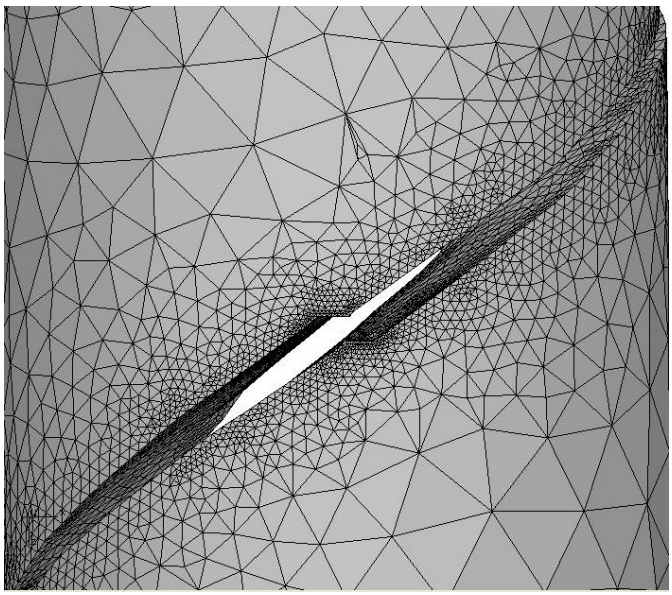
FRANC3D

Command Language & Python Extensions



Version 9.2

Fracture Analysis Consultants, Inc
https://www.fracanalysis.com/

Revised: Aug 2026

Table of Contents:

1. Introduction

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

The PyF3D module works with Python version 3.12. Versions between 8.0 and 8.4 used Python 3.6 – 3.9. If your system does not have version 3.12, you can download (and build) Python yourself (see www.python.org).

Note that the PATH and PYTHONPATH environment variables might need to be set for the Python module 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.

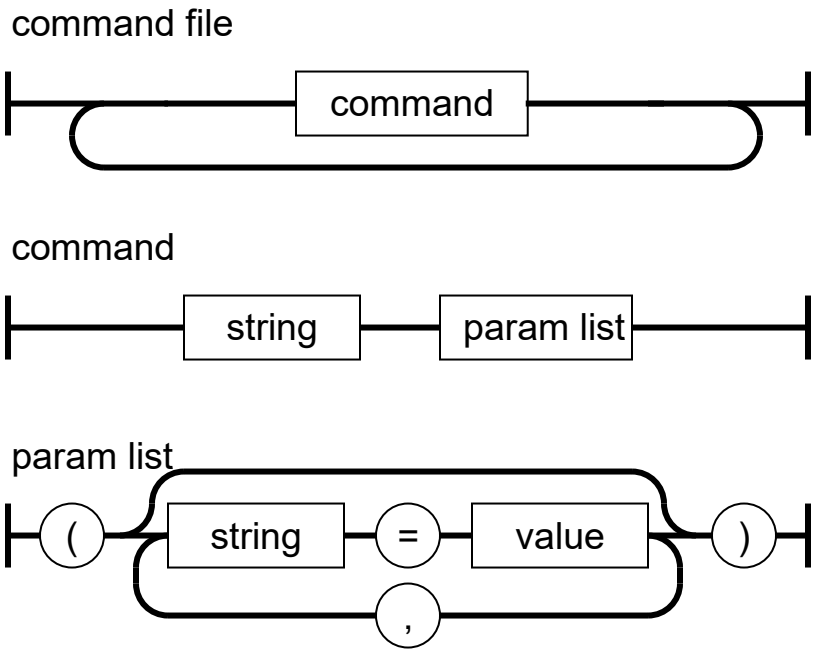
2. Command File Syntax

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.

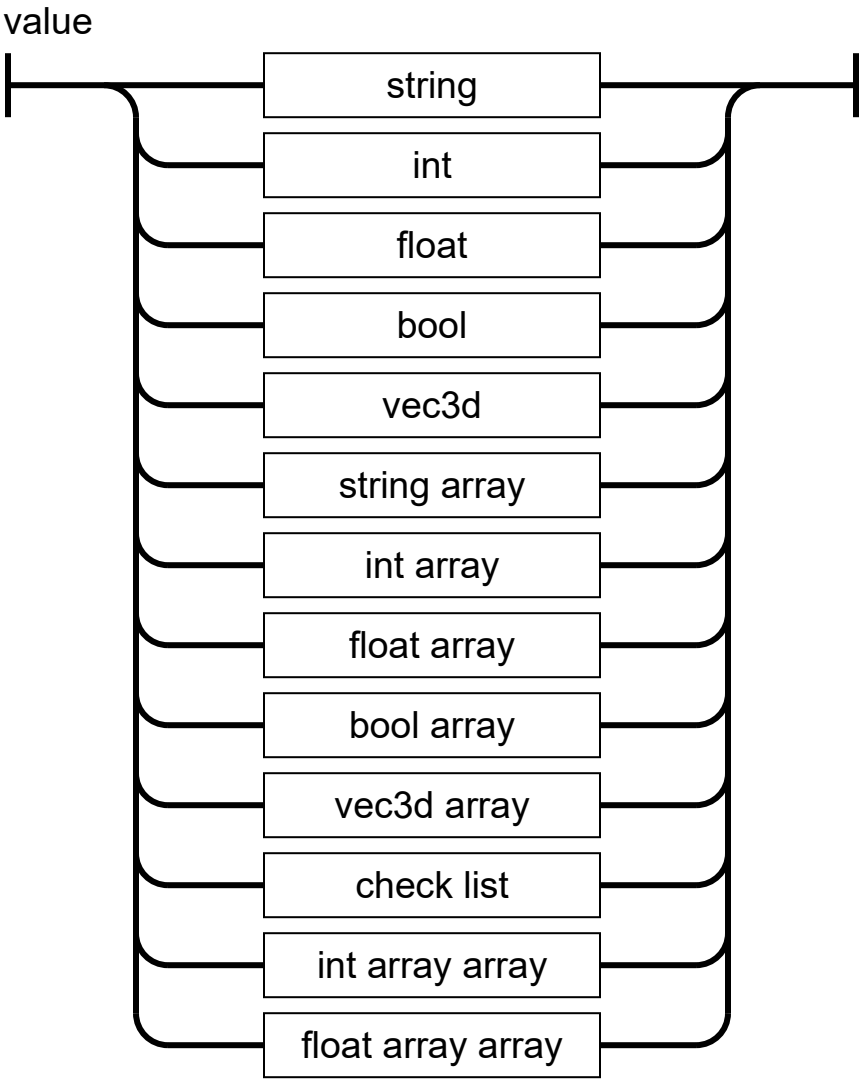
These session files contain a series of command statements. In command files, lines starting with '#' are comment lines. Informally, the command statements look something like:

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

The syntax diagram for a command file is:

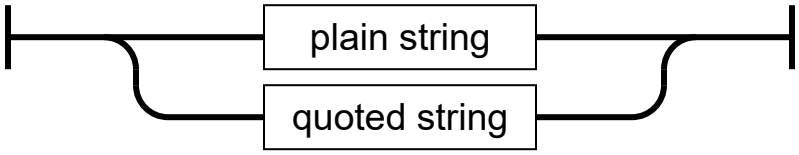


Each parameter value is one of the following types:

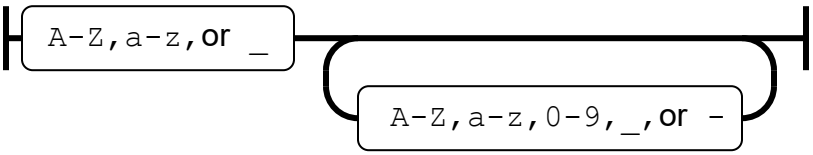


Where the types are defined as:

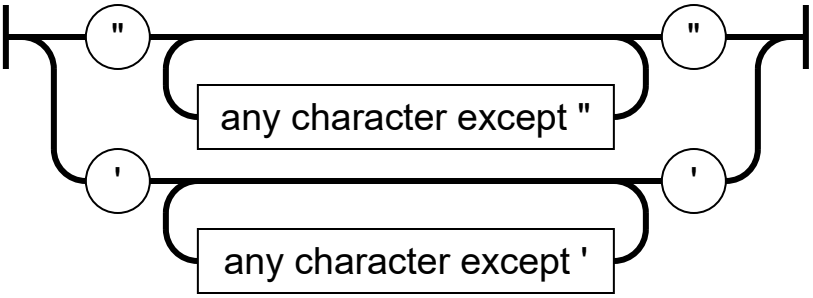
string



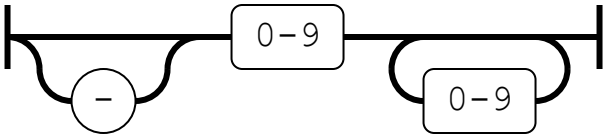
plain string



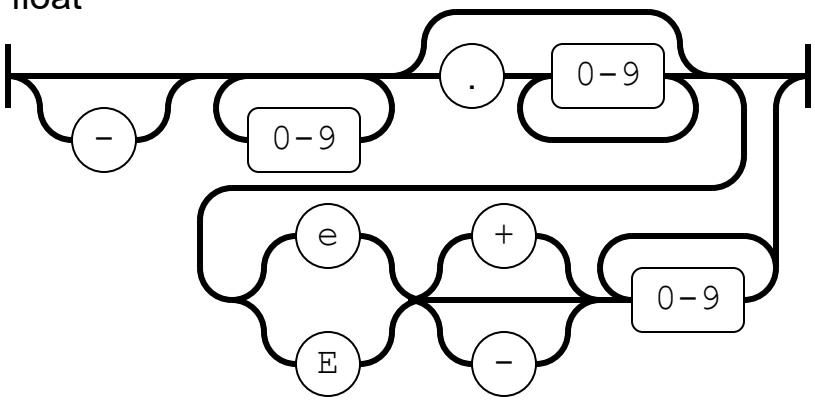
quoted string



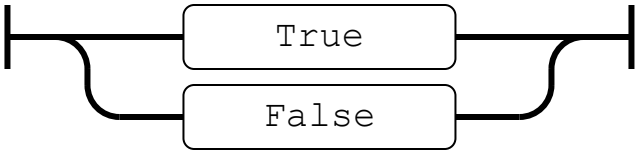
int



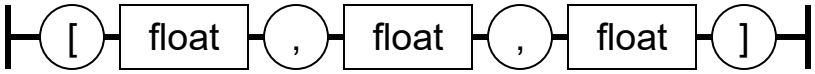
float



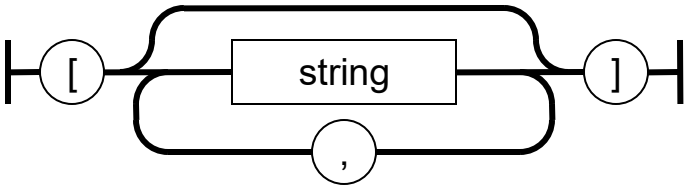
bool



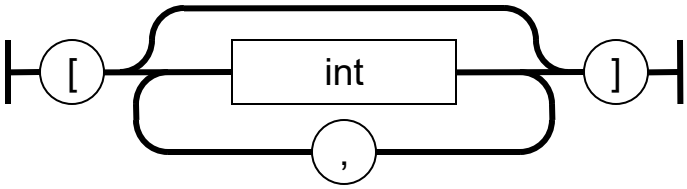
vec3d



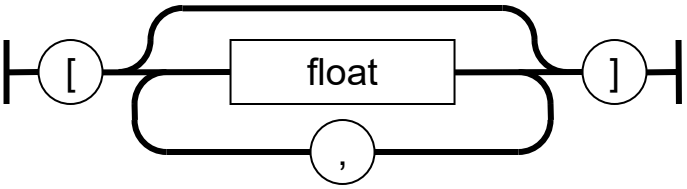
string array



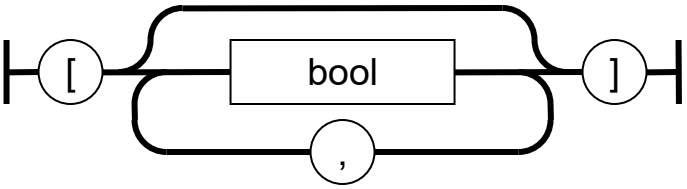
int array



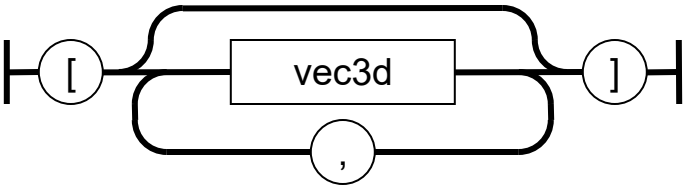
float array



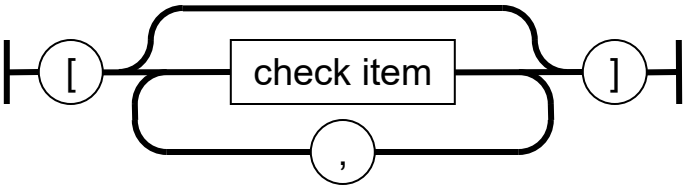
bool array

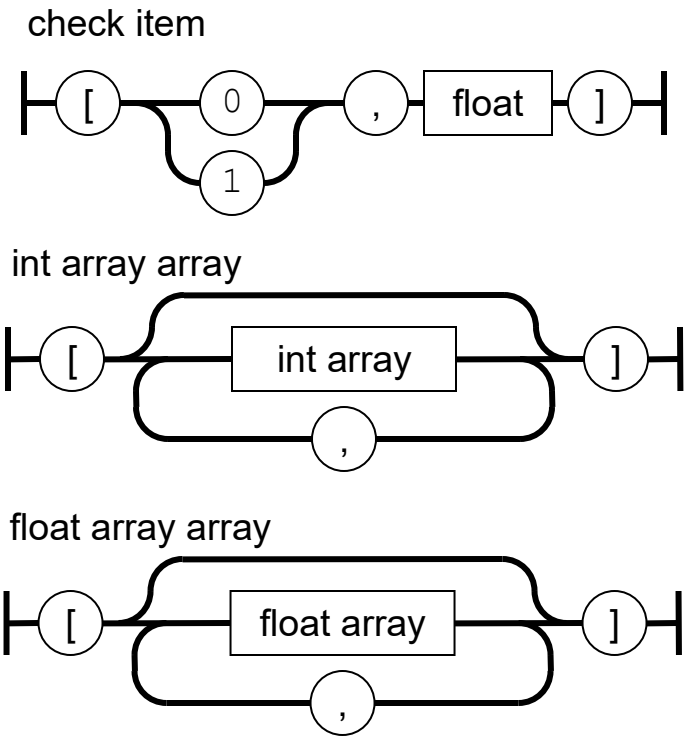


vec3d array



check list





2.1 Commands

The FRANC3D commands 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_params (opt) – see 2.1.1.2
growth_plan (opt) – see 2.1.1.3
growth_variables (opt) – see 2.1.1.4
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
correct_mid_side = *bool* (opt) - correction term if crack face traction is present
large_rotations = *bool* (opt) - correction term for 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
finite_strains = *bool* (opt) – use finite strain for J-integral

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, TIME_EXT, PASSES_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 phi0

kink_angle_strategy = *string* (opt) – crack growth kink angle model
MTS, MSS, GEN, SERR, PLANAR, USER_ANG

kink_angle_parameter = *string* (opt) – non-proportional kink angle parameter
DELTA_K, KMAX

max_kink_angle = *float* (opt) – maximum kink angle allowed (degrees)

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

dyn_pairing_metric = *string* (opt) – dynamic pairing setting
P_DELTA_K, P_DADN

dyn_pairing_hcf_lcf = *string* (opt) – dynamic pairing setting
P_TOTAL, P_SIMPLIFIED

crystallographic_ks = *bool* (opt) – compute crystallographic resolved Ks

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 data

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"]

growth_model=[\+
"XML_FILE: USRMTM_6052_TEMP.xml",\+
"XML_RECORD: 1",\+
"CYCLES_TEMP_INTERP: INTERPOLATE"],
```

2.1.1.2.3 load_step_map

parameters:

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

example:

```
load_step_map=[\+
"VERSION: 1",\+
"MAX_SUB: 5",\+
"0 0",\+
"0 0",\+
"0 0",\+
"0 0",\+
"0 0",\+
"LABELS: 5",\+
"5 Load Step 5",\+
"4 Load Step 4",\+
"3 Load Step 3",\+
"2 Load Step 2",\+
"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
SCONST, LINEAR, SUSER, COMPUTED
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_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 Kmin , 1 is Kmax
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

do_template = *bool* (opt) – use template
temp_radius_type = *string* (opt) – template radius type
ABSOLUTE, RELATIVE
temp_radius = *float* (opt) – template radius
temp_radii = *float* array (opt) – template radii for multiple crack fronts
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_intersection = *bool* (opt) – use simple intersections
front_elem_type = *string* (opt) – element type
WEDGE, COLLAPSED_BRICK, TET
use_qrtr_points = *bool* (opt) – use quarter point elements (for wedge or brick)
constrained_front = *bool* (opt) – constrain collapsed brick element nodes at front
tet_elements = *bool* (opt) – use tetrahedral elements in template
notch_radius = *float* (opt) – notch radius if flaw type is NOTCH
notch_temp_param_type = *string* (opt) – notch template parameter type
RADIUS_AND_NARC, NRING_AND_NARC, NRING_AND_RADIUS
notch_total_radius = *float* (opt) – notch total radius
notch_num_on_arc = *int* (opt) – number of elements around notch
notch_num_rings = *int* (opt) – number of rings of elements around notch
notch_draft_angle = *float* (opt) – notch angle (in degrees)
notch_draft_location = *float* (opt) – notch draft location
model_output = *string* (opt) – Sierra option
WITH_TEMPLATE, WITHOUT_TEMPLATE, TEMPLATE_ONLY

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* (opt) - 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)
ALL_FRAMES - write output for all substeps/iterations of load step
USER_FRAMES - write output for specific frames for abaqus
FULL_OUTPUT - write output for all nodes of full model
TEMPLATE_OUTPUT - write output for just template nodes

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
locglob_contact_adjust = *float* (opt) – amount node positions can be adjusted

locglob_constraint_adjust = *float* (opt) – amount node positions can be adjusted
locglob_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:

locglob_contact_symmetric_pair = *bool* (opt) – create symmetric pair
locglob_contact_as_target = *bool* (opt) –
locglob_contact_mat_id = *int* (opt) – material ID
locglob_contact_mu = *float* (opt) – coefficient of friction
locglob_contact_real_id = *int* (opt) – real constant ID
locglob_contact_real_mod_3 = *float* (opt) – real property
locglob_contact_real_mod_4 = *float* (opt) – real property
locglob_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 local+global base connections:

locglob_constraint_bgset_id = *int* (opt) – ID
locglob_constraint_main_surf_id = *int* (opt) – main surface ID
locglob_constraint_mate_surf_id = *int* (opt) – mate surface ID
locglob_constraint_search_dist = *float* (opt) – search distance
locglob_constraint_ext_factor = *float* (opt) – factor

For crack face contact:

crack_contact_bctset_id = *int* (opt) – set ID
crack_contact_main_surf_id = *int* (opt) – main surface ID
crack_contact_mate_surf_id = *int* (opt) – mate surface ID
crack_contact_friction = *float* (opt) – coefficient of friction
crack_contact_min_dist = *float* (opt) – minimum distance
crack_contact_max_dist = *float* (opt) – maximum distance
crack_contact_offset = *float* (opt) – offset

For extra connections:

constraint_bgset_id = *int* (opt) – ID
constraint_main_surf_id = *int* (opt) – main surface ID
constraint_mate_surf_id = *int* (opt) – mate surface ID
constraint_search_dist = *float* (opt) – search distance
constraint_ext_factor = *float* (opt) – factor

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

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. The status can be retrieved using GetGrowthStatus if using Python.

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 (deprecated)
at_nodes = *bool* (opt) - compute at nodes rather than geometric points (deprecated)

example:

```
ComputeCOD()
```

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 ComputeJRings( )

Compute the J-integral for the current crack front.

parameters:

sif_params => see Section 2.1.1.1: sif_params

example:

```
ComputeJRings(do_epj=true)
```

2.1.7 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)
```

2.1.8 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
CFT_temperature => see Section 2.1.8.2
amplitude = *string* (opt) ABAQUS amplitude function name

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

2.1.8.1 CFT index and load case

All crack surface tractions (CFT) have an internal index. Indices 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.

2.1.8.2 CFT temperature

All crack surface tractions (CFT) have temperature settings; these settings are only active if the model includes temperature.

temperature_source = *string* (opt) – temperature source for traction load step
NONE - no temperature
CONSTANT - constant temperature
PRIOR_LS - temperature from prior load step

DTP_LS - temperature from external .dtp file
MESH_FILE - temperature from external FE (.cdb, .inp, .bdf) file
constant_temperature = *float* (opt) – constant temperature value
external_temp_loadstep = *int* (opt) – load step from external source
temperature_file = *filename* (opt) – external source filename
do_thermal_expansion = *bool* (opt) – has non-zero coefficient of thermal expansion

2.1.9 CrackTractDelete( )

Delete a crack-face traction.

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

example:
```
CrackTractDelete(index=1)
```

2.1.10 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
modal_analysis = *bool* (opt) - stress from modal analysis
second_substep = *int* (opt) - second load substep ID if modal stress
stress_scale = *float* (opt) - external stress factor
load_case = *int* (req) - traction load case
CFT_temperature => see Section 2.1.8.2
amplitude = *string* (opt) ABAQUS amplitude function name

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

2.1.11 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
CFT_temperature => see Section 2.1.8.2
amplitude = *string* (opt) ABAQUS amplitude function name

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

2.1.12 CrackTract1DPoly( )

Define a 1D polynomial crack-face traction distribution.

parameters:
index = *int* (req) - crack traction index
force_dir = *float array* (req) – force direction
variation_dir = *float array* (req) – variation direction
origin = *float array* (req) – origin
coefficients = *float array* (req) – polynomial coefficients
load_case = *int* (req) - traction load case
CFT_temperature => see Section 2.1.8.2
amplitude = *string* (opt) ABAQUS amplitude function name

example:
```
CrackTract1DPoly(index=1,
   force_dir=[1,0,0],
   variation_dir=[0,1,0],
   origin=[5,5,10],
   coefficients=[2,0.2],
   load_case=2)
```

2.1.13 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
CFT_temperature => see Section 2.1.8.2
amplitude = *string* (opt) ABAQUS amplitude function name

example:

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

2.1.14 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
CFT_temperature => see Section 2.1.8.2
amplitude = *string* (opt) ABAQUS amplitude function name

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)
```

2.1.15 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)
```

2.1.16 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_MSWT – 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
FRET_AVRG_NONE – no averaging
FRET_ELEM_CENT – average at element center
FRET_USER_DEFN – average over user-defined length
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)
```

2.1.17 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 fit

fretting_function_id = *int* (req) – function id

- linear=0; linear polynomial fit
- quadratic=1; quadratic polynomial fit
- cubic=2; cubic polynomial fit
- power law=0; nonlinear power law fit
- power1 law=1; nonlinear power law fit
- power2 law=2; nonlinear power law fit
- exponential=3; nonlinear exponential fit

example:

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

2.1.18 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()
```

2.1.19 *GetCrackData*( )

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

2.1.20 *GetIntegrationResults*( )

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

2.1.21 *GetGrowthStatus***( )**

Get the growth status; command is only available from Python.
It returns Normal, Threshold, Critical or Front_Failed. See Section 3.2.

2.1.22 GrowCrack( )

Grow crack front(s).

parameters:

sif_params => see Section 2.1.1.1
growth_vars => see Section 2.1.1.3
template_params => see Section 2.1.1.5
fit_params => see Section 2.1.1.6
file_name = *string (opt)* – file name for saving the crack
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]])
```

2.1.23 GrowCrackFromFile( )

Grow crack from a file.

parameters:

read_files = *list of file names (req)* - file name of new front points
template_params => see Section 2.1.1.5
fit_params => see Section 2.1.1.6
front_indices = *list of front indices (req)* – crack front index
save_file = *single quoted string (opt)* - file name

example:

```
GrowCrackFromFile(temp_radius_type=ABSOLUTE,
   temp_radius=0.05,
   read_files=['abaqus_cube/small_step.frt'],
```

```
front_indices=[0])
```

2.1.24 GrowMergeCrack ( )

Grow and merge coplanar crack fronts.

parameters:

sif_params => see Section 2.1.1.1
growth_vars => see Section 2.1.1.3
template_params => see Section 2.1.1.5
fit_params => see Section 2.1.1.6
file_name = *string (opt)* – file name for saving the crack

example:

```
GrowMergeCrack(sif_method=M_INTEGRAL,
  median_step=0.1,
  temp_radius_type=ABSOLUTE,
  temp_radius=0.05)
```

2.1.25 Include( )

Include a file; only available in batch mode.

parameters:

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

2.1.26 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. Template parameters that differ from the default values should be specified on the command line even if they are already specified in the *.crk* file.

For example, the *.crk* file might use collapsed brick elements at the crack front, specified by: TEMP_FRONT_ELEM_TYPE: COLLAPSED_BRICK. The session log must also define the element type in the command, specified by "front_elem_type=COLLAPSED_BRICK".

parameters:

flaw_insert_params => see Section 2.1.26.1
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)
```

2.1.26.1 flaw_insert_params

parameters:

rotation_axes = *int array* (opt) - ordered list of up to three rotations 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)
template_params => see Section 2.1.1.5
anchor_node = *int* (opt) – node ID for translation
anchor_normal = *vec3d* (opt) – normal to crack surface
node_stress = *float array* (opt) – stress components at anchor node
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"

2.1.27 InsertMergedFlaw( )

Insert a flaw from a *.crk* file into a model that already contains a crack.

file_name = *single quoted string* (req) - name of a flaw (.crk) definition file

2.1.28 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'])
```

2.1.29 InsertMultParamFlaw( )

Insert multiple parameterized flaws.

parameters:

flaw_type = *string array* (req) - list of flaw types:
- CRACK - zero volume crack
- NOTCH - finite volume notch
- VOID - finite volume void

crack_type = *string array* (req if flaw_type=CRACK) - list of crack types:
see Section 2.1.30

notch_type = *string array* (req if flaw_type=NOTCH) - list of notch types:
see Section 2.1.30

void_type = *string array* (req if flaw_type=VOID) - list of void types:
see Section 2.1.30

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

flaw_insert_params *array* => see Section 2.1.26.1; (should use temp_radii)

example:

```
InsertMultParamFlaw(
   flaw_type=[CRACK,CRACK],
   crack_type=[ELLIPSE,ELLIPSE],
   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]],
   temp_radii=[0.02,0.03])
```

2.1.30 InsertParamFlaw( )

Insert a parameterized flaw.

parameters:

flaw_type = *string* (req) - flaw type:
- CRACK - zero volume crack
- NOTCH - finite volume notch
- VOID - finite volume void

crack_type = *string* (req if flaw_type=CRACK) - crack types:

ELLIPSE - elliptical crack
CELLIPSE - curved elliptical crack
THRU - through the thickness crack
CENTER - center crack
LONG - long surface crack
ELONG - long interior crack
RING - double front ring crack
PCIRCLE - portion of a circle
USER - user-defined boundary points crack
UMESH - user-defined mesh crack

notch_type = *string* (req if flaw_type=NOTCH) - void types:
TELLIPSE - thick elliptical flaw
TTHRU - thick through the thickness flaw
TCENTER - thick center flaw
TLONG - thick long surface flaw
TCELLIPSE- thick curved elliptical flaw

void_type = *string* (req if flaw_type=VOID) - void types:
ELLIPSOID - ellipsoidal flaw
UMESH - user-defined mesh void

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

flaw_insert_params => see Section 2.1.26.1

example:

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

2.1.31 InsertUserBdryFlaw( )

Insert a user-defined-boundary flaw. Front_flags=1 indicate that the points are part of the front.

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 2.1.26.1

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)
```

2.1.32 InsertUserMeshFlaw( )

Insert a user-defined-mesh flaw; either a crack or a void can be defined.

parameters:

flaw_type = *string array* (opt) – default is CRACK
mesh_file = *quoted string* (req) - name of a surface mesh file
scaling = *float* (opt) – scale factor
do_reverse = *bool* (opt) – reverse surface mesh normal if true
(only used for symmetry cracks)
flaw_insert_params => see Section 2.1.26.1

example:

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

2.1.33 IntegrateStep( )

Integrates 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 CheckGrowthStatus() or GetGrowthStatus when using Python.

parameters:

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

example:

```
IntegrateStep()
```

2.1.34 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')
```

2.1.35 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')
```

2.1.36 ReadCIData ( )

Read contour integral data from FE analysis output file; either ABAQUS *.dat* or ANSYS *.out*. This command is not available in the Python module.

parameters:

file_name = *single quoted string* (req) – analysis output file name

example:

```
ReadFullGrowthHist(file_name='cracked_model.dat')
```

2.1.37 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')
```

2.1.38 ReadGrowthParams ( )

Read a crack growth parameters file; this command is deprecated.
The SetGrowthParams command should be used instead; see Section 2.1.47.

parameters:

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

example:

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

2.1.39 ReadResponse( )

Read a response file.

parameters:

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

example:

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

2.1.40 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_CFACE_TRACT,NO_CFACE_CNTCT],
   merge_tol=0.0001,
   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])
```

2.1.41 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;
(local cracked portion mesh file)
rslt_file_name = *single quoted string* (req) - file name for the FEM results

(cracked model 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 (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)
```

2.1.42 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')
```

2.1.43 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)
```

2.1.44 SetDisplayUnits( )

The display units for graphs and tables.

parameters:

model_length = *string* (req) – length units
MM, M, INCH
model_stress = *string* (req) – stress units
MPA, PA, PSI, KSI
temperature = *string* (req) – temperature units
C, F, K

2.1.45 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)
```

2.1.46 SetFemUnits( )

The units for the FE model.

parameters:

model_length = *string* (req) – length units
MM, M, INCH
model_stress = *string* (req) – stress units
MPA, PA, PSI, KSI

temperature = *string* (req) – temperature units
C, F, K

2.1.47 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. If you want to change the values from what is in the file, you will need to edit the file itself.

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"])
```

2.1.48 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')
```

2.1.49 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 when moving from a crack front to a nearby surface.

example:

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

2.1.50 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')
```

2.1.51 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])
```

2.1.52 SetWorkingDirectory( )

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

parameters:

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

example:

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

2.1.53 SifHistory( )

Compute 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)
```

2.1.54 *StartRecording*( )

Starts recording the commands to a log file; this command is only available from the Python interface. The output is saved to py_session.log if a file name is not provided.

example:

```
StartRecording()
StartRecording("my_session.log")
```

2.1.55 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"])
```

2.1.56 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
NCRD -	normalized front position
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
T - include T-stress
TEMP - include temperature
CRD - include crack-front Cartesian coordinates
AXES - include crack-front local coordinate axes
REVERSE - reverse the data along the crack front
GI - include mode one energy release rate
GII - include mode two energy release rate
GIII - include mode three energy release rate
COD - include crack opening displacements
CSD - include crack sliding displacements
CTD - include crack tearing displacements
CODCRD - include COD evaluation point coordinates
MODULI - include the elastic modulus at the evaluation points
UNITS - include the units

example:

```
WriteCOD(
   file_name='sif_data.sif',flags=[TAB,KI,TEMP])
```

2.1.57 WriteCrackData( )

Generate a file containing predicted crack growth data. This command is issued from the Advanced menu option: **Export Crack Data**. An *.fcg* file is created and it contains all the growth history.

parameters:

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

example:

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

2.1.58 WriteFatigueData( )

Generate a file containing fatigue data. This command is issued from the Data Table button in the Fatigue dialog. A *.txt* file is written containing the fatigue cycles.

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
PASSES -	include passes/flights
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
AREA -	include crack area
UNITS -	include units

example:

```
WriteFatigueData(
   file_name='ftg_data.txt'
   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")
```

2.1.59 WriteGrowthData( )

Generate a file containing crack-growth data. This command is issued from the Crack Front Plots dialog. A *.txt* file is written containing the data for the current crack front.

parameters:

file_name = *single quoted string* (req) - output file name
growth_params = (req) => see Section 2.1.1.2
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
KTH - include Kthreshold
R - include R (ratio)
TEMP - include Temperature
CRD - include front point coordinates
AXES - include front local axes
REVERSE - output in reverse order
UNITS - include units

example:

```
WriteGrowthData(
   file_name='growth_data.txt'
   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")
```

2.1.60 WriteGrowthPathData( )

Generate a file containing crack-growth data along a path. This command is issued from the Crack Front Plots dialog when using the path option. A *.txt* file is written containing the data for the current crack front.

parameters:

file_name = *single quoted string* (req) - output file name
growth_params = (req) => see Section 2.1.1.2
event = *int* (opt) - ID (index) of the load event (1 .. n)
flags = *string array* (opt) – see Section 2.1.58

2.1.61 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])
```

2.1.62 WriteGrowthRateFile( )

Generate a file containing crack growth rate data. This command is issued from the Growth Rate File menu option.

parameters:

file_name = *single quoted string* (req) - output file name
step = *int* (opt) - ID (index) of the crack step

2.1.63 WriteJInt( )

Generate a file containing J-integral data for either elastic or elasto-plastic material.

parameters:

file_name = *single quoted string* (req) - output file name
flags = *string array* (opt) – see Section 2.1.56

example:

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

2.1.64 WriteResolvedMaxDir( )

Generate a file containing resolved maximum.

2.1.65 WriteResolvedMaxDirPath( )

Generate a file containing resolved maximum along a path.

2.1.66 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
UNITS - include units

example:

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

2.1.67 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 = *string array* (opt) - see Section 2.1.65

example:

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

2.1.68 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])
```

2.1.69 WriteSif( )

Generate a file containing crack-front SIF data. This command is issued from the SIF plot dialog. A *.txt* file is created containing the SIFs along the crack front.

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) – see Section 2.1.56

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)
```

2.1.70 WriteSifPath( )

Generate a file containing SIFs along a path. This command is issued from the SIF Along a Path dialog. A *.txt* file is created containing the SIFs along the path through the crack fronts.

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 = *string array* (opt) – see Section 2.1.56

example:

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

2.1.71 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 9.0

```
SetWorkingDirectory(
   directory='C:\Abaqus_base')

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

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

InsertFileFlaw(
   file_name='Cube_Crack.crk')

RunAnalysis(
   model_type=ABAQUS,
   file_name='AC_crack.fdb',
   flags=[TRANSFER_BC,NO_CFACE_TRACT,NO_CFACE_CNTCT],
   connection_type=MERGE,
   command='"abaqus.bat" job=AC_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=0.075,
    extrapolate=[[3,3]])

AutoGrowth(
    model_type=ABAQUS,
    cur_step=1,
    file_name='AC_crack',
    num_steps=5,
    step_type=SCONST,
    const_step_size=0.1,
    check_Kc=true,
    check_DKth=true,
    maximum_steps=5,
    temp_radius=0.075,
    extrapolate=[[3,3]],
    flags=[TRANSFER_BC,NO_CFACE_TRACT,NO_CFACE_CNTCT],
    connection_type=MERGE,
    command='"abaqus.bat" job=AC_crack_STEP_001 ask_delete=NO -interactive -analysis ')

WriteSifPath(
    file_name='k_vs_a.sif',
    load_step=1,
    flags=[TAB,A,KI,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 use 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 sent 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, CrackData and Maximize modules, which are distributed with the PyF3D module.

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

The PyF3D module is linked to Python libraries, and Version 9.0 is linked with Python 3.12. Versions between 8.0 and 8.4 used Python 3.6 – 3.9. We have a (MS Windows) Version 9.0 that is linked to Python 3.9, 3.10, 3.11 and 3.13; you can look for these in the FAC downloads folder.

A typical Python script starts by importing the "sys" module and appending the path to the *PyF3D.dll* file before importing the 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 along with PYTHONPATH and PYTHONHOME might need to be set.

In MS Windows, 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.

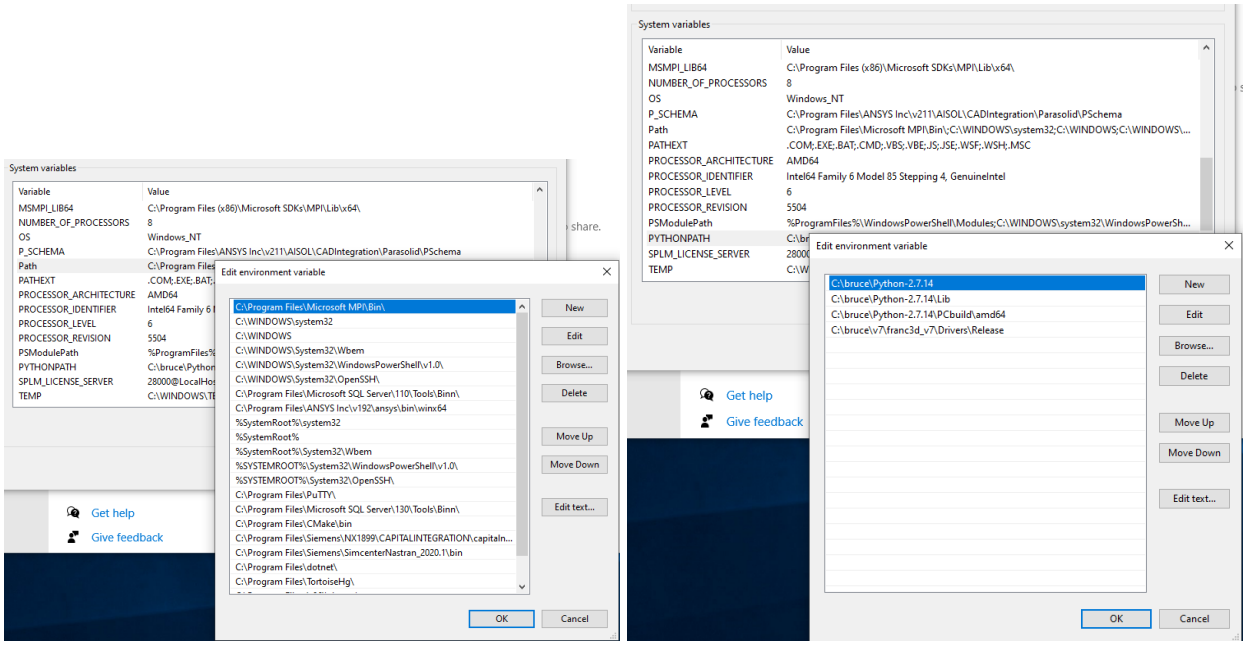


Figure 3.1: PATH and PYTHONPATH system variables.

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 Python3 folder name as needed.

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.

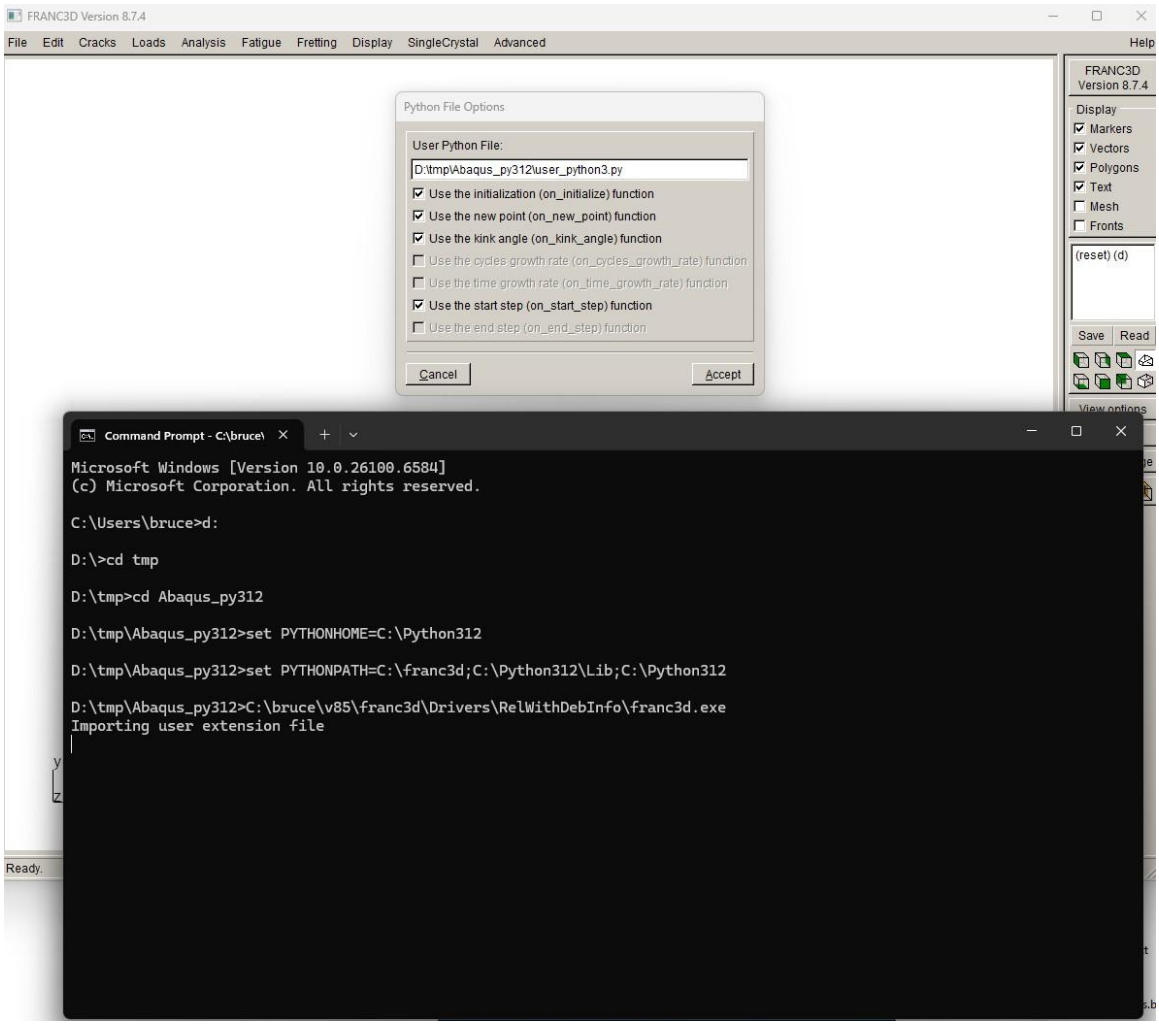


Figure 3.2: PYTHONPATH set from the command line in a CMD window.

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 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/

NOTE: for Linux, you can export PATH with the Python folder first if needed; *i.e.*

export PATH=/home/bruce/Python-3.12/:$PATH

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 are required. You can use "wait_for_license" to allow the app to automatically wait for a license.

example:

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

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,
      merge_tol=0.0001,
      connection_type="MERGE",
      command='"ANSYS252.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"])
```

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")
```

ComputeJRings – see Section 2.1.6

example:

ComputeSif – see Section 2.1.7

example:

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

CrackTractConst – see Section 2.1.8

example:

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

CrackTractDelete – see Section 2.1.9

example:

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

CrackTractExternalDist – see Section 2.1.10

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 2.1.11

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'))
```

CrackTract1DPoly – see Section 2.1.12

example:

```
f3d.CrackTract1DPoly(index=0,
            load_case=2))
```

CrackTract1DRad – see Section 2.1.13

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 2.1.14

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 2.1.15

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 2.1.16

example:

```
f3d.FretNucleationCycles()
```

FretNucleationDataImport – see Section 2.1.17

example:

```
f3d.FretNucleationDataImport()
```

GetBuildInfo – see Section 2.1.18

example:

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

GetCrackData – see Section 2.1.19; returns CrackData object

example:

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

GetIntegrationResults – see Section 2.1.20; returns IntegrationResults object

example:

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

GetGrowthStatus – see Section 2.1.21; returns the GrowthStatus

example:

cgstat = `f3d.GetGrowthStatus()`

GrowCrack – see Section 2.1.22

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 2.1.23

example:

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

GrowMergeCrack – see Section 2.1.24

example:

```
f3d.GrowMergeCrack()
```

Include – see Section 2.1.25

example:

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

InsertFileFlaw – see Section 2.1.26

example:

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

InsertMergedFlaw – see Section 2.1.27

example:

```
f3d.InsertMergedFlaw(
```

InsertMultFileFlaw – see Section 2.1.28

example:

```
f3d.InsertMultFileFlaw(
```

```
            file_names='crack_1.crk','crack_2.crk')
```

InsertMultParamFlaw – see Section 2.1.29

example:

```
f3d.InsertMultParamFlaw(
       flaw_type=["CRACK","CRACK"],
       crack_type=["ELLIPSE","ELLIPSE"],
       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 2.1.30

example:

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

InsertUserBdryFlaw – see Section 2.1.31

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 2.1.32

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 2.1.33

example:

```
f3d.IntegrateStep()
```

OpenFdbModel – see Section 2.1.34

example:

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

OpenMeshModel – see Section 2.1.35

example:

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

ReadCIData– see Section 2.1.36

example:

```
f3d.ReadCIData()
```

ReadFullGrowthHist – see Section 2.1.37

example:

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

ReadGrowthParams – see Section 2.1.38

example:

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

ReadResponse – see Section 2.1.39

example:

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

RunAnalysis – see Section 2.1.40

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 2.1.41

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 2.1.42

example:

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

SaveMeshModel – see Section 2.1.43

example:

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

SetDisplayUnits – see Section 2.1.44

example:

SetEdgeParameters – see Section 2.1.45

example:

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

SetFemUnits – see Section 2.1.46

example:

SetGrowthParams – see Section 2.1.47

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 2.1.48

SetMeshingParameters – see Section 2.1.49

example:

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

SetPreference – command only available in Python.

example:

```
f3d.SetPreference(
      "prog_defs","old_vol_meshing","TRUE")
```

SetStatusFile – see Section 2.1.50; command not available in Python.

SetUserExtensionsFile – see Section 2.1.51

example:

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

SetWorkingDirectory – see Section 2.1.52

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 Section 2.1.53

example:

```
f3d.SifHistory ()
```

StartRecording – writes the subsequent Python commands to a py_session.log file; see Section 2.1.54

example:

```
f3d.StartRecording (file_name='my_session.log')
```

- 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 Section 2.1.55

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 Section 2.1.56

example:

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

WriteCrackData – see Section 2.1.57

example:

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

WriteFatigueData – see Section 2.1.58

WriteGrowthData – see Section 2.1.59

example:

```
f3d.WriteGrowthData()
```

WriteGrowthPathData – see Section 2.1.60

example:

```
f3d.WriteGrowthData()
```

WriteGrowthParams – see Section 2.1.61

example:

```
f3d.WriteGrowthParams()
```

WriteGrowthRateFile – see Section 2.1.62

example:

```
f3d.WriteGrowthParams()
```

WriteJInt – see Section 2.1.63

example:

```
f3d. WriteJInt(file_name=crack_js.dat')
```

WriteResolvedMaxDir – see Section 2.1.64

WriteResolvedMaxDirPath – see Section 2.1.65

WriteResolvedSif – see Section 2.1.66

WriteResolvedSifPath – see Section 2.1.67

WriteSERR – see Section 2.1.68

WriteSif – see Section 2.1.69

example:

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

WriteSifPath – see Section 2.1.70

WriteStdTempData – see Section 2.1.71

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 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`

UserIndx – 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 defines 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) substep 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 is returned. If no sub step argument is given, the temperature for the final (or only) sub step is 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 as weights for mode II and mode III. The maximum shear stress kink criterion is the direction is

$$\max\left(\sqrt{\left(\eta_{II}K^r_{II}(\theta)\right)^2+\left(\eta_{III}K^r_{III}(\theta)\right)^2}\right),$$

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 as weights for 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 as weights for mode II and mode III. The maximum strain energy release rate kink criterion is the direction is

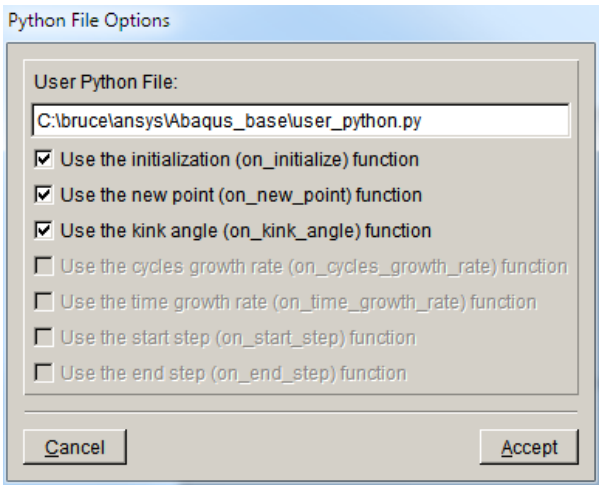
$$\max\left(K^r_I(\theta)^2+\left(\eta_{II}K^r_{II}(\theta)\right)^2+\left(\eta_{III}K^r_{III}(\theta)\right)^2\right),$$

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 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):

    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 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):

    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 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
```