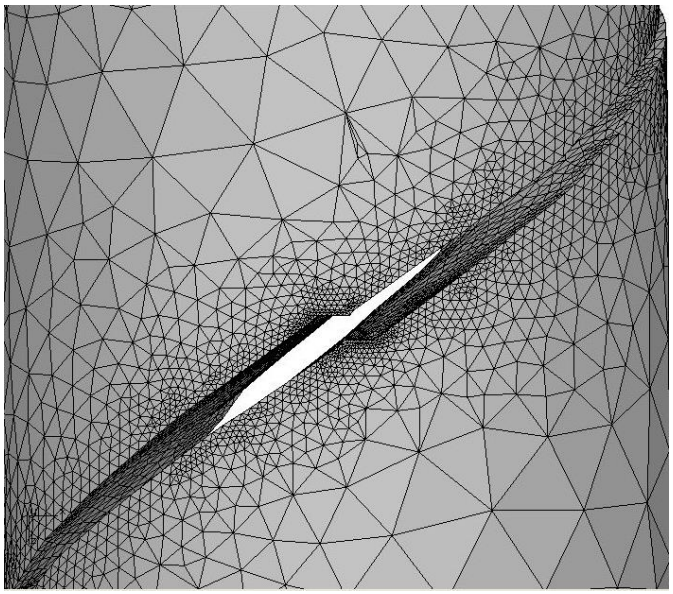
FRANC3D

Command Language & Python Extensions



Version 7.1

Fracture Analysis Consultants, Inc
www.fracanalysis.com

Revised: Nov 2017

Table of Contents:

1. Introduction

This document describes the FRANC3D command language and the Python extensions. The PyF3D module works with Python versions 2.7.6 - 2.7.10 (and probably 2.7.11). If your system does not have one of these versions, you can download and build Python yourself (see www.python.org).

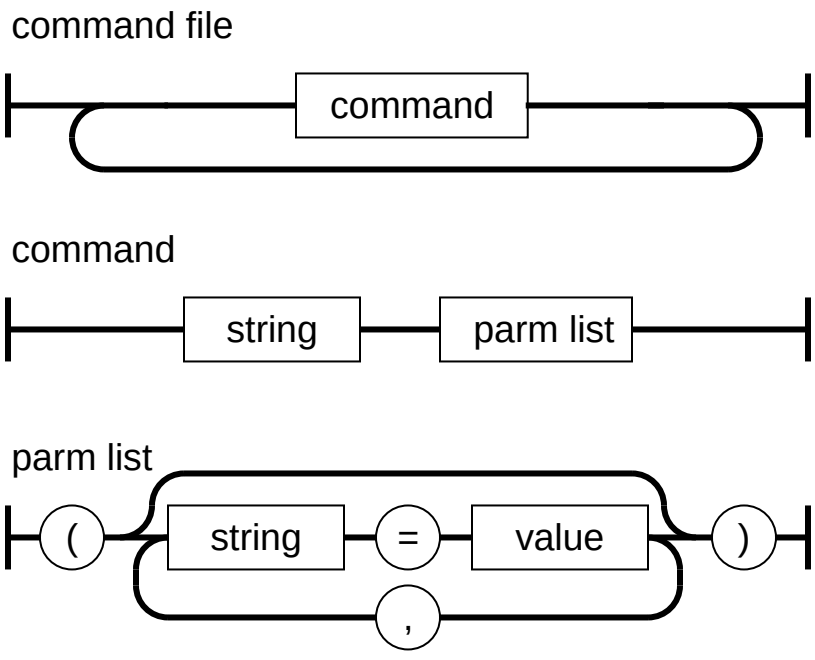
2. Command File Syntax

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

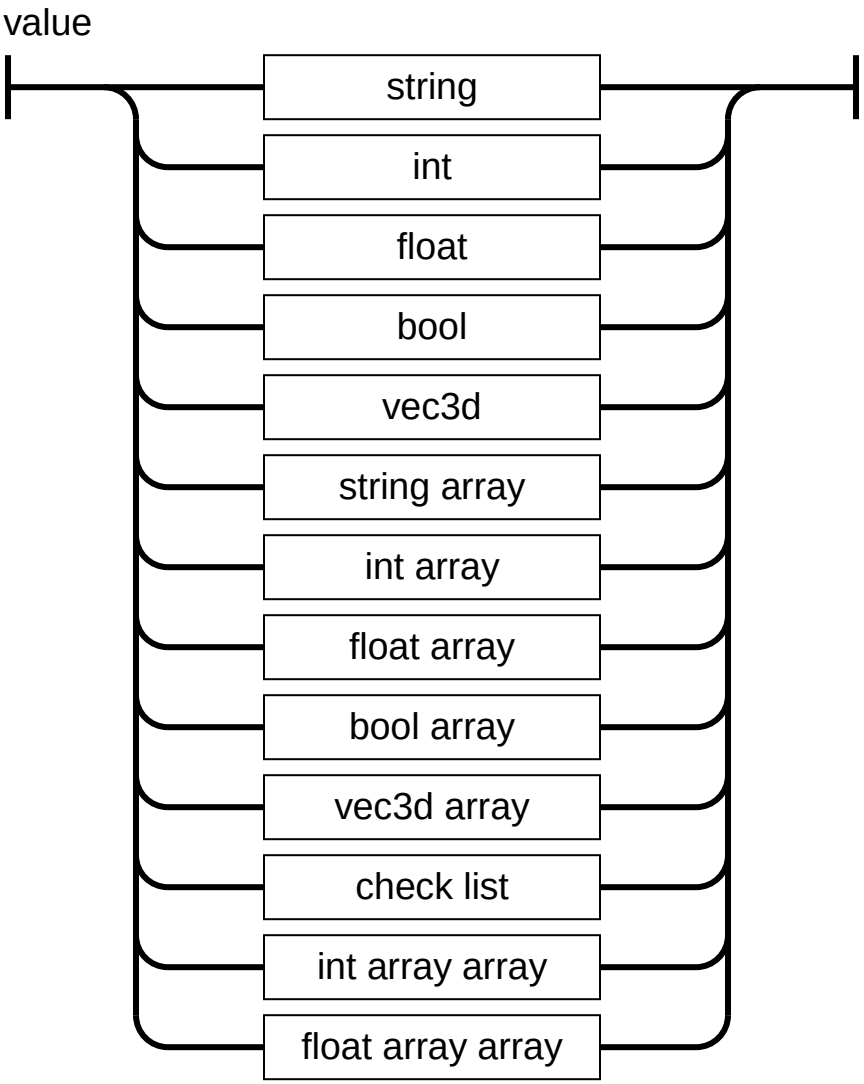
These session files contain a series of command statements. In command files, lines starting with '#' are comment lines. Informally, the command statements look something like:

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

More formally, the syntax diagram for a command files is:

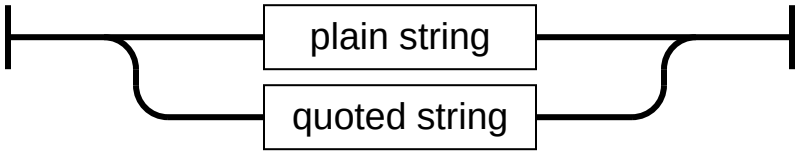


Each parameter value is one of the following types:

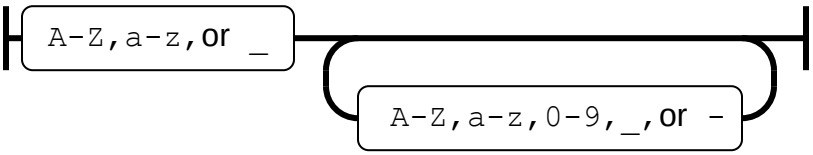


Where the types are defined as:

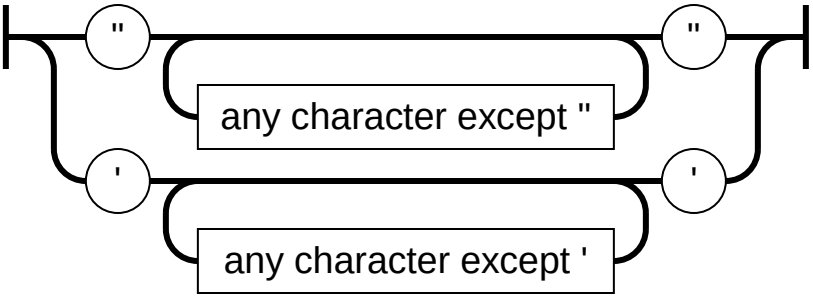
string



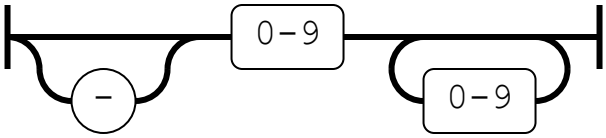
plain string



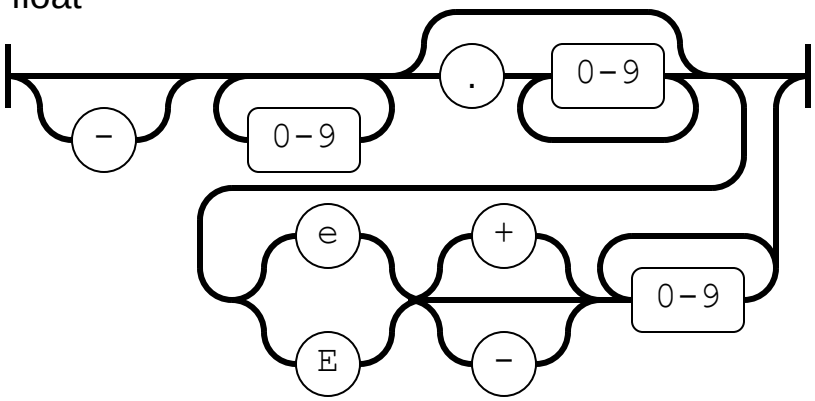
quoted string



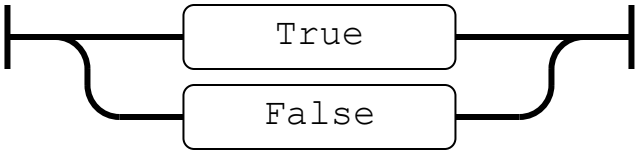
int



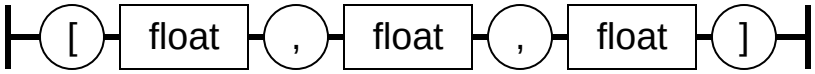
float



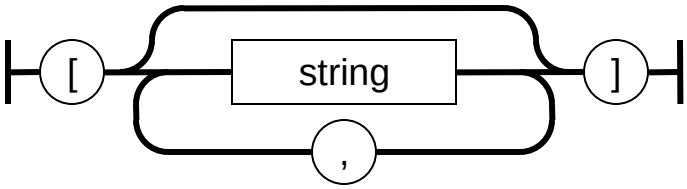
bool



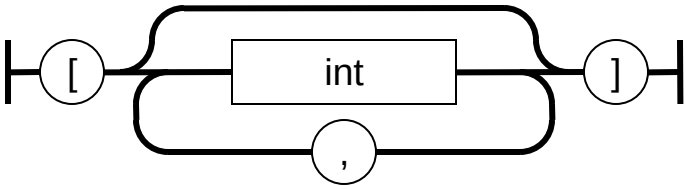
vec3d



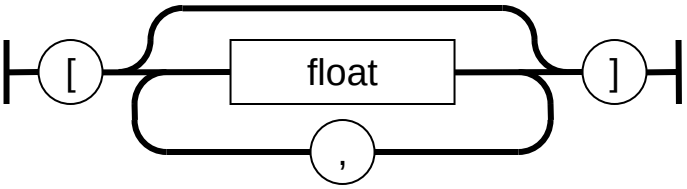
string array



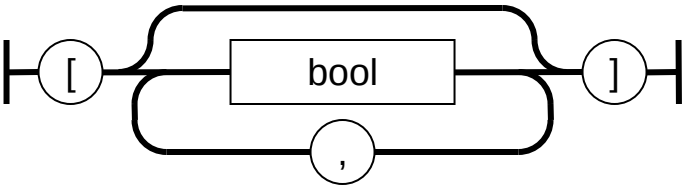
int array



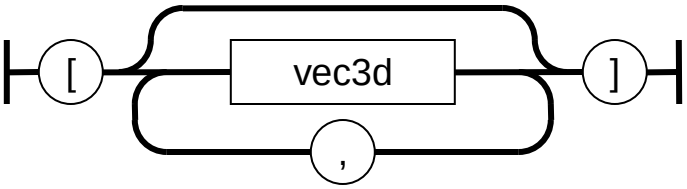
float array



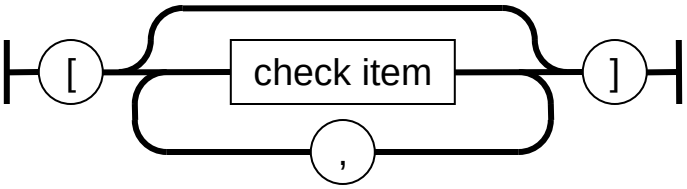
bool array

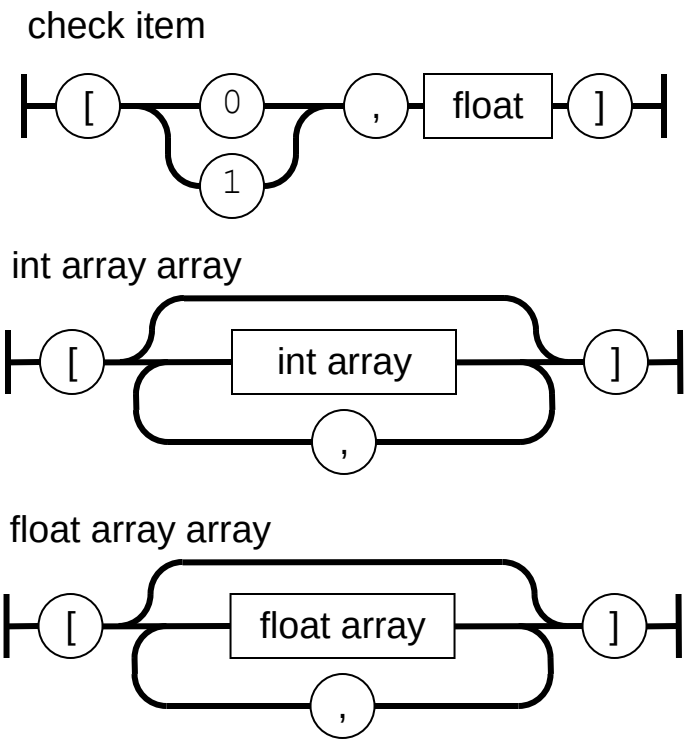


vec3d array



check list





2.1 Commands

The FRANC3D commands are listed next with their parameter list followed by an example. The 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) using the solver specified by the model type parameter.

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 2.1.1.2
template_params (opt) – see 2.1.1.3
front_fitting_params (opt) – see 2.1.1.4
analysis_options (req) – see 2.1.1.5

example:

```
AutoGrowth (
   model_type="ABAQUS",
   cur_step=1,
   file_name='/models/abaqus_cube/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=[NO_WRITE_TEMP,TRANSFER_BC,
       NO_CFACE_TRACT,NO_CFACE_CNTCT],
connection_type=MERGE,
executable='abaqus.bat',
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
crack_face_press_type = *string* (opt) - crack-face pressure type
crack_face_press = *float* (opt) - crack-face pressure
do_crack_face_contact = *bool* (opt) - include contact pressure terms
correct_mid_side = *bool* (opt) - correction term for curved crack fronts
large_rotations = *bool* (opt) - correction term large rigid body type rotation

2.1.1.2 growth_params

parameters:

growth_type = *string* (opt) - SIF computation method
fem_units = *string* (opt) – US or SI units
SERR_eff = *bool* (opt) – strain energy release rate
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
accelerated = *bool* (opt) – accelerated integration
constant_t_k = *bool* (opt) – constant SIF over time
extension_type = *string* (opt) – extension type
eta_II = *float* (opt) –

eta_III = *float* (opt) –
load_schedule_data = *string* (opt) – load schedule
growth_model = *string* (opt) – crack growth rate model
load_step_map = *string* (opt) – FE load step map
retardation_alg = *string* (opt) – crack retardation model
willenborg_params = *float array* (opt) – Willenborg model parameters
kink_angle_strategy = *string* (opt) – crack growth kink angle model
quasi_static_n = *float* (opt) – quasi-static crack growth power
quasi_static_loads = *string* (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

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 > Kc
check_DKth = *bool* (opt) – check to see if DK < DKth
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_load_case = *int array* (opt) – HCF load steps

2.1.1.4 growth_variables

median_step = *float* (opt) – median crack growth step size
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
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 – partial fitting and extrapolation
- NOFIT_NOEXTRAP – no fitting or extrapolation
- HERMITIAN – Hermitian polynomial fit through closed-front points

polynomial_order = *int array* (opt) – polynomial order for each front
discard = *int array array* (opt) – discard end points for each front
extrapolate = *float array array* (opt) – extrapolate curve ends for each front
retain_all_nodes = *bool array array* (opt) – retains nodes as geometric points on the new crack front; only applies to GrowCrackFromFile

2.1.1.7 general_analysis_options

flags = *string* (req) - list of analysis options:
- TRANSFER_BC - transfer boundary conditions from uncracked mesh (default)
- NO_TRANSFER_BC - do not transfer of boundary conditions
- WRITE_TEMP - write nodal temperatures (set if needed)
- NO_WRITE_TEMP - do not write nodal temperatures
- 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
- CHECK_ONLY - perform a data check analysis only
- CONTOUR_INTEGRAL - perform contour integral calculation

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
file_name = *single quoted string* (req) - mesh file name
global_model = *single quoted string* (opt) - name of the global model
executable = *single quoted string* (opt) - path to the analysis executable
command = *single quoted string* (opt) - analysis command passed to the OS
merge_surf_labels = *string array* (opt) - labels for the merge surfaces on the submodel
global_surf_labels = *string array* (opt) - labels for the merge surfaces on the global model

2.1.2 CloseModel( )

Close the current model.

example:

```
CloseModel()
```

2.1.3 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)
```

2.1.4 ComputeGrowthParams( )

Compute 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,
   paris_C=1.0e-10,paris_N=2)
```

2.1.5 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)
```

2.1.6 CrackTractConst( )

Define or edit constant crack-face tractions.

parameters:

index = *int* (req) - crack traction index
press = *float* (req) - constant pressure magnitude
load_case = *int* (req) - traction load case

example:

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

2.1.6.1 CFT index and load case

All crack surface tractions (CFT) have an internal index. Indexes start at 0; each CFT has its own index. 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.

2.1.7 CrackTractDelete( )

Delete a crack-face traction.

parameters:

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

example:

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

2.1.8 CrackTractExternalDist( )

Define or edit 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

example:

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

2.1.9 CrackTractSurface( )

Define or edit surface treatment crack-face tractions.

parameters:

index = *int* (req) - crack traction index
dist = *float array array* (req) - distribution distance/traction pairs
load_case = *int* (req) - traction load case
file_name = *string* (opt) - text file with element face ids

example:

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

2.1.10 CrackTract1DRad( )

Define or edit 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 array* (req) - distribution distance/traction pairs
load_case = *int* (req) - traction load case

example:

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

2.1.11 CrackTract2DRad( )

Define or edit 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 array* (req) - table of traction values for all radial and axial locations
load_case = *int* (req) - traction load case

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)
```

2.1.12 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_load_cases = *int array* (req) – list of load case ids in results
contact_pair = *string array* (req) – contact pair name
results_option = *int* (req) – read pair of results 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_load_cases=[0,1],
      contact_pair=[contact_6_6_contact_5_6],
      results_option=0)
```

2.1.13 FretNucleationCycles( )

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_MSWT – modified Smith-Watson-Topper (Fatemi-Socie) model

fretting_ params = *float array* (req) – fretting model parameters
do_averaging = *bool* (opt) – do volume averaging of stress / strain
add_residual = *bool* (opt) – add residual stress
save_file = *string* (opt) – file name to save fretting cycles

example:

```
FretNucleationCycles(fretting_model_type=FRET_SEQ,
      fretting_params=[0.43,52476,-0.6471,450.85,-0.03582],
      do_averaging=false)
```

2.1.14 FretNucleationDataImport( )

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='C:\temp\fret_data.txt',
        fretting_function_type=1,
        fretting_function_id=2)
```

2.1.15 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

example:

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

2.1.16 GrowCrackFromFile( )

Grow crack front(s) 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'))
```

2.1.17 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

2.1.18 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 2.1.18.1
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)
```

2.1.18.1 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
refinement_level = *int* (opt) - number of times the flaw patches will be recursively subdivided
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 element 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 intersect free surfaces only if the crack-front meets the surface at nearly a right angle, otherwise the template terminates within the body

(default = false)

do_template = *bool* (opt) – true if template used (default = true)

2.1.19 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'])
```

2.1.20 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) - 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) - list of void types:
- ELLIPSOID - ellipsoidal flaw
- NONE - place holder for crack types

flaw_params = *float array array* (req) - list of lists of flaw size parameters

flaw_insert_params *array* => see Section 2.1.18.1

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])
```

2.1.21 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 2.1.18.1

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)
```

2.1.22 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

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)
```

2.1.23 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
flaw_insert_params => see Section 2.1.18.1

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)
```

2.1.24 MapState( )

Map material state variables between the previous and the current model – is not currently implemented.

parameters:

flags = *string array* (req) - list of state variables to map:
- TEMP - nodal temperatures
- DISP - nodal displacements
- STRESS - nodal stresses
- STRAIN - nodal strains

example:

```
MapState(
   flags=[DISP,STRESS,STRAIN])
```

2.1.25 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')
```

2.1.26 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

example:

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

2.1.27 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')
```

2.1.28 ReadGrowthParams ( )

Read a crack growth parameters file.

parameters:

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

example:

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

2.1.29 ReadResponse( )

Read a response file.

parameters:

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

example:

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

2.1.30 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=[NO_WRITE_TEMP,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])
```

2.1.31 SaveFdbModel( )

Save a FRANC3D database (.fdb) file.

parameters:

file_name = *single quoted string* (req) - .fdb file name
mesh_file_name = *single quoted string* (opt) - file name for the mesh model
rslt_file_name = *single quoted string* (req) - file name for the FEM results
global_file_name = *single quoted string* (opt) - file name for the global mesh
anaysis_code = *string* (req) - analysis code:
   ABAQUS, ANSYS, or NASTRAN

flags = *string array* (opt) - list of analysis option flags
- LINEAR - use first order elements
- QUADRATIC - use second order elements (default)
- TRANSFER_BC - transfer boundary conditions from uncracked mesh
- NO_TRANSFER_BC - renumber nodes and elements (dense numbering from 1)

example:

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

2.1.32 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')
```

2.1.33 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
- LINEAR - use first order elements
- QUADRATIC - use second order elements (default)
- TRANSFER_BC - transfer boundary conditions from uncracked mesh
- NO_TRANSFER_BC - renumber nodes and elements (dense numbering from 1)

example:

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

2.1.34 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)
```

2.1.35 SetGrowthParams( )

Set parameters for crack growth.

parameters:

growth_params => see Section 2.1.1.2: growth_params

example:

```
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"])
```

2.1.36 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')
```

2.1.37 SetMeshingParameters( )

Modify meshing parameters.

parameters:

max_gen_elems = *int* (opt) - maximum number of elements that will be generated
surf_refine_bdry_factor = *float* (opt) - boundary refinement factor for surface meshing
surf_near_bdry_factor = *float* (opt) - near boundary node factor for surface meshing
optimal_sphere_factor = *float* (opt) - optimal sphere size factor for volume meshing
optimal_size_factor = *float* (opt) - optimal octtree size factor for volume meshing
volume_refine_factor = *float* (opt) - optimal octtree refinement factor for volume meshing
all_planar_surf_meshing = *bool* (opt) - force planar surface meshing
do_surface_smoothing = *bool* (opt) - surface smoothing flag
do_coarsen_crack_mouth = *bool* (opt) - coarsen crack mouth flag
do_surface_refinement = *bool* (opt) - surface element refinement flag
max_vol_restarts = *int* (opt) - maximum number of volume meshing restarts
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

example:

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

2.1.38 SetPrevMeshTool( )

Set previous mesh and results.

example:

```
SetPrevMeshTool()
```

2.1.39 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')
```

2.1.40 SetTrimRegions( )

Specify the ID's of regions into which crack will not grow. They will be trimmed at the bi-material interface.

parameters:

region = *int array* (req) - list of region ids

example:

```
SetTrimRegions(
  regions=[3])
```

2.1.41 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)
```

2.1.42 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])
```

2.1.43 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')
```

2.1.44 SifHistory( )

Compute the SIF history.

parameters:

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 = *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

example:

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

2.1.45 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')
```

2.1.46 WriteCOD( )

Generate a file containing crack-front crack opening displacement 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
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
PNT -	include COD evaluation point coordinates
AXES -	include crack-front local coordinate axes
MODULI -	include the Youngs' moduli at the front and evaluation points

example:

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

2.1.47 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')
```

2.1.48 WriteFatigueData( )

Generate a file containing crack-front crack opening displacement data.

parameters:

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

2.1.49 WriteGrowthParams( )

Generate a file containing crack-growth data.

parameters:

file_name = *single quoted string* (req) - output file name
front_id = *int* (opt) – crack front id
flags = *string array* (opt) – array of output options:

SPACE -	use spaces as delimiters
TAB -	use tabs as delimiters (default)
COMMA -	use commas as delimiters
XYYY -	first column is position, remaining columns are values (default)
XYXY -	odd columns are position, even columns are values
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
CRD -	include crack-front Cartesian coordinates
AXES -	include crack-front local coordinate axes
ANGLE -	include the predicted kink angle
DIR -	include the normalized predicted propagation direction
REVERSE -	reverse the ordering of the values
DK -	include delta K1
EXT -	include extension
R -	include R ratio
FULL_STEP -	
GI -	include mode I energy release rate
GII -	include mode II energy release rate
GIII -	include mode III energy release rate

example:

```
WriteGrowthParams(
   file_name='growth_params.dat')
```

2.1.50 WriteSERR( )

Generate a file containing strain energy release rate 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
flags = *string array* (opt) - array of output options:
SPACE - use spaces as delimiters
TAB - use tabs as delimiters (default)
COMMA - use commas as delimiters
XYYY - first column is position, remaining columns are values (default)
XYXY - odd columns are position, even columns are values
GI - include mode one SERR
GII - include mode two SERR
GIII - include mode three SERR
J - include J-integral values
CRD - include crack-front Cartesian coordinates
AXES - include crack-front local coordinate axes
ANGLE - include the predicted kink angle
DIR - include the normalized predicted propagation direction
REVERSE - reverse the ordering of the values

example:

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

2.1.51 WriteSif( )

Generate a file containing crack-front fracture parameters and 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
flags = *string array* (opt) - array of output options:
SPACE - use spaces as delimiters
TAB - use tabs as delimiters (default)
COMMA - use commas as delimiters
XYYY - first column is position, remaining columns are values (default)
XYXY - odd columns are position, even columns are values
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

CRD - include crack-front Cartesian coordinates
AXES - include crack-front local coordinate axes
ANGLE - include the predicted kink angle
DIR - include the normalized predicted propagation direction
REVERSE - reverse the ordering of the values

example:

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

2.1.52 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
flags = *string array* (opt) - array of output options: see WriteSif flags

example:

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

2.1.53 WriteSifHistory( )

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
flags = *string array* (opt) - array of output options: see WriteSif flags
geometry_name = *single quoted string* (opt) – geometry file name

example:

```
WriteSifHistory(
   file_name=sif_hist.sif)
```

2.1.54 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

An example command file might look like this:

```
# FRANC3D Version 7.1.0

SetWorkingDirectory(
   directory='C:\Abaqus_base')

OpenMeshModel(
   model_type=ABAQUS,
   file_name='Abaqus-Cube.inp',
   retained_nodes_file='Abaqus-Cube_RETAINED.txt')

InsertFileFlaw(
   file_name='Cube_Crack.crk')

RunAnalysis(
```

```
    model_type=ABAQUS,
    file_name='Abaqus_cube_crack.fdb',
    flags=[NO_WRITE_TEMP,TRANSFER_BC,NO_CFACE_TRACT,NO_CFACE_CNTCT],
    connection_type=MERGE,
    executable='abaqus.bat',
    command='"abaqus.bat" job=Abaqus_cube_crack ask_delete=NO -interactive -analysis ')

ComputeSif()

SetUnits(
    model_length=MM,
    model_stress=MPA,
    temperature=C)

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=[NO_WRITE_TEMP,TRANSFER_BC,NO_CFACE_TRACT,NO_CFACE_CNTCT],
    connection_type=MERGE,
    executable='abaqus.bat',
    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 PyF3D.pyd) must be imported into Python. The module requires the Vec3D.py module, which is distributed with the PyF3D module. The PyF3D module is linked against Python libraries. The current version 7.0 module is linked with Python 2.7. A typical Python script would start 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")
import PyF3D
import Vec3D
```

Note that for MSWindows, PyF3D.pyd is the same as PyF3D.dll; just the extension is changed.

There are eight classes defined in this module: 1) F3DApp, 2) FemModel, 3) FemResults, 4) Flaw, 5) CrackGrowthData, 6) CrackStep, 7) CrackFront, and 8) FrontPoint.

File names must be provided using single quoted strings. Note that 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 class F3DApp

The F3DApp is a FRANC3D application object. This is the Python analog to the main window in the GUI version. Most interesting things involve an instance of this object.

constructor:

F3DApp - No arguments.

example:

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

methods:

All of the 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)
```

CloseModel – see Section 2.1.2

example:

```
f3d.CloseModel()
```

ComputeCOD – see Section 2.1.3

example:

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

ComputeGrowthParams – see Section 2.1.4.

example:

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

ComputeSif – see Section 2.1.5

example:

```
f3d.ComputeSif(sif_method="M_INTEGRAL",
      do_therm_terms=True)
```

CrackTractConst – see Section 2.1.6

example:

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

CrackTractDelete – see Section 2.1.7

example:

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

CrackTractExternalDist – see Section 2.1.8

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 2.1.9

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 2.1.10

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 2.1.11

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 2.1.12

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 2.1.13

example:

```
f3d.FretNucleationCycles()
```

FretNucleationDataImport – see Section 2.1.14

example:

```
f3d.FretNucleationDataImport()
```

GrowCrack – see Section 2.1.15

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 2.1.16

example:

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

Include – see Section 2.1.17

example:

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

InsertFileFlaw – see Section 2.1.18

example:

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

InsertMultFileFlaw – see Section 2.1.19

example:

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

InsertMultParamFlaw – see Section 2.1.20

example:

```
f3d.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])
```

InsertParamFlaw – see Section 2.1.21

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 2.1.22

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 2.1.23

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)
```

MapState – see Section 2.1.24

example:

```
f3d.MapState()
```

OpenFdbModel – see Section 2.1.25

example:

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

OpenMeshModel – see Section 2.1.26

example:

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

ReadFullGrowthHist – see Section 2.1.27

example:

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

ReadGrowthParams – see Section 2.1.28

example:

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

ReadResponse – see Section 2.1.29

example:

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

RunAnalysis – see Section 2.1.30

example:

```
f3d.RunAnalysis(
      model_type="ANSYS",
      file_name='static.fdb',
      flags=["WRITE_TEMP","TRANSFER_BC",
             "CFACE_TRACT","NO_CFACE_CNTCT"],
      merge_tol=0.0001,
      connection_type="MERGE",
      element_series_type="SOLID_185_187",
      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 2.1.31

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 2.1.32

example:

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

SaveMeshModel – see Section 2.1.33

example:

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

SetEdgeParameters – see Section 2.1.34

example:

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

SetGrowthParams – see Section 2.1.35

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 2.1.36

SetMeshingParameters – see Sections 2.1.37

example:

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

SetPrevMeshTool – see Sections 2.1.38

SetStatusFile – see Sections 2.1.39

SetTrimRegions – see Sections 2.1.40

example:

```
f3d.SetTrimRegions(regions=[3])
```

SetUnits – see Sections 2.1.41

example:

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

SetUserExtensionsFile – see Sections 2.1.42

example:

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

SetWorkingDirectory – see Sections 2.1.43; for Windows the path separator is \\

example:

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

SifHistory – see Sections 2.1.44

example:

```
f3d.SifHistory ()
```

StartRecording – writes the subsequent Python commands to a py_session.log file

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 2.1.45

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 2.1.46

example:

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

WriteCrackData – see Sections 2.1.47

example:

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

WriteFatigueData – see Sections 2.1.48

WriteGrowthParams – see Sections 2.1.49

example:

```
f3d.WriteGrowthParams()
```

WriteSERR – see Sections 2.1.50

WriteSif – see Sections 2.1.51

example:

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

WriteSifPath – see Sections 2.1.52

WriteSifHistory – see Sections 2.1.53

WriteStdTempData – see Sections 2.1.54

example:

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

The following methods do not have command line equivalents. They are implemented to support the ABAQUS interface where FRANC3D commands are called directly from ABAQUS CAE.

GetCrackData – returns CrackGrowthData object

example:

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

GetFemModel – returns FemModel object

example:

```
fem = f3d.GetFemModel()
```

GetFemResults – returns FemResults object

example:

```
fem = f3d.GetFemResults()
```

InsertMemFlaw – inserts flaw object

example:

f3d.`InsertMemFlaw(flaw)`

OpenMeshMemModel – processes mesh information into a FemModel object

example:

fem = f3d.`OpenMeshMemModel(fem)`

ReadMemResponse – processes FE results into a FemResults object

example:

f3d.`ReadMemResponse(res)`

3.2.2 class FemModel

The FemModel class stores the FE model data.

constructor:

FemModel() - arguments.

example:

```
fem = PyF3D.FemModel()
```

methods:

Clear – clears the database

example:

```
fem.Clear()
```

AddNode – add a node to the database

parameters:

```
id – integer
coordinates – Vec3D
```

example:

```
fem.AddNode(1,Vec3D.Vec3D(0.0,1.0,2.0))
```

AddElem – add an element to the database

parameters:

```
id – integer
material id – integer
coordinate system id – integer
element type – string
        TET_4
        TET_10
```

PYRAMID_5
PYRAMID_13
WEDGE_6
WEDGE_15
BRICK_8
BRICK_20
BRICK_8

node list – [integer,integer,…]

example:

```
fem.AddElem(1,1,1,"BRICK_8",[1,2,3,4,5,6,7,8])
```

AddMatProps – add a material to database

parameters:

id – integer
mat – integer

example:

```
fem.AddMaterial(1,0)
```

AddBc –

parameters:

id – integer
entity – string
NODE
ELEMENT
NODE_GROUP
ELEMENT_GROUP
face – int (if entity is ELEMENT)
type – string
UX
UY
UZ
FX
FY
FZ
MX
MY
MZ
CPRESS
VPRESS
TRACT
BF
TEMP
value – integer
load case – integer

example:

```
fem.AddBc(1,"NODE","UX",0.0,1)
```

AddNodeSet – add a group of nodes or elements to the database

parameters:

```
label – string
entity list – [integer,integer,…]
```

example:

```
fem.AddNodeSet("test_group",[1,2,4,8])
```

AddElementSet – add a group of nodes or elements to the database

parameters:

```
label – string
entity list – [integer,integer,…]
```

example:

```
fem.AddElementSet("test_group",[1,2,4,8])
```

AddCoordSys – add a coordinate system to the database

parameters:

```
id – integer
angles – Vec3D
```

example:

```
fem.AddCoordSys("test_cs",[0.,90.,0.])
```

AddBcCase – add a load case to the database

parameters:

```
id – integer
```

example:

```
fem.AddBcCase(1)
```

HasNode – tests whether a node exists in the database; returns True or False

parameters:

```
id – integer
```

example:

status = `fem.HasNode(1)`

HasElem – tests whether an element exists in the database; returns True or False

parameters:

```
id – integer
```

example:

```
status = fem.HasElem(1)
```

HasMatProps – tests whether a material exists in the database; returns True or False

parameters:

```
id – integer
```

example:

```
status = fem.HasMat(1)
```

HasBc – tests whether a boundary condition exists in the database; returns True or False

parameters:

```
id – integer
entity – string (see AddBc)
type – string (see AddBc)
```

example:

```
status = fem.HasBc(1,"NODE","UX")
```

HasNodeSet – tests whether a node set exists in the database; returns True or False

parameters:

```
label – string
```

example:

```
status = fem.HasNodeSet("test")
```

HasElementSet – tests whether an element set exists; returns True or False

parameters:

```
label – string
```

example:

```
status = fem.HasElementSet("test")
```

HasCoordSys – tests whether a coordinate system exists; returns True or False

parameters:

```
id – integer
```

example:

```
status = fem.HasCoordSys(1)
```

HasBcCase – tests whether a load case exists in the database; returns True or False

parameters:

```
id – integer
```

example:

status = `fem.HasBcCase(1)`

GetNode – get node from database

parameters:

```
id – integer
```

example:

nd = `fem.GetNode(1)`

GetElem – get element from database

parameters:

```
id – integer
```

example:

el = `fem.GetElem(1)`

GetMatProps – get material properties from database

parameters:

```
id – integer
```

example:

mat = `fem.GetMat(1)`

GetBc – get boundary condition from database

parameters:

```
id – integer
entity – string (see AddBc)
type – string (see AddBc)
```

example:

bc = `fem.GetBc(1,"NODE","UX")`

GetNodeSet – get node set from database

parameters:

```
label – string
```

example:

ns = `fem.GetNodeSet("test")`

GetElementSet – get element set from database

parameters:
`label - string`

example:
es = `fem.GetElementSet("test")`

GetCoordSys – get node from database
parameters:
`id - integer`

example:
cs = `fem.GetCoordSys(1)`

GetNodeIterator – get node iterator
example:
ni = `fem.GetNodeIterator()`

GetElemIterator – get element iterator
example:
ei = `fem.GetElemIterator()`

GetMatPropsIterator – get material iterator
example:
mi = `fem.GetMatPropsIterator()`

GetBcIterator – get boundary condition iterator
example:
bci = `fem.GetBcIterator()`

GetNodeSetIterator – get node set iterator
example:
nsi = `fem.GetNodeSetIterator()`

GetElemSetIterator – get element set iterator
example:
esi = `fem.GetElemSetIterator()`

GetCoordSysIterator – get coordinate system iterator
example:
csi = `fem.GetCoordSysIterator()`

GetBcCaseIterator – get load case iterator
example:
bcsi = `fem.GetBcCaseIterator()`

GetNumNode – returns the total number of nodes
example:
nn = `fem.GetNumNode()`

GetNumElem – returns the total number of elements
example:
ne = `fem.GetNumElem()`

GetNumMatProps – returns the total number of materials
example:
nm = `fem.GetNumMatProps()`

GetNumBc – returns the total number of boundary conditions
example:
nbc = `fem.GetNumBc()`

GetNumNodeSet – returns the total number of node sets
example:
nns = `fem.GetNumNodeSet ()`

GetNumElemSet – returns the total number of element sets
example:
nes = `fem.GetNumElemSet ()`

GetNumCs – returns the total number of coordinate systems
example:
ncs = `fem.GetNumCs()`

3.2.3 class FemResults

The FemResults class stores the FE results data. By default, load case 0 is added when the object is created.

constructor:

FemResults()

example:

```
fres = PyF3D.FemResults()
```

methods:

Clear – clear the database

example:

```
fres.Clear()
```

AddScalarResults – add a scalar result to database

parameters:

```
id – integer
entity – string
        NODE
        ELEMENT
type – string
        TEMP
value – integer
load case – integer
```

example:

```
fres.AddScalarResults(1,”NODE”,”TEMP”,20.0,1)
```

AddVectorResults –

parameters:

```
id – integer
entity – string
        NODE
        ELEMENT
type – string
        DISP
        FORCE
value – Vec3D
load case – integer
```

example:

```
fres.AddVectorResults(1,”NODE”,”DISP”,Vec3D,1)
```

AddScalarList – add a list of scalar results

AddVectorList – add a list of vector results

HasResults – determine if database contains results; returns True or False

parameters:

```
id – integer
entity – string
        NODE
        ELEMENT
type – string
```

TEMP
…
load case – integer

example:
fres.AddHasResults(1,”NODE”,”TEMP”,1)

GetScalarResults – returns scalar result

parameters:
id – integer
entity – string
NODE
ELEMENT
type – string
TEMP
load case – integer

example:
fres.GetScalarResults(1,”NODE”,”TEMP”,1)

GetVectorResults – returns vector results

parameters:
id – integer
entity – string
NODE
ELEMENT
type – string
DISP
FORCE
load case – integer

example:
fres.GetVectorResults(1,”NODE”,”DISP”,1)

GetScalarList – returns list of scalar results

GetVectorList – returns list of vector results

3.2.4 class Flaw

The Flaw class stores parametric flaw data.

constructor:

Flaw()

example:

```
flaw = PyF3D.Flaw()
```

methods:

Save – saves the flaw data to a file

parameters:

```
file_name – string
```

example:

```
flaw.Save("crack_data.crk")
```

3.2.5 class CrackData

The CrackData class stores the crack growth data.

constructor:

CrackData()

example:

```
cgd = PyF3D.CrackData()
```

methods:

Step – returns the list of crack growth steps

example:

```
cgd.Step()
```

StartPoint – returns the crack start point or origin

example:

```
cgd.StartPoint()
```

StartLength – returns the initial crack length

example:

```
cgd.Startlength()
```

3.2.6 class CrackStep

The CrackStep class stores the crack growth data per step.

constructor:

CrackStep()

example:

```
cgs = PyF3D.CrackStep()
```

methods:

Front – returns the crack front

example:

```
cgs.Front()
```

Name –

UserIndx –

ExtensionAmount –

3.2.7 class CrackFront

The CrackFront class stores the crack growth data per front.

constructor:

CrackFront()

example:

```
cgf = PyF3D.CrackFront()
```

methods:

Point – returns list of front points

example:

```
cgf.Point()
```

Id – returns the crack front ID

example:

```
cgf.Id()
```

StartPoint – returns the start point coordinates of the crack front

example:
```
cgf.StartPoint()
```

StopPoint – returns the end point coordinates of the crack front

example:
```
cgf.StopPoint()
```

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

example:
```
cgf.FitFront()
```

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

example:
```
cgf.FitOldFront()
```

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

example:
```
cgf.FitExtensions()
```

ExtensionMult – returns the extension multiplier

example:
```
cgf.ExtensionMult()
```

3.2.8 class FrontPoint

The FrontPoint class stores the crack growth data per crack front point.

constructor:

FrontPoint()

example:
```
fp = PyF3D.FrontPoint()
```

methods:

K – returns the SIF values at the point along the crack front for the first load case

example:

```
fp.K()
```

J – returns the J-integral values at the point along the crack front for the first load case

example:

```
fp.J()
```

T – returns the T-stress value at the point along the crack front for the first load case

example:

```
fp.T()
```

G – returns the G (strain energy release rate) at the point along the crack front for the first load case

example:

```
fp.G()
```

COD – returns the crack opening displacements at the point along the crack front for the first load case

example:

```
fp.COD()
```

Temp – returns the tempaerature at the point along the crack front for the first load case

example:

```
fp.G()
```

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

example:

```
fp.NPos()
```

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

example:

```
fp.Coord()
```

Axes – returns the local axes at the point along the crack front

example:

```
fp.Axes()
```

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

example:

```
fp.KinkAngle()
```

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

example:

```
fp.Extension()
```

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

example:

```
fp.NodeId()
```

ElemId – returns the element ID for the point along the crack front

example:

```
fp.ElemId()
```

4. Python Crack Growth Extensions

The FRANC3D program can be extended in some limited ways 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 subroutine names that FRANC3D currently recognizes and a short description follows:

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 user functions describe 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 crack data.

NumCrackSteps() – returns the number of crack growth steps in the current model.

NumCrackFront(crack_step_num) – returns the number of crack fronts for the given crack step number.

NumCrackPoint(crack_step_num,crack_front_num) – returns the number of crack front points for the given step an crack front.

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.

GetCrackIndexNum() – 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 K's for load step 1 are returned. If no sub step argument is given, the K's 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 G's for load step 1 are returned. If no sub step argument is given, the G's for the final (or only) sub step are returned. None is returned if G's are not available.

GetCOD([step[,sub step]]) – returns crack opening displacements (modes I, II, and III). If no arguments are given, the COD's for load step 1 are returned. If no sub step argument is given, the COD's for the final (or only) sub step are returned. None is returned if COD's 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.

SetCrackIndexNum(index) – used to set the crack 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 θ 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

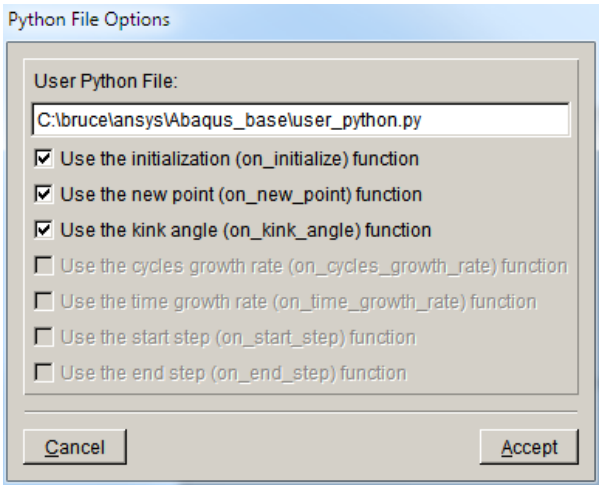
$$\max\left(K_{I}^{r}(q)^{2}+\left(h_{II}K_{II}^{r}(q)\right)^{2}+\left(h_{III}K_{III}^{r}(q)\right)^{2}\right),$$

where K^r is the resolved stress intensity factors in the θ direction.

Vec3D(e1,e2,e3) – is a predefined class that stores a vector of 3 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 give one the option turn on or off defined functions.



4.3 Example user extension files

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

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

import math

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

def hoop_stress(theta,Ksum):

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

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

def on_kink_angle():

    # compute the sum of the K's 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 file 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
#  base 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):

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

# ----------------------------------------------------------------------
# new point function
# ----------------------------------------------------------------------

def on_new_point():

    # compute the sum of the K's 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 exention based on a Paris like model

    extend = 0.001 * Ksum[0]**3.2

    return extend, angle
```