III. The maximum shear stress kink criterion is the direction is

$$\max \left( \sqrt{\left( h_{II} K_{II}^r(q) \right)^2 + \left( h_{III} K_{III}^r(q) \right)^2} \right),$$

where  $K^r$  is the resolved stress intensity factors in the  $\theta$  direction.

`GeneralKinkAngle(Kvec,eta_II,eta_III)` – computes the kink angle by both the maximum tensile stress and maximum shear stress criteria and returns a kink angle associated with the greater resolved stress intensity factor of the two. The first argument is the  $K_I$ ,  $K_{II}$ , and  $K_{III}$  values either as `Vec3D` or a sequence of three floats. The second and third arguments are parameters used to weight the mode II and mode III.

`SerrKinkAngle(Kvec,eta_II,eta_III)` – computes the kink angle by a maximum strain energy release rate criterion. The first argument is the  $K_I$ ,  $K_{II}$ , and  $K_{III}$  values either as `Vec3D` or a sequence of three floats. The second and third arguments are parameters used to weight the mode II and mode III. The maximum strain energy release rate kink criterion is the direction is

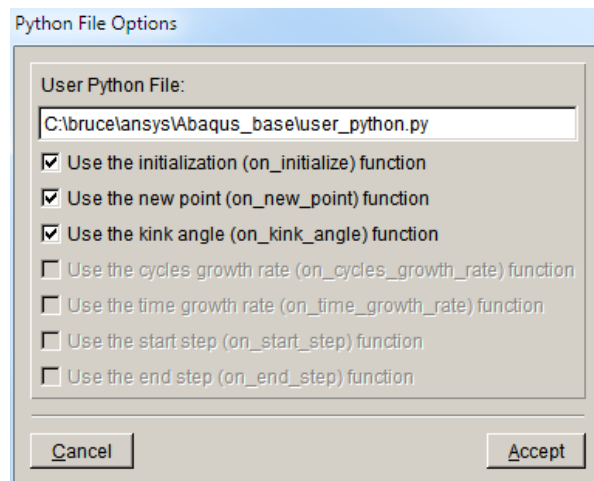
$$\max \left( K_I^r(q)^2 + \left( h_{II} K_{II}^r(q) \right)^2 + \left( h_{III} K_{III}^r(q) \right)^2 \right),$$

where  $K^r$  is the resolved stress intensity factors in the  $\theta$  direction.

`Vec3D(e1,e2,e3)` – is a predefined class that stores a vector of three values. Individual components can be accessed like a list (e.g., `Kvec = GetK()` ; `KII = Kvec[1]`). In addition, most common arithmetic operations (addition, subtraction, scalar multiplication, scalar division, inner product, etc) are defined for `Vec3D`'s

## 4.2 Specifying a user extension file

Before any user python extensions can be used, the file containing the python code must be read into FRANC3D. Under the **Advanced** menu, select the **Read User Extensions...** option. After selecting the .py file FRANC3D scans the file to find which user extensions are defined in the file. It then displays the dialog shown below, which gives one the option to turn on or off defined functions.



### 4.3 Example user extension files

This example implements a crack growth extension based on a Paris type model where the C coefficient is a linear function of the temperature. It calls a built-in function to compute the kink angle.

```
# =====
# User defined crack extension function:
#
# This version computes an extension based on a temperature dependent
# Paris model for a specified number of cycles.
# =====
# -----
# initialization function
# -----

def on_initialize():

    global C_b, C_m, n, cycles

    C_b = 100.0
    C_m = 10.0
    n = 3
    cycles = 1000.0
    return

# -----
# crack extension function
# -----

def on_new_point():

    global C_b, C_m, n, cycles

    C = C_m * GetTemp() + C_b

    dadN = C * GetK()[0]**n

    angle = MtsKinkAngle(GetK()[0])

    return dadN * cycles, angle
```

This example implements the maximum tensile stress kink angle criterion. The maximum tensile stress criterion is built-in to FRANC3D, but this example shows how it could be done as a user extension.

```
# =====
# User defined kink angle function:
#
# This version computes the Max Tensile Stress criterion
# for the sum of the Ks from all load cases
# =====
```

```

import math

# -----
# function to compute the hoop stress
# -----

def hoop_stress(theta,Ksum):

    KIIf = Ksum[0] * (math.cos(theta/2) * (1.0-math.sin(theta/2)**2))
    KIIIf = 0.75 * Ksum[1] * (math.sin(theta/2) + math.sin(3*theta/2))
    return KIIf - KIIIf

# -----
# kink angle function
# -----

def on_kink_angle():

    # compute the sum of the Ks for all load steps (note
    # load steps are numbered from 1 to n

    Ksum = Vec3D(0,0,0)
    for i in xrange(NumLoadSteps()) : Ksum += GetK(i+1)

    # call the F3D builtin Maximize function to find the
    # angle where the hoop stress is maximum

    max_ang = Maximize(-math.pi/2,math.pi/2,hoop_stress,Ksum)

    return max_ang

```

This example implements the computation of a new crack front point where the kink angle determined by the maximum tensile stress criterion and the crack extension is computed using a Paris model.

```

# =====
# User defined new point function:
#
# This version computes a new crack front point location
# based on a max hoop stress angle criterion and an assumed
# Paris like growth model
# =====

import math

# -----
# function to compute the hoop stress
# -----

def hoop_stress(theta,Ksum):

    KIIf = Ksum[0] * (math.cos(theta/2) * (1.0-math.sin(theta/2)**2))
    KIIIf = 0.75 * Ksum[1] * (math.sin(theta/2) + math.sin(3*theta/2))
    return KIIf - KIIIf

```

```
# -----  
# new point function  
# -----  
  
def on_new_point():  
  
    # compute the sum of the Ks for all load steps (note  
    # load steps are numbered from 1 to n  
  
    Ksum = Vec3D(0,0,0)  
    for i in xrange(NumLoadSteps()) : Ksum += GetK(i+1)  
  
    # call the F3D builtin Maximize function to find the  
    # angle where the hoop stress is maximum  
  
    angle = Maximize(-math.pi/2,math.pi/2,hoop_stress,Ksum)  
  
    # compute an extension based on a Paris like model  
  
    extend = 0.001 * Ksum[0]**3.2  
  
    return extend, angle
```