AFLR4 Developer Integration Notes

AFLR4 can be integrated within other systems relatively easily depending on the CAD usage. Integration requires access to and installation of the *AFLR4* developer library package file AFLR4_LIB,*.tar.gz or source package AFLR4_SRC,*.tar.gz (limited access). Installation and setup of developer package files are described in SimSys Developer Install and Setup Instructions (pdf). Please contact David L. Marcum for assistance. Call back functions that allow registering external routines for integration with specific CAD systems and for other tasks are included. *AFLR4* uses the Engineering Geometry Aircraft Design System (*EGADS*) from MIT and *Open CASCADE* from Open CASCADE S.A.S. Both *EGADS* and *Open CASCADE* are freely available as part of the *Engineering Sketch Pad (ESP)* and licensed under The GNU Lesser General Public License, version 2.1. *EGADS* and *Open CASCADE* libraries are required only to use the CAD geometry capability integrated in *AFLR4*. All *AFLR4* package files include the required libraries and headers. An integration library of functions for *EGADS* use is included with *AFLR4*. Integration with other CAD systems requires creation of equivalent functions to replace the built-in functionality. See the last section, **Integration of Alternative CAD Systems**, for more information.

Integration of *AFLR4* for Automated Surface Meshing

AFLR4 integration related tasks are included within the routines located in the src/aflr4/main/ directory. All of these routines are specific to integration of *AFLR4* within a main program or system. They typically serve as simplified APIs of various startup or I/O tasks. All, except the main program file *aflr4.c* have a file name of the form *aflr4_main_xxx.c*. All of these routines are referenced only by other routines in the same directory. They are not included in the *AFLR4* library libaflr4.a (or aflr4.lib or AFLR.dll on Windows) and are instead included in a separate libaflr4_main.a (or aflr4_main.lib on Windows). Some or all of the *aflr4_main_xxx* routines could also be compiled directly with a main program.

Implementation basics are best obtained by viewing the sequence of calls within the main program. In any integration of *AFLR4* the starting point is to setup integration of external routines for a specific CAD system, parallel processing, and file I/O. Default call back functions for registering routines are included in routine *aflr4_main_register*. It should it be called before calling other *AFLR4* routines. Define flags are used to control which external routines are registered. Alternatively, you can simply include the contents of this routine in your code.

Register external functions for AFLR4.

void ***aflr4_main_register*** (void)

Next, the input parameters must be set. Multiple choices are available. The input parameters include parameters that control what *AFLR4* does along with the geometry definition for the given configuration. A reduced set of the most important parameters are described in the next section on **Integration of *AFLR4* for Interactive Use**. It should not be necessary to set other parameters. They are available only for very specific special case uses and for completeness. A description of each input parameter is provided on the AFLR4 Option Details page. Select I/O related input parameters are described in the following.

Input_File_Name

Input CAD or discrete geometry definition file case name or full file name.
Specifies either the case name or full file name for the input CAD file or discrete geometry definition file (surface mesh file). Note that if only a case name is specified, then an input CAD file is searched for first. If no suitable CAD file type is found, then a discrete definition file is searched for.

Output_Grid_File_Name

Output grid file name or suffix.
Specifies either the full file name or file name suffix for the output grid file.
default=".meshb"

Output_File_Flag

Output file flag.
If Output_File_Flag=0, then send all output to standard output or standard error.
If Output_File_Flag=1, then send informational output to both standard output (or standard error) and a file named case_name.program_name.out.
If Output_File_Flag=2, then send maximum informational output to a file named case_name.program_name.out only.
If Output_File_Flag=-1, then send and append informational output to both standard output (or standard error) and a file named case_name.program_name.out.
If Output_File_Flag=-2, then send maximum and append informational output to a file named case_name.program_name.out only.
Error messages will always go to both the file (if any) and standard error.
default=0 min=-2 max=2

Input parameters can be specified using either the *AFLR4* parameter structure *AFLR4_Param_Struct_Ptr* or an argument vector. The main program provided uses the system command line argument vector by default. Input parameters in all cases are specified by a name and a value. Only known parameter names should be specified. Unknown parameter names will produce an error message and a non-zero return error flag when *aflr4_main_setup_input_param* is called. For all input parameter methods, *aflr4_main_setup_input_param* can be called to do input data and parameter structure checking and setup along with startup tasks. Note that the case name used for any output files is derived from the Output_Grid_File_Name. If that is not set or is only a suffix, then the case name is derived from the Input_File_Name. If neither is set, then the case name must be directly set using *ug_set_case_name* ("case_name") prior to calling routine *aflr4_main_setup_input_param*.

Do input data and parameter structure setup and checking along with startup tasks using either the program argument vector or the parameter structure.

INT_ ***aflr4_main_setup_input_param*** (char *argv[], int argc,
UG_Param_Struct
**AFLR4_Param_Struct_Ptr)

INPUT ARGUMENTS

Note: If argc = 0 then argv is ignored and can be NULL.
AFLR4_Param_Struct_Ptr is used and assumed to be initialized and set.
If argc > 0 then argv is used and assumed to be set.
AFLR4_Param_Struct_Ptr may be NULL on input and will be allocated, initialized, and set.

argv Program command line argument vector or equivalent.

argc Program command line argument count or equivalent.

AFLR4_Param_Struct_Ptr *AFLR4* input parameter structure.

RETURN VALUE

0 Normal completion without errors.
>0 An error occurred.

OUTPUT ARGUMENTS

Note: If argc = 0 then AFLR4_Param_Struct_Ptr may include new parameter values.
It will not be reallocated.

AFLR4_Param_Struct_Ptr *AFLR4* input parameter structure.

While the main program provided uses the system command line argument vector, an alternative is to use the *AFLR4* parameter structure to set all input parameters. An example of this mode is shown in the test mode routine *aflr4_main_test_mode_input* for test_input_mode=1. In this mode, character input parameters (Input_File_Name and Output_Grid_File_Name) are set using *ug_set_char_param*.

ug_set_char_param (“name_of_param”, “char_param”, AFLR4_Param_Struct);

And, I/O related integer input parameters (Output_File_Flag) are set using *ug_set_int_param*.

ug_set_int_param (“name_of_param”, param_value, AFLR4_Param_Struct);

Other input parameters are set in a similar manner and are described in the next section on **Integration of *AFLR4* for Interactive Use**.

Another mode of setting the input parameters is to create and set an argument vector with desired input parameters. An example of this method is shown in the test mode routine *aflr4_main_test_mode_input* for test_input_mode=2. When using an argument vector, the parameter structure can be setup by *aflr4_main_setup_input_param*. The example shown in *aflr4_main_test_mode_input* illustrates how to set new argument vector entries using *ug_add_new_arg*, *ug_add_flag_param_arg, ug_add_int_param_arg, ug_add_double_param_arg,* and *ug_add_char_param_arg* routines. The routine *ug_add_new_arg* can be used to allocate and initialize a new argument vector.

Allocate and initialize a new argument vector or add a new argument to an argument vector.

INT_ ***ug_add_new_arg*** (char ***argv, char *new_arg)

INPUT ARGUMENTS

argv	Argument vector.
new_arg	New argument to add to the argument vector. If new_arg = “*allocate_and_initialize_argv*" then allocate a new argument vector with one empty argument. Otherwise set the new_arg string as a new argument vector entry added to the existing argument vector.

RETURN VALUE

0	Normal completion without errors.
>0	An error occurred.

OUTPUT ARGUMENTS

argv New argument vector.

After a new argument vector is created the arguments can be added using the following routines.

ug_add_flag_arg ("flag", argc, argv);

ug_add_int_arg ("name_of_param", param_value, argc, argv);

ug_add_double_arg ("name_of_param", param_value, argc, argv);

ug_add_char_arg ("name_of_param", "char_param", argc, argv);

Note that if input parameter checking and startup tasks are not needed, then alternative methods to calling *aflr4_main_setup_input_param* may be used to set the required *AFLR4* input parameter structure. In this case, *aflr4_setup_param* may be called instead if the argument vector is directly set and used to set input parameters. If the *AFLR4* input parameter structure is directly set and used to set input parameters, then no call is needed to either *aflr4_main_setup_input_param* or *aflr4_setup_param*.

Allocate, initialize and setup the AFLR4 parameter structure.

INT_ ***aflr4_ setup_ param*** (INT_ mmsg_setup, INT_ no_aflr2, int argc, char *argv[], UG_Param_Struct **AFLR4_Param_Struct_Ptr)

INPUT ARGUMENTS

mmsg_setup	Setup message flag. If mmsg_setup = 0 then do not output setup messages If mmsg_setup = 1 then output setup messages
no_aflr2	AFLR2 input parameter flag. If no_aflr2 = 0 then include AFLR2 input parameters. This should always be set to 0 for usage described above If no_aflr2 = 1 then do not include AFLR2 input parameters.
argc	Argument count.
argv	Argument vector.

RETURN VALUE

0	Normal completion without errors.
>0	An error occurred.

<u>OUTPUT ARGUMENTS</u>

AFLR4_Param_Struct_Ptr	*AFLR4* input parameter structure allocated, initialized and set with options found in the argument vector.

Choices are also available for specifying the input data that defines the configuration geometry. By default, this data is input and set from a file specified in the input parameters. Alternatively, other means of setting the data that defines the geometry can be used, such as another file reader or a geometry creation part of the system in which *AFLR4* is being integrated. No examples of such are shown in the main program. For CAD geometry definitions the CAD structure "*model*" must be set with use of an alternative method. And, for discrete definitions "*nbface, nnode, idibf, inibf,* and *x*" must be set with use of an alternative method. In default mode, *AFLR4* internally reads and sets CAD or discrete geometry definition data along with saving it internally using *aflr4_main_data_input*.

Read and set CAD or discrete geometry definition data.

INT_ ***aflr4_main_data_input*** (UG_Param_Struct *AFLR4_Param_Struct_Ptr)

<u>INPUT ARGUMENTS</u>

AFLR4_Param_Struct_Ptr	*AFLR4* input parameter structure.

<u>RETURN VALUE</u>

0	Normal completion without errors.
>0	An error occurred.

If input data is set directly then the geometry definition type must be set directly.

ug_set_int_param ("geom_type", 1, AFLR4_Param_Struct_Ptr); // CAD definition

ug_set_int_param ("geom_type", 2, AFLR4_Param_Struct_Ptr); // discrete definition

Definition data must then be set with either *aflr4_set_ext_cad_data* or *dgeom_set_disc_def_data*.

Set CAD geometry definition data.

INT_ ***aflr4_set_ext_cad_data*** (void *model)

<u>INPUT ARGUMENTS</u>

model CAD geometry definition data structure.

<u>RETURN VALUE</u>

0 Normal completion without errors.
>0 An error occurred.

Set discrete geometry definition data.

INT_ ***dgeom_set_disc_def_data*** (INT_ nbface, INT_ nnode, INT_1D *idibf, INT_3D *inibf,
DOUBLE_3D *x)

<u>INPUT ARGUMENTS</u>

nbface Number of tria-faces for discrete geometry definition.

nnode Number of nodes/vertices for discrete geometry definition.

idibf Tria-face surface ID label (nbface+1 in length) for discrete geometry definition.

inibf Tria-face connectivity (nbface+1 in length) for discrete geometry definition.

x XYZ coordinates (nnode+1 in length) for discrete geometry definition.

<u>RETURN VALUE</u>

0 Normal completion without errors.
>0 An error occurred.

After all input data is setup, the configuration surface mesh can be generated. Routine *aflr4_setup_and_grid_gen* sets up the geometry data, automatic spacing parameters, and generates a surface mesh for the given input data and geometry configuration

Setup geometry data and automatic spacing parameters and generate complete surface grid for given configuration.

INT_ ***aflr4_setup_and_grid_gen*** (UG_Param_Struct *AFLR4_Param_Struct_Ptr);

INPUT ARGUMENTS

AFLR4_Param_Struct_Ptr AFLR4 input parameter structure.

RETURN VALUE

0	Normal completion without errors.
>0	An error occurred.

OUTPUT ARGUMENTS

AFLR4_Param_Struct AFLR4 input parameter data structure with possible changes.

The generated surface mesh and all local parameters are stored internally and can be output to a file using routine *aflr4_main_data_output*.

Write output surface mesh data.

INT_ ***aflr4_main_data_output*** (UG_Param_Struct *AFLR4_Param_Struct_Ptr)

INPUT ARGUMENTS

AFLR4_Param_Struct_Ptr *AFLR4* input parameter structure.

RETURN VALUE

0	Normal completion without errors.
>0	An error occurred.

Note that if the geometry definition is CAD based and aflr4_main_data_output is not called, then you must directly reset the CAD geometry definition data structure model using routine *aflr4_cad_geom_reset*.

aflr4_cad_geom_reset_attr (AFLR4_Param_Struct_Ptr);

Routine *aflr4_get_def* can also be used to retrieve the generated surface mesh.

Get a copy of data arrays for a given surface definition.

INT_ ***aflr4_get_def*** (INT_ idef, INT_ noquad, INT_ *nbface, INT_ *nnode, INT_ *nquad,
INT_1D **ibcibf, INT_1D **idibf, INT_3D **inibf, INT_4D **iniq,
DOUBLE_2D **u, DOUBLE_3D **x)

INPUT ARGUMENTS

idef — ID label for surface definition.

noquad — If there are quad faces and noquad=0 then get them.
If there are quad faces and noquad=1 then replace quad-faces with tria-faces.
If there are no quad faces generated, then noquad is not used.

RETURN VALUE

0 — Normal completion without errors.
>0 — An error occurred.

OUTPUT ARGUMENTS

nbface — Number of tria-faces for generated mesh.

nnode — Number of nodes/vertices for generated mesh.

nquad — Number of quad-faces for generated mesh.

ibcibf — Surface face BC label (nbface+nquad+1 in length) for generated mesh.

idibf — Surface face ID label (nbface+nquad +1 in length) for generated mesh.

inibf — Tria-face connectivity (nbface+1 in length) for generated mesh.

iniq — Quad-face connectivity (nquad+1 in length) for generated mesh.

u — UV coordinates (nnode+1 in length) for generated mesh.
Note that UV coordinates are local to individual surface definitions. For the overall glue-only surface mesh these values are not useful on curves shared between definitions.

x — XYZ coordinates (nnode+1 in length) for generated mesh.

Note that the ID label input argument in routine *aflr4_get_def* for the complete surface mesh (glue-only composite) can be obtained by the following call.

dgeom_def_get_idef (0, &idef);

Integration of *AFLR4* for Interactive Use

AFLR4 can be integrated in an interactive system in which the surface mesh may be generated multiple times for the same configuration. *AFLR4* can be used interactively by calling *aflr4_setup_and_grid_gen* (previously described) multiple times. The generated surface mesh and all local parameters are stored by *aflr4_setup_and_grid_gen* within the dgeom_data_structure and dgeom_def_structure.

Prior to calling *aflr4_setup_and_grid_gen* the CAD or discrete geometry definition data should be input and set using the process outlined in the previous section. In an interactive environment, when little is known about the configuration, leave all input parameters at default values before the first call to *aflr4_setup_and_grid_gen*, except for the automatic farfield grid BC flag and auto-spacing mode flag. The following will turn on automatic determination of the farfield (if any) and turn off surface generation based on curvature and proximity.

ug_set_int_param (“auto_set_ff_bc”, 1, AFLR4_Param_Struct);
ug_set_int_param (“auto_mode”, 0, AFLR4_Param_Struct);

For CAD data files you can also force input attributes to be ignored by setting CAD reset parameter flag.

ug_set_int_param (“cad_param_reset”, 1, AFLR4_Param_Struct);

Note that if the CAD parameter reset option is not set and a *CAD* data file has *AFLR4* attributes (named AFLR_* or AFLR4_*) attached to its Model and/or Faces then those attributes will be used to set the associated *AFLR4* parameters. The initial settings recommended above should produce a surface mesh that is sufficient to visualize the configuration and allow one to set desired parameters interactively. To generate the surface mesh again, set all desired parameters and reset those set o the initial step and then call *aflr4_setup_and_grid_generation*. Note that all data set on the previous generation is cleared, except for the discrete or CAD geometry definition data, when *aflr4_setup_and_grid_generation* is called. The following will reset the automatic farfield grid BC flag, auto-spacing mode flag, and CAD reset parameter (if set) flags to default values.

ug_set_int_param (“auto_set_ff_bc”, 0, AFLR4_Param_Struct);
ug_set_int_param (“auto_mode”, 2, AFLR4_Param_Struct);
ug_set_int_param (“cad_param_reset”, 1, AFLR4_Param_Struct);

Alternatively, you can get the default values directly and then reset the parameter.

ug_get_int_param (“auto_set_ff_bc@def”, &auto_set_ff_bc, AFLR4_Param_Struct);

ug_get_int_param ("auto_mode@def", &auto_mode, AFLR4_Param_Struct);
ug_get_int_param ("cad_param_reset@def", &cad_param_reset, AFLR4_Param_Struct);

ug_set_int_param ("auto_set_ff_bc", auto_set_ff_bc, AFLR4_Param_Struct);
ug_set_int_param ("auto_mode", auto_mode, AFLR4_Param_Struct);
ug_set_int_param ("cad_param_reset", cad_param_reset, AFLR4_Param_Struct);

Note that a farfield can be added to a configuration using the farfield add-on flag and size factor. Set these parameters using *ug_set_int_param* for add_ff_geom and *ug_set_double_param* for ff_size.

add_ff_geom

Farfield add-on flag.
If add_ff_geom=0 do not add a farfield geometry to the configuration.
If add_ff_geom=1 a box-shaped farfield geometry definition is added to the configuration. The farfield box is created with all sides set to length L determined from the configuration bounding-box multiplied by the farfield size factor. Where

$Lx = ff\_size * (X_{max}-X_{min})$
$Ly = ff\_size * (Y_{max}-Y_{min})$
$Lz = ff\_size * (Z_{max}-Z_{min})$
$L = MAX (Lx, Ly, Lz)$

If add_ff_geom=2 then Lx, Ly, Lz are used to create a rectangular box.
Note that the option to automatically set farfield BCs (auto_set_ff_bc=1) is turned on when this option is on (add_ff_geom=1 or 2).
default=0 min=0 max=2

ff_size

Farfield size factor.
default=10

In an interactive environment the following *AFLR4* input parameters would be appropriate to set prior to regenerating the surface mesh with a call to *aflr4_setup_and_grid_generation.* Each of these and are "global" in the sense that they alter the automatic mesh spacing and/or surface meshing processes for all definitions of the overall configuration. The following integer global parameters are set using calls to routine *ug_set_int_param*.

ug_set_int_param ("name_of_param", param_value, AFLR4_Param_Struct);

auto_mode

Auto-spacing mode flag.
If auto_mode = 0 then do not set surface mesh spacing automatically. Note that bounding curve curvature is always used to determine bounding curve spacing.
If auto_mode = 1 then set surface mesh spacing automatically based on curvature.
If auto_mode = 2 then set surface mesh spacing automatically based on curvature and modification of spacing with proximity checking. Proximity of components/bodies to each other is estimated and surface spacing is locally reduced if needed. Proximity checking is automatically disabled if there is only one component/body defined. Note that the goal of proximity checking is that a sufficient number of elements will be generated between surfaces that are close to each other.
default=2 min=0 max=2

auto_set_ff_bc

Automatic farfield grid BC flag.
If auto_set_ff_bc=0 then no surface definitions will be automatically set to a farfield grid BC.
If auto_set_ff_bc=1 then automatic farfield grid BC mode is active and AFLR4 will determine which body is the is the outermost and set the grid BC flag to farfield for all surface definitions of that body. If there is only one body, or a farfield grid BC is set for any surface definition, then automatic farfield grid BC mode is turned off (auto_set_ff_bc=0) and nothing is done.
default=0 min=0 max=1

mer_all

Global edge mesh spacing refinement weight flag.
If mer_all = 0 then do not reduce edge mesh spacing.
If mier4 = 1 then reduce edge mesh spacing based on discontinuity level between adjacent surfaces on both sides of the edge. For each surface, the level of discontinuity (as defined by angerw1 and angerw2) determines the edge spacing refinement weight for potentially reducing the edge spacing. This option is equivalent to setting the edge mesh spacing refinement weight to erw_all for each surface definition. Note that no modification is done to edges that belong to surfaces with a grid BC of farfield or BL intersecting.
default=0 min=0 max=1

mier4

Isolated edge refinement flag.
An isolated interior edge is connected only to boundary nodes. Isolated edges are refined by placing a new node in the middle of the edge.
If mier4 = 0 then do not refine isolated interior edges.
If mier4 = 1 then refine isolated interior edges if the surface has local curvature. Local relative curvature is defined using a factor multiplied by the deviation between the location of a point in the middle of a surface mesh discrete edge and the location of the same point on the actual surface.
If mier4 = 2 then refine all isolated interior edges.
default=1 min=0 max=2

min_ncell

Minimum number of cells between two components/bodies.
Proximity of components/bodies to each other is estimated and surface spacing is locally reduced if needed (see auto_mode). Local surface spacing is selectively reduced when components/bodies are close and their existing local surface spacing would generate less than the minimum number of cells specified by min_ncell. or if there is only one component/body defined.
default=3 min=1 max=2000000000

The following floating-point global parameters are set using calls to routine *ug_set_double_param*.

ug_set_double_param (“name_of_param”, param_value, AFLR4_Param_Struct);

abs_min_scale

Relative scale of absolute minimum spacing to reference length.
The relative scale of absolute minimum spacing to reference length (ref_len) controls the absolute minimum spacing that can be set on any component/body surface by proximity checking. The parameters ref_len, max_scale, min_scale and abs_min_scale are all used to set spacing values on all component/body surfaces (those that are not on farfield or BL intersecting surfaces). Note that the value of abs_min_scale is limited to be less than or equal to min_scale.

*max_spacing = max_scale * ref_len*
*min_spacing = min_scale * ref_len*
*abs_min_spacing = abs_min_scale * ref_len*

default=0.0025 min=1e-12 max=1

max_scale

Relative scale of maximum spacing to reference length.
The relative scale of maximum spacing to reference length (ref_len) controls the maximum spacing that can be set on any component/body surface. The parameters ref_len, max_scale, min_scale and abs_min_scale are all used to set spacing values on all component/body surfaces (those that are not on farfield or BL intersecting surface).

*max_spacing = max_scale * ref_len*
*min_spacing = min_scale * ref_len*
*abs_min_spacing = abs_min_scale * ref_len*

default=0.1 min=1e-12 max=1

min_scale

Relative scale of minimum spacing to reference length.
The relative scale of minimum spacing to reference length (ref_len) controls the minimum spacing that can be set on any component/body surface. The parameters ref_len, max_scale, min_scale and abs_min_scale are all used to set spacing values on all component/body surfaces (those that are not on farfield or BL intersecting surface).

*max_spacing = max_scale * ref_len*
*min_spacing = min_scale * ref_len*
*abs_min_spacing = abs_min_scale * ref_len*

default=0.005 min=1e-12 max=1

ref_len

Reference length for components/bodies in grid units. Reference length should be set to a physically relevant characteristic length for the configuration such as wing chord length or pipe diameter. If ref_len = 0 then it will be set to the bounding box for the largest component/body of interest. The parameters ref_len, max_scale, min_scale and abs_min_scale are all used to set spacing values on all component/body surfaces (those that are not on farfield or BL intersecting surfaces).

*max_spacing = max_scale * ref_len*
*min_spacing = min_scale * ref_len*
*abs_min_spacing = abs_min_scale * ref_len*

default=0 min=0 max=1e+19

ff_cdfr

Farfield growth rate for field point spacing.
The farfield spacing is set to a uniform value dependent upon the maximum size of the domain, maximum size of inner bodies, maximum and minimum body spacing, and farfield growth rate.

*ff_spacing = (ff_cdfr-1)*L+(min_spacing+max_spacing)/2*

where L is the approximate distance between inner bodies and farfield.
default=1.3 min=1 max=10

sf_global

Global surface mesh spacing scale factor.
The surface mesh spacing can be scaled by a global scale factor given by sf_global. With the global scale factor, the calculated spacing is multiplied by the value of sf_global (if it is not equal to 1). Note that the global spacing scale factor sf_global is independent of the surface mesh spacing scale factor that can be set on individual surface definitions.
default=1 min=0.001 max=1000

erw_all

Global edge mesh spacing refinement weight.
Edge mesh spacing can be reduced on all surfaces (if mer_all=1) based on discontinuity level between adjacent surfaces on both sides of the edge. For each surface the level of discontinuity (as defined by angerw1 and angerw2) determines the edge spacing refinement weight for potentially reducing the edge spacing. The edge mesh spacing refinement weight is then used as an interpolation weight between the unmodified spacing and the modified spacing. A value of one applies the maximum modification and a value of zero applies no change in edge spacing. If the global edge mesh spacing refinement weight flag, mer_all, is set to 1 then that is equivalent to setting the edge mesh spacing refinement weight equal to erw_all on all surface definitions. Note that no modification is done to edges that belong to surfaces with a grid BC of farfield or BL intersecting. Also, note that the global weight, erw_all, is not applicable if mer_all=0.
default=0.8 min=0 max=1

BL_thickness

Boundary layer thickness for proximity checking.
Proximity of components/bodies to each other is estimated and surface spacing is locally reduced if needed. Note that if the Reynolds Number, Re_L, is set then the BL_thickness value is set to an estimate for turbulent flow. If the set or

calculated value of BL_thickness>0 then the boundary layer thickness is included in the calculation for the required surface spacing during proximity checking.
default=0 min=0 max=1e+19

Re_l

Reynolds Number for estimating BL thickness.
The Reynolds Number based on reference length, Re_l, (if set) along with reference length, ref_len, are used to estimate the BL thickness, BL_thickness, for turbulent flow. If Re_l>0 then this estimated value is used to set BL_thickness.
default=0 min=0 max=1e+19

In addition to the above global parameters there are multiple parameters that can be set on each individual surface definition. These parameters can only be set after the configuration is defined and registered within the *DGEOM* definition structure and the surface mesh is generated which is done during the first call to *aflr4_setup_and_grid_gen*. Get and set these “local” parameters using the following.

index = -1; // or set to known location value for definition ID idef

ierr = dgeom_def_get_xxx (idef, &index, &xxx);
ierr = dgeom_def_set_xxx (idef, &index, xxx);

where idef is the surface definition ID label, index_ is the definition location (which if it is not known should be set to -1), and xxx is the name of the parameter. Only set the definition location to -1 before the first call for a given surface definition ID. The parameter names are listed below.

ibc

Grid BC value.
For each of the following keywords there is a defined value (listed in src/ug3/UG3_Grid_BC_def.h) for a given face/surface
that is used by both AFLR3 and AFLR4. Predefined AFLR grid BC values are listed below. Set the ibc value equal to one of these keywords. If a grid BC value is not specified for a given surface definition, then it is set to standard BL generating surface grid BC of -STD_UG3_ GBC.

FARFIELD_UG3_GBC	*0*	*farfield surface*
STD_UG3_GBC	*1*	*standard surface*
-STD_UG3_ GBC	*-1*	*standard BL generating surface*
BL_INT_UG3_ GBC	*2*	*symmetry or standard surface that intersects BL*
TRANSP_SRC_UG3_ GBC	*3*	*embedded/transparent surface converted to source nodes*
TRANSP_BL_INT_UG3_ GBC	*4*	*embedded/transparent surface that intersects BL*
TRANSP_UG3_ GBC	*5*	*embedded/transparent surface*
-TRANSP_UG3_ GBC	*-5*	*embedded/transparent BL generating surface*
TRANSP_INTRNL_UG3_ GBC	*6*	*embedded/transparent surface converted to an internal surface coordinates are retained but connectivity is not*
-TRANSP_INTRNL_UG3_ GBC	*-6*	*embedded/transparent BL generating surface converted to an internal set of coordinates that are retained*

Within AFLR4 the grid BC determines how automatic spacing is applied. There are four basic grid BC types that are each treated differently.

1. Surfaces that are part of the farfield should be specified with a FARFIELD_UG3_GBC grid BC. Farfield faces/surfaces are given a uniform spacing independent of other surfaces with different grid BCs.

2. Surfaces that represent standard solid surfaces should be given either a STD_UG3_GBC or -STD_UG3_GBC (BL generating) grid BC. Standard surfaces are given a curvature dependent spacing that may be modified by proximity checking.

3. Surfaces that intersect a BL region should be given either a BL_INT_UG3_GBC or TRANSP_BL_INT_UG3_GBC (transparent surface with volume mesh on both sides) grid BC. A common example for the BL_INT_UG3_GBC grid BC is a symmetry plane. Faces/surfaces set as BL intersecting surfaces are excluded from auto spacing calculations within AFLR4 and use edge spacing derived from their neighbors.

4. Surfaces set as transparent surfaces will have a volume mesh on both sides. They can have free edges and can have non-manifold connections to standard solid surfaces and/or BL intersecting surfaces. Vertices in the final

surface mesh are not duplicated at non-manifold connections. Transparent surfaces use curvature driven surface spacing as used on standard solid surfaces. However, at non-manifold connections with standard solid surfaces they inherit the surface spacing set on the solid surface they are attached to. They are also excluded from proximity checking. Typical examples of transparent surfaces include wake sheets or multi-material interface surfaces. Note that any surface with free edges is automatically set to a TRANSP_UG3_ GBC grid BC

icmp

Component ID identifier for given surface definition.
Component IDs are used for proximity checking. Proximity is only checked between different components. A component is one or more surface definitions that represent a component of the full configuration that should be treated individually. For example, a wing-body-strut-nacelle configuration could be considered as four components with wing surfaces set to component 1, body surfaces set to component 2, nacelle surfaces set to 3, and store surfaces set to 4. If each component is a topologically closed surface/body, then there is no need to set components. If component IDs are not specified, then component identifiers are set for each body defined in an EGADS model or for a discrete definition, each topologically closed surface/bodiy of the overall configuration. Proximity checking is disabled if there is only one component/body defined. Note that proximity checking only applies to standard surfaces. Component identifiers are set by one of three methods, chosen in the following order.

1. If the component identifier, icmp, is set for a definition then it is used.
2. If multiple bodies are defined in an EGADS model, then body index is used to set component identifier.
3. For a discrete definition or an EGADS model with only one body, component identifiers are set to an index based on topologically closed surfaces/bodies of the overall configuration.

mier

Isolated edge refinement flag for given surface definition.
An isolated interior edge is connected only to boundary nodes. Isolated edges are refined by placing a new node in the middle of of the edge.
If mier = 0 then do not refine isolated interior edges on the surface definition.
If mier = 1 then refine isolated interior edges on the surface definition if the surface has local curvature. Local relative curvature is defined using a factor multiplied by the deviation between the location of a point in the middle of a surface mesh discrete edge and the location of the same point on the actual surface.

If mier = 2 then refine all isolated interior edges on the surface definition.
Note that if not set then the isolated edge refinement flag is set to the global value mier4.

erw

Edge mesh spacing refinement weight for given surface definition.
Edge mesh spacing can be reduced on a given surface based on the discontinuity level between adjacent surfaces on both sides of the edge. The edge mesh spacing refinement weight is used as an interpolation weight between the unmodified spacing and the modified spacing. A value of one applies the maximum modification and a value of zero applies no change in edge spacing. Note that no modification is done to edges that belong to a farfield or BL intersecting face/surface.

sf

Surface mesh spacing for given surface definition.
Curvature dependent spacing can be scaled on the surface by the value of the scale factor set. If the scale factor is not set, then the default value is 1.0 (no scaling).

The following global parameters are not typically set by most users. However, they might be exposed and allowed to be set by more advanced users interactively. Set the following parameters using *ug_set_int_param*.

high_order_eval

Discrete geometry high-order evaluation flag.
If high_order_eval = 0 then evaluate discrete geometry using a linear approximation.
If high_order_eval = 1 then evaluate discrete geometry using a high-order approximation.
default=1 min=0 max=1

Set all others that follow using *ug_set_double_param.*

angdbe

Discontinuous boundary edge angle.
Angle between two adjacent boundary edge vectors used to identify edge discontinuities.
default=30 min=0 max=179.9

angerw1

Minimum discontinuous edge angle.
If the angle between the normal vectors for two adjacent faces of two different surface definitions is greater than angerw1 then the edge between them is considered a minimum discontinuity.
default=10 min=0 max=179.9

angerw2

Maximum discontinuous edge angle.
If the angle between the normals for two adjacent faces of two different surface definitions is greater than angerw2 then the edge between them is considered a maximum discontinuity.
default=30 min=0 max=179.9

cdfr

Maximum geometric growth rate.
Used as the advancing-front growth limit. The element size for new nodes is limited to be less than the physical size of the local front advanced from multiplied by cdfr. A cdfr value just above 1.0 will produce a grid with optimal element quality. A value of cdfr well above a value of 1.0 will decrease the number of grid nodes generated and potentially decrease the element quality.
default=1.1 min=1 max=3

curv_factor

Curvature factor.
For surface curvature the spacing is derived from the curvature factor and curvature radius.

Curvature = 1 / Curvature_Radius
Spacing = curv_factor / Curvature

The resulting spacing is limited by the minimum and maximum spacing set by min_scale and max_scale. Note that if curv_factor=0 then surface curvature adjustment is not used.
default=0.1 min=0 max=1e+19

gtol

Relative glue tolerance.
This tolerance is relative to the local discrete edge lengths of the faces or edges attached to the nodes/vertices being considered for gluing.
default=0.0001 min=0 max=1e+19

length_ratio

Curvature length ratio threshold.
The curvature length ratio threshold is used to determine spacing variation for curvature along a curve. The curvature length ratio is defined as:

LR = [L(A,C) + L(C,B)] / L(A,B)

Where LR is the curvature length ratio and A, B, and C are points on the curve with point C approximately at the mid-point between A and B. And, where L(A,B) is the straight line length between A and B, L(A,C) is the straight line length between A and C, and L(C,B) is the straight line length between C and B. Note that LR is always one or more. If LR > length_ratio then the curve is recursively refined. The resulting spacing between points (limited by the minimum spacing set by min_scale) is used to regenerate the edge grid along the curve.
default=1.0001 min=1 max=1.1

Integration of Alternative CAD Systems

As previously mentioned, *AFLR4* includes an integration library of functions for *EGADS* and *Open CASCADE* CAD system functionality. By default, the stand-alone version of *AFLR4* registers and uses these functions for CAD geometry definitions. Integration of an alternative CAD system requires creation of equivalent functions to replace the built-in functionality. Several call back functions are provided with *AFLR4* to facilitate the integration. However, the CAD system specific functions must be created to complete the integration. All functions are registered in the main program and the routines included in src/aflr4/main serve as models for what needs to be integrated within another system to use *AFLR4* routines and register alternative CAD systems. All *EGADS* specific routines are within #ifdef _ENABLE_EGADS_ blocks. In particular, the routine src/aflr4/main/aflr4_main_register.c registers all of the functions required to integrate a CAD system. The routines being registered for *EGADS* usage are the ones that have to be created to integrate another CAD system. A brief description of each follows. All of these *EGADS* specific routines are located within directory src/egads_aflr4 and should be used as a model to guide creation of the routines required for integration of the alternative CAD system.

egads_auto_cad_geom_setup	CAD geometry setup specific to automatic spacing (curve, surface and proximity).
egads_cad_geom_add_ff	Create and add a farfield to geometry. This routine is ***not required*** if this functionality is not needed. If this is the case, then simply do not register a routine.
egads_cad_geom_data_cleanup	Cleanup all CAD system data allocated that is not controlled by *AFLR4*.
egads_cad_geom_file_read	Read a CAD system geometry file, load the model and allocated CAD specific data. This routine is ***not required*** if the system *AFLR4* is being integrated within has the capability to read and/or create the geometry elsewhere.
egads_cad_geom_file_write	Write a CAD system geometry file. This routine is ***not required*** if the system *AFLR4* is being integrated within has the capability to write or otherwise save the geometry elsewhere.
egads_cad_geom_reset_attr	Reset the *AFLR4* attributes that can be attached to the CAD model using data saved in the *AFLR4* parameter structure. Note that

	EGADS has the capability to attach arbitrary attributes for *AFLR4* and other systems. *AFLR4* can use these to setup the parameter structure. This data is also set by default values and an argument vector with specific options. If the alternative CAD system does not support attributes, then all operations and *EGADS* functions related to attributes can be ignored. In this case setup of the *AFLR4* parameter structure would be dependent solely on setting up the argument vector for *AFLR4*.
egads_cad_geom_setup	Setup initial geometry surface definitions within the *AFLR4* -DGEOM surface definition structure. Also, setup topology and extract CAD attributes for *AFLR4* specific data, such as grid BC and surface spacing related data.
egads_set_ext_cad_data	Allocated the *AFLR4* CAD data structure with data required by the CAD system being integrated.
egads_eval_curv_at_uv	Get surface curvature data at a given *U,V* coordinate location for a specified surface definition.
egads_eval_xyz_at_uv	Get *X,Y,Z* coordinates at a given *U,V* coordinate location for a specified surface definition.
egads_eval_uv_bounds	Get *U,V* coordinate bounds for a specified surface definition. Note that if the CAD system uses trimmed surface definitions then the bounds are for the complete surface definition not the trimming curves. The bounds are used to detect singularities in the surface definition
egads_eval_xyz_at_u	Get *X,Y,Z* coordinates at a given *T* coordinate (arc length) location on a specified curve.
egads_eval_edge_uv	Get *U,V* coordinates on a specified surface definition at a given *T* coordinate (arc length) location on a specified curve.
egads_eval_arclen	Get arclength along a specified edge at a given *U,V* coordinate location.

Creation of the routines for integration of an alternative CAD system typically requires a combination of interpreting and understanding each of these routines along with a basic understanding of the API's for called *EGADS* routines (EG_*). A list of the *EGADS* APIs that are called follows.

EG_getTopology	Used for EGADS model, body, face, loop, edge, local_edge, and node.
EG_getBodyTopos	Used for EGADS body with SHELL, FACE, EDGE, and NODE.
EG_indexBodyTopo	Used for EGADS body.
EG_effectiveMap	Used for finding local UV coordinates with Effective Topology.
EG_getBoundingBox	Used for model and face.
EG_getInfo	Used for EGADS edge.
EG_evaluate	Used for EGADS face and edge.
EG_getRange	Used for UV coordinate transformation.
EG_getEdgeUV	Used for EGADS face and edge.
EG_arcLength	Used for EGADS edge.
EG_curvature	Used for EGADS face.
EG_alloc	Used for allocating EGADS data.
EG_free	Used for all EGADS data.
EG_revision	Used to output EGADS model revision.
EG_attributeAdd	Used for EGADS attributes on model, face and edge.
EG_attributeRet	Used for EGADS attributes on model, face and edge.
EG_getContext	Used to add a farfield to an existing EGADS model.
EG_copyObject	Used to add a farfield to an existing EGADS model.
EG_makeSolidBody	Used to add a farfield to an existing EGADS model.
EG_makeTopology	Used to add a farfield to an existing EGADS model.
EG_attributeDup	Used to add a farfield to an existing EGADS model.
EG_close	Used to cleanup EGADS model.
EG_open	Used to read an EGADS file.
EG_loadModel	Used to read an EGADS file.
EG_saveModel	Used to save or write an EGADS file.
EG_deleteObject	Used to delete an existing EGADS model.
EG_getTolerance	Used to create and check an EGADS Tess object.
EG_initTessBody	Used to create and check an EGADS Tess object.
EG_setTessEdge	Used to create and check an EGADS Tess object.
EG_setTessFace	Used to create and check an EGADS Tess object.
EG_statusTessBody	Used to create and check an EGADS Tess object.

A complete description of the *EGADS* system, data, and APIs is contained in the EGADS overall description and specification document (pdf) from MIT. This file along with source code and pre-built binaries can be found at MIT's Engineering Sketch Pad software distribution site.

AFLR4 Home