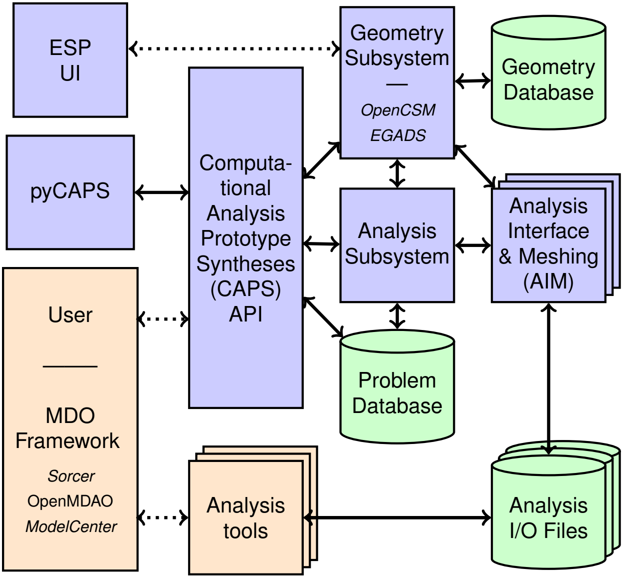
Computational Aircraft Prototype Syntheses: The CAPS API

Part of ESP Revision 1.12

Bob Haimes
haimes@mit.edu
Aerospace Computational Design Lab
Massachusetts Institute of Technology

CAPS Infrastructure in ESP



CAPS Definitions

Problem Object

The Problem is the top-level *container* for a single mission. It maintains a single set of interrelated geometric models, analyses to be executed, connectivity and data associated with the run(s), which can be both multi-fidelity and multidisciplinary. There can be multiple Problems in a single execution of CAPS and each Problem is designed to be *thread safe* allowing for multi-threading of CAPS at the highest level.

Value Object

A Value Object is the fundamental data container that is used within CAPS. It can represent *inputs* to the Analysis and Geometry subsystems and *outputs* from both. Also Value Objects can refer to *mission* parameters that are stored at the top-level of the CAPS database. The values contained in any *input* Value Object can be bypassed by the *linkage* connection to another Value (or *DataSet*) Object of the same *shape*. Attributes are also cast to temporary (*User*) Value Objects.

CAPS Definitions

Analysis Object

The Analysis Object refers to an instance of running an analysis code. It holds the *input* and *output* Value Objects for the instance and a directory path in which to execute the code (though no explicit execution is initiated). Multiple various analyses can be utilized and multiple instances of the same analysis can be handled under the same Problem.

Bound Object

A Bound is a logical grouping of BRep Objects that all represent the same entity in an engineering sense (such as the outer surface of the wing). A Bound may include BRep entities from multiple Bodies; this enables the passing of information from one Body (for example, the aero OML) to another (the structures Body).

Dimensionally:

- 1D – Collection of Edges
- 2D – Collection of Faces

CAPS Definitions

VertexSet Object

A VertexSet is a *connected* or *unconnected* group of locations at which discrete information is defined. Each *connected* VertexSet is associated with one Bound and a single *Analysis*. A VertexSet can contain more than one DataSet. A *connected* VertexSet can refer to 2 differing sets of locations. This occurs when the solver stores its data at different locations than the vertices that define the discrete geometry (i.e. cell centered or non-isoparametric FEM discretizations). In these cases the solution data is provided in a different manner than the geometric.

DataSet Object

A DataSet is a set of engineering data associated with a VertexSet. The rank of a DataSet is the (user/pre)-defined number of dependent values associated with each vertex; for example, scalar data (such as *pressure*) will have rank of one and vector data (such as *displacement*) will have a rank of three. Values in the DataSet can either be deposited there by an application or can be computed (via evaluations, data transfers or sensitivity calculations).

CAPS Objects

Object	SubTypes	Parent Object
capsProblem	Parametric, Static	
capsValue	GeometryIn, GeometryOut, Branch, Parameter, User	capsProblem, capsValue
capsAnalysis		capsProblem
capsValue	AnalysisIn, AnalysisOut	capsAnalysis, capsValue
capsBound		capsProblem
capsVertexSet	Connected, Unconnected	capsBound
capsDataSet	User, Analysis, Interpolate, Conserve, Builtin, Sensitivity	capsVertexSet

Body Objects are EGADS Objects (egos)

CAPS Intent

This value reflects the "Intention" that will be used for a specific analysis. When a geometric entity (a Body) is constructed it should have the Attribute "capsIntent" assigned. The assignment along with the Body Type allows CAPS the ability to filter the Bodies available for any Analysis.

Intent	**Reference**	**Body Type**	**Reference**
0	ALL	0	ALL
32	WAKE	20	NODE
64	STRUCTURE	6	WIREBODY
128	LINEARAERO	7	FACEBODY
256	FULLPOTENTIAL	8	SHEETBODY
512	CFD	9	SOLIDBODY

The Body filtering is performed by a "bit-orred" combination of the intent and the Body type. For example: if you were looking for a BEM you would specify $(SHEETBODY|STRUCTURE)$, which would be 72. If you wanted all SHEETBODYs regardless of analysis it would be $(SHEETBODY|ALL)$, or simply the value 8. And if you wanted any body marked as structure the it would be $(ALL|STRUCTURE)$, 64 in this case.

Other Reserved CAPS Attribute names

capsLength

This string Attribute must be applied to an EGADS Body to indicate the length units used in the geometric construction.

capsBound

This string Attribute must be applied to EGADS BRep Objects to indicate which CAPS Bound(s) are associated with the geometry. A entity can be assigned to multiple Bounds by having the Bound names separated by a semicolon. Face examples could be “Wing”, “Wing;Flap”, “Fuselage”, and etc.

Note: Bound names should not cross dimensional lines.

capsGroup

This string Attribute can be applied to EGADS BRep Objects to assist in grouping geometry into logical sets. A geometric entity can be assigned to multiple groups in the same manner as the capsBound attribute.

Note: CAPS does not internally use this, but is suggested of classifying geometry.

AIMs and query Functions – capsValue Structure

The capsValue Structure is simply the data found within a CAPS Value Object. `aimInputs` and `aimOutputs` must fill the structure with the *type*, *form* and optionally *units* of the data. `aimInputs` also sets the default value(s) in the *vals* member. The structure's members listed below must be filled (most have defaults).

Value Type – no default

The value *type* can be one of:

0	Boolean
1	Integer
2	Double
3	*Character* String
4	*String* Tuple

Shape of the Value – 0 is the default

dim can be one of:

0	scalar only
1	vector or scalar
2	scalar, vector or 2D array

AIMs and query Functions – capsValue Structure

Value Dimensions – 1 is the default

nrow and *ncol* set the dimension of the Value. If both are 1 this has a `scalar` shape. If either *nrow* or *ncol* are one then the shape is `vector`. If both are greater than 1 then this represents a 2D array of values.

Varying Length – the default is “Fixed”

The member *lfixed* indicates whether the length of the Value is allowed to change.

Varying Shape – the default is “Fixed”

The member *sfixed* indicates whether the *shape* of the Value is allowed to change.

Can Value be NULL – the default is “NotAllowed”

The member *nullVal* indicates whether the Value is or can be NULL:
0 – “NotAllowed”, 1 – “NotNull”, 2 – “IsNull”

AIMs and query Functions – capsValue Structure

capsValue Member Usage Notes

- *sfixed* & *dim*
  If the shape is “Fixed” then *nrow* and *ncol* must fit that shape (or a lesser dimension). [Note that the length can change if *lfixed* is “Change”.] If *sfixed* is “Change” then you change *dim* before changing *nrow* and *ncol* to a higher dimension than the current setting.
- *lfixed* & *nrow*/*ncol*
  If the length is “Fixed” then all updates of the Value(s) must match in both *nrow* and *ncol* (which presumes a “Fixed” shape).
- *nullVal* & *nrow*/*ncol*
  nrow and *ncol* should remain at their values even if the Value is NULL to maintain the dimension (and possibly length) when “Fixed”. To indicate a NULL all that is necessary is to set *nullVal* to “IsNull”. The actual allocated storage can remain in the *vals* member or set to NULL.
- Use `EG_alloc` to allocate any memory required for the *vals* member.

CAPS API – Utilities

Open CAPS Problem

```
icode = caps_open(char *name, char *pname, capsObj *problem)
```

name	the input file name – action based on file extension:

*.caps	read the saved CAPS problem file
*.csm	initialize the project using the specified OpenCSM file
*.egads	initialize the project based on the static geometry

pname	the input CAPS problem process name
problem	the returned CAPS problem Object

Set Verbosity Level

```
icode = caps_outLevel(capsObj problem, int outLevel)
```

problem	the CAPS problem object
outLevel	0 - minimal, 1 - standard (default), 2 - debug
icode	the integer return code / old outLevel

Close CAPS Problem

```
icode = caps_close(capsObj problem)
```

problem	the input CAPS problem to close and perform a memory cleanup

CAPS API – Utilities

Save Problem file

```
icode = caps_save(capsObj problem, char *name)
```

problem	the input CAPS problem Object to write
name	the save file name – no extension (added by this function)
icode	the integer return code

Information about an Object

```
icode = caps_info(capsObj object, char **name, enum *type, enum *stype,
                  capsObj *link, capsObj *parent, capsOwn *last)
```

object	the input CAPS Object
name	the returned Object name pointer (if any)
type	the returned data type: Problem, Value, Analysis, Bound, VertexSet, DataSet
stype	the returned subtype (depending on type)
link	the returned linkage Value Object (NULL – no link)
parent	the returned parent Object (NULL for a Problem or an Attribute generated User Value)
last	the returned last owner to *touch* the Object
icode	integer return code

CAPS API – Utilities

Children Sizing info from a Parent Object

```
icode = caps_size(capsObj object, enum type, enum stype, int *size)
```

object	the input CAPS Object
type	the data type to size: Bodies, Attributes, Value, Analysis, Bound, VertexSet, DataSet
stype	the subtype to size (depending on type)
size	the returned size
icode	integer return code

Get Child by Index

```
icode = caps_childByIndex(capsObj object, enum type, enum stype,
                          int index, capsObj *child)
```

object	the input parent Object
type	the Object type to return: Value, Analysis, Bound, VertexSet, DataSet
stype	the subtype to find (depending on type)
index	the index [1-`size`]
child	the returned CAPS Object
icode	integer return code

CAPS API – Utilities

Get Child by Name

```
icode = caps_childByName(capsObj object, enum type, enum stype,
                         char *name, capsObj *child)
```

object	the input parent Object
type	the Object type to return: Value, Analysis, Bound, VertexSet, DataSet
stype	the subtype to find (depending on type)
name	a pointer to the index character string
child	the returned CAPS Object
icode	integer return code

Delete an Object

```
icode = caps_delete(capsObj object)
```

object	the Object to be deleted Note: only Value Objects of subtype User and Bound Objects may be deleted!
icode	integer return code

CAPS API – Utilities

Get Body by index

```
icode = caps_bodyByIndex(capsObj obj, int ind, ego *body, char **unit)
```

obj	the input CAPS Problem or Analysis Object
ind	the index [1-`size`]
body	the returned EGADS Body Object
units	pointer to the string declaring the length units – NULL for unitless values
icode	integer return code

Set Owner Data

```
icode = caps_setOwner(capsObj prob, char *pname, capsOwn *owner)
```

prob	the input CAPS Problem Object
pname	a pointer to the process name character string
owner	a pointer to the CAPS Owner structure to fill
icode	integer return code

Notes: (1) This increases the Problem's sequence number
(2) This does not return the owner pointer, but uses the address to fill
(3) The internal strings can be freed up with `caps_freeOwner`

CAPS API – Utilities

Free Owner Information

```
caps_freeOwner(capsOwn *owner)
```

owner	a pointer to the CAPS Owner structure to free up the members `pname`, `pID` and `user`

Get Owner Information

```
icode = caps_ownerInfo(capsOwn owner, char **pname, char **pID,
                       char **userID, short datetime[6], long *sNum)
```

owner	the input CAPS Owner structure
pname	the returned pointer to the process name
pID	the returned pointer to the process ID
userID	the returned pointer to the user ID
datetime	the filled date/time stamp info [year, month, day, hour, minute, second]
sNum	the sequence number (always increasing)
icode	integer return code

CAPS API – Utilities

Get Error Information

```
icode = caps_errorInfo(capsErrs *errors, int eindex, capsObj *errObj,
                       int *nLines, char ***lines)
```

errors	the input CAPS Error structure
eindex	the index into `error` (1 bias)
errObj	the offending CAPS Object
nLines	the returned number of comment lines to describe the error
lines	a pointer to a list of character strings with the error description
icode	integer return code

Free Error Structure

```
icode = caps_freeError(capsErrs *errors)
```

errors	the CAPS Error structure to be freed
icode	integer return code

CAPS API – Value Objects

Create A Value Object

```
icode = caps_makeValue(capsObj problem, char *vname, enum subtype,
                       enum vtype, int nrow, int ncol, void *data,
                       char *units, capsObj *val)
```

problem	the input CAPS Problem Object where the Value to to reside
vname	the Value Object name to be created
subtype	the Object subtype: Parameter or User
vtype	the value data type:

0	Boolean	2	Double	4	*String* Tuple
1	Integer	3	*Character* String		

nrow	number of rows (not needed for Character Strings)
ncol	number of columns (not needed for strings) – `vlen = nrow * ncol`
data	pointer to the appropriate block of memory must be a pointer to a *capsTuple* structure(s) when `vtype` is a Tuple
units	pointer to the string declaring the units – NULL for unitless values
val	the returned CAPS Value Object
icode	integer return code

CAPS API – Value Objects

Retrieve Values

```
icode = caps_getValue(capsObj val, enum *vtype, int *vlen, void **data,
                      char **units, int *nErr, capsErrs **errs)
```

val	the input Value Object
vtype	the returned data type: 0 Boolean | 2 Double | 4 *String* Tuple 1 Integer | 3 *Character* String | 5 Value *Object*
vlen	the returned value length
data	a filled pointer to the appropriate block of memory (NULL – don't fill) Can use `childByIndex` to get Value Objects
units	the returned pointer to the string declaring the units
nErr	the returned number of errors generated – 0 means no errors
errs	the returned CAPS error structure – NULL with no errors
icode	integer return code

Use the structure *capsTuple* when casting `data` if a Tuple (4)

CAPS API – Value Objects

Reset A Value Object

```
icode = caps_setValue(capsObj val, int nrow, int ncol, void *data)
```

val	the input CAPS Value Object (not for GeometryOut or AnalysisOut)
nrow	number of rows (not needed for Character Strings)
ncol	number of columns (not needed for strings) – `vlen = nrow * ncol`
data	pointer to the appropriate block of memory used to reset the values

Get Valid Value Range

```
icode = caps_getLimits(capsObj val, void **limits)
```

val	the input Value Object
limits	an returned pointer to a block of memory containing the valid range [2*sizeof(`vtype`) in length] – or – NULL if not yet filled

Set Valid Value Range

```
icode = caps_setLimits(capsObj val, void *limits)
```

val	the input Value Object (only for the User & Parameter subtypes)
limits	a pointer to the appropriate block of memory which contains the minimum and maximum range allowed (2 in length)
icode	integer return code

CAPS API – Value Object

Get Value Shape/Dimension

```
icode = caps_getValueShape(capsObj val, int *dim, enum *lfixed,
                           enum *sfixed, enum *ntype,
                           int *nrow, int *ncol)
```

val	the input Value Object
dim	the returned dimensionality: 0 scalar only 1 vector or scalar 2 scalar, vector or 2D array
lfixed	0 – the length(s) can change, 1 – the length is fixed
sfixed	0 – the Shape can change, 1 – Shape is fixed
ntype	0 – NULL invalid, 1 – not NULL, 2 – is NULL
nrow	number of rows – parent index for `Value` vtypes
ncol	number of columns Note: `vlen = nrow * ncol`
icode	integer return code

CAPS API – Value Object

Set Value Shape/Dimension

```
icode = caps_setValueShape(capsObj val, int dim, enum lfixed,
                           enum sfixed, enum ntype)
```

val	the input Value Object (only for the User & Parameter subtypes)
dim	the dimensionality: 0 scalar only 1 vector or scalar 2 scalar, vector or 2D array
lfixed	0 – the length(s) can change, 1 – the length is fixed
sfixed	0 – the Shape can change, 1 – Shape is fixed
ntype	0 – NULL invalid, 1 – not NULL, 2 – is NULL

Units conversion

```
icode = caps_convert(capsObj val, char *units, double in, double *out)
```

val	the reference Value Object
units	the pointer to the string declaring the source units
in	the source value to be converted
out	the returned converted value in the Value Object's units

CAPS API – Value Object

Transfer Values

```
icode = caps_transferValues(capsObj src, enum tmethod, capsObj dst,
                            int *nErr, capsErrs **errs)
```

src	the source input Value Object (not for `Value` or `Tuple` vtypes) – or – DataSet Object
tmethod	0 – copy, 1 – integrate, 2 – weighted average – (1 & 2 only for DataSet `src`)
dst	the destination Value Object to receive the data Notes: • Must not be GeometryOut or AnalysisOut • Shapes must be compatible • Overwrites any Linkage
nErr	the returned number of errors generated – 0 means no errors
errs	the returned CAPS error structure – NULL with no errors
icode	integer return code

CAPS API – Value Object

Establish Linkage

```
icode = caps_makeLinkage(capsObj link, enum tmethod, capsObj trgt)
```

link	linking Value Object (not for `Value` or `Tuple` vtypes or Value subtype User) – or – DataSet Object
tmethod	0 – copy, 1 – integrate, 2 – weighted average – (1 & 2 only for DataSet `link`)
trgt	the target Value Object which will get its data from `link` Notes: • Must not be GeometryOut or AnalysisOut • Shapes must be compatible • `link` = NULL removes any Linkage
icode	integer return code

Note: circular linkages are not allowed!

CAPS API – Attributes

Get Attribute by name

```
icode = caps_attrByName(capsObj object, char *name, capsObj *attr)
```

object	any CAPS Object
name	a string referring to the Attribute name
attr	the returned User Value Object (must be deleted when no longer needed)
icode	integer return code

Get Attribute by index

```
icode = caps_attrByIndex(capsObj object, int in, capsObj *attr)
```

object	any CAPS Object
in	the index (bias 1) to the list of Attributes
attr	the returned User Value Object (must be deleted when no longer needed) Attribute name is the Value Object name
icode	integer return code

Note: The *shape* of the original Value Object is not maintained, but the length is correct.

CAPS API – Attributes

Set an Attribute

```
icode = caps_setAttr(capsObj object, char *name, capsObj attr)
```

object	any CAPS Object
name	a string referring to the Attribute name – NULL: use name in `attr` Note: an existing Attribute of this name is overwritten with the new value
attr	the Value Object containing the attribute The attribute will not maintain the Value Object's *shape*
icode	integer return code

Delete an Attribute

```
icode = caps_deleteAttr(capsObj object, char *name)
```

object	any CAPS Object
name	a string referring to the Attribute to delete NULL deletes all attributes attached to the Object
icode	integer return code

CAPS API – Analysis

Query Analysis – Does not ‘load’ or create an object

```
icode = caps_queryAnalysis(capsObj problem, char *aname,
                           int *nIn, int *nOut, int *execution)
```

problem	a CAPS Problem Object
aname	the Analysis (and AIM plugin) name Note: this causes the the DLL/Shared-Object to be loaded (if not already resident)
nIn	the returned number of Inputs
nOut	the returned number of Outputs
execution	the returned execution flag: 0 – no execution, 1 – AIM performs analysis
icode	integer return code

Free memory in Value Structure

```
caps_freeValue(capsValue *value)
```

value	a pointer to the Value structure to be cleaned up

CAPS API – Analysis

Query Analysis Input Information

```
icode = caps_getInput(capsObj problem, char *aname, int index,
                      char **ainame, capsValue *default)
```

problem	a CAPS Problem Object
aname	the Analysis (and AIM plugin) name
index	the Input index [1-`nIn`]
ainame	a pointer to the returned Analysis Input variable name (use `EG_free` to free memory)
default	a pointer to the filled default value(s) and units – use `caps_freeValue` to cleanup

Query Analysis Output Information

```
icode = caps_getOutput(capsObj problem, char *aname, int index,
                       char **aoname, capsValue *form)
```

problem	a CAPS Problem Object
aname	the Analysis (and AIM plugin) name
index	the Output index [1-`nOut`]
aoname	a pointer to the returned Analysis Output variable name (use `EG_free`)
form	a pointer to the Value Shape & Units information – returned use `caps_freeValue` to cleanup

CAPS API – Analysis

Load Analysis into a Problem

```
icode = caps_load(capsObj problem, char *aname, char *apath,
                  char *unitSys, int intentCombo, int naobj,
                  capsObj *aobjs, capsObj *analysis)
```

problem	a CAPS Problem Object
aname	the Analysis (and AIM plugin) name Note: this causes the the DLL/Shared-Object to be loaded (if not already resident)
apath	the absolute filesystem path to both read and write files this is required even if the AIM does not use the the filesystem, so that the combination of `aname` and `apath` is unique
unitSys	pointer to string describing the unit system to be used by the AIM (can be NULL) see specific AIM documentation for a list of strings for which the AIM will respond
intentCombo	the *Intent* “orred” with the Geometry Type (including wildcards) ignored if the Analysis only supports a single *Intent*
naobj	the number of *parent* Analysis Object(s)
aobjs	a list of the *parent* Analysis Object(s) – may be NULL if `naobj` == 0
analysis	the resultant Analysis Object
icode	integer return code

CAPS API – Analysis

Initialize Analysis from another Analysis Object

```
icode = caps_dupAnalysis(capsObj from, char *apath, int naobj,
                         capsObj *aobjs, capsObj *analysis)
```

from	an existing CAPS Analysis Object
apath	the absolute filesystem path to both read and write files required so that the combination of `aname` and `apath` is unique
naobj	the number of *parent* Analysis Object(s)
aobjs	a list of the *parent* Analysis Object(s) – may be NULL if `naobj` == 0
analysis	the resultant Analysis Object
icode	integer return code

Get Dirty Analysis Object(s)

```
icode = caps_dirtyAnalysis(capsObj object, int *nAobj, capsObj **aobjs)
```

problem	a CAPS Problem, Bound or Analysis Object
nAobjs	the returned number of *dirty* Analysis Objects
aobjs	a returned pointer to the list of *dirty* Analysis Objects (*freeable*)
icode	integer return code

CAPS API – Analysis

Get Info about an Analysis Object

```
icode = caps_analysisInfo(capsObj analysis, char **apath, char **uSys,
                          int *intent, int *naobj, capsObj *aobjs,
                          int *nfields, char ***fnames, int **ranks,
                          int *exec, int *status)
```

analysis	the input Analysis Object
apath	a returned pointer to the string specifying the filesystem path for file I/O
uSys	returned pointer to string describing the unit system used by the AIM (can be NULL)
intent	the returned *Intent* associated with this Analysis Object
naobj	the returned number of *parent* Analysis Object(s)
aobjs	a returned pointer to a list of the *parent* Analysis Object(s)
nfields	the returned number of fields for DataSet filling
fnames	a returned pointer to a list of character strings with the field/DataSet names
ranks	a returned pointer to a list of ranks associated with each field
exec	the returned execution flag: 0 – no execution, 1 – AIM performs analysis
status	0 – up to date, 1 – *dirty* Analysis inputs, 2 – *dirty* Geometry inputs 3 – both Geometry & Analysis inputs are *dirty* , 4 – new geometry, 5 – *post Analysis* required, 6 – Execution & *post Analysis* required

CAPS API – Analysis

Generate Analysis Inputs

```
icode = caps_preAnalysis(capsObj analysis, int *nErr, capsErrs **errs)
```

analysis	the Analysis (or Problem) Object a *Geometry*-only regen is forced when this is a Problem Object
nErr	the returned number of errors generated – 0 means no errors
errs	the returned CAPS error structure – NULL with no errors
icode	integer return code

Mark Analysis as Run

```
icode = caps_postAnalysis(capsObj analysis, capsOwn current, int *nErr,
                          capsErrs **errors)
```

analysis	the Analysis Object Note: this clears all Analysis Output Objects to force reloads/recomputes
current	the CAPS owner structure information for the run
nErr	the returned number of errors generated – 0 means no errors
errors	the returned CAPS error structure – NULL with no errors
icode	integer return code

CAPS API – Analysis

Backdoor AIM Specific Communication

```
icode = caps_AIMbackdoor(capsObj analysis, char *JSONin,
                         char **JSONout)
```

analysis	the Analysis Object
JSONin	a pointer to a character string that AIM function `aimBackdoor` will respond to.
JSONout	a returned pointer to a character string that AIM function `aimBackdoor` creates and passes back as the result to the request (may be *freeable* – depending on the AIM).
icode	integer return code

CAPS API – Analysis Data

Create a Bound – Open until `completeBound`

```
icode = caps_makeBound(capsObj problem, int dim, char *bname,
                       capsObj *bound)
```

problem	a CAPS Problem Object
dim	the dimensionality of the Bound (1 – 3)
bname	the Bound name (matching the *capsBound* Attribute)
bound	the resultant *open* Bound Object
icode	integer return code

Complete a Bound

```
icode = caps_completeBound(capsObj bound)
```

bound	the CAPS Bound Object to close after creating all of the VertexSets & DataSets make calls to `makeVertexSet` and `makeDataSet` in between these 2 functions
icode	integer return code

CAPS API – Analysis Data

Get Information about a Bound

```
icode = caps_boundInfo(capsObj bound, enum *state, int *dim,
                       double *plims)
```

bound	the CAPS Bound Object
state	the returned Bound state: -1 Open 0 Empty & Closed 1 single BRep entity 2 multiple BRep entities -2 multiple BRep entities – Error in reparameterization!
dim	the returned dimensionality of the Bound (1 – 3)
plims	the filled parameterization limits (2 values when `dim` is 1, 4 when `dim` is 2)
icode	integer return code

CAPS API – Analysis Data

Make a VertexSet

```
icode = caps_makeVertexSet(capsObj bound, capsObj analysis,
                           char *vname, capsObj *vset)
```

bound	an input *open* CAPS Bound Object
analysis	the Analysis Object (NULL – Unconnected)
vname	a character string naming the VertexSet (can be NULL for a Connected VertexSet)
vset	the returned VertexSet Object
icode	integer return code

Get Info about a VertexSet

```
icode = caps_vertexSetInfo(capsObj vset, int *nGpts, int *nDpts,
                           capsObj *bound, capsObj *analysis)
```

vset	the VertexSet Object
nGpts	the returned number of *Geometry* points in the VertexSet
nDpts	the returned number of point *Data* positions in the VertexSet
bound	the returned associated Bound Object
analysis	the returned associated Analysis Object (NULL – Unconnected)
icode	integer return code

CAPS API – Analysis Data

Fill an Unconnected VertexSet

```
icode = caps_fillUnVertexSet(capsObj vset, int npts, double *xyzs)
```

vset	the input Unconnected VertexSet Object
npts	the number of points in the VertexSet
xyzs	the point positions (3*`npts` in length)
icode	integer return code

Create a DataSet

```
icode = caps_makeDataSet(capsObj vset, char *dname, enum method,
                         int rank, capsObj *dset)
```

vset	the VertexSet Object – associated Bound must be *open*
dname	a pointer to a string containing the name of the DataSet (i.e., *pressure*)
method	the method used for data transfers: (Sensitivity, Analysis, Interpolate, Conserve, User)
rank	the rank of the data (e.g., 1 – scalar, 3 – vector)
dset	the returned DataSet Object
icode	integer return code

CAPS API – Analysis Data

DataSet Naming Conventions

- Multiple DataSets in a Bound can have the same Name
- Allows for automatic data transfers
- One *source* (from either *Analysis* or *User* Methods)
- Reserved Names:

DSet Name	rank	Meaning	Comments
xyz	3	*Geometry* Positions	
xyzd	3	*Data* Positions	Not for vertex-based discretizations
param*	1/2	t or [u,v] data for *Geometry* Positions	
paramd*	1/2	t or [u,v] for *Data* Positions	Not for vertex-based discretizations
*GeomIn**	3	Sensitivity for the Geometry Input *GeomIn*	can have *[irow, icol]* in name

* Note: not valid for 3D Bounds

CAPS API – Analysis Data

Get Data from a DataSet

```
icode = caps_getData(capsObj dset, int *npts, int *rank,
                     double **data, char **units)
```

dset	the DataSet Object
npts	the returned number of points in the DataSet
rank	the returned rank of the data (e.g., 1 – scalar, 3 – vector)
data	the returned pointer to the data (`rank*npts` in length)
units	the returned pointer to the string declaring the units
icode	integer return code

Get History of a DataSet

```
icode = caps_getHistory(capsObj dset, capsObj *vset, int *nhist,
                        capsOwn **hist)
```

dset	the DataSet Object
vset	the returned associated VertexSet Object
nhist	the returned length of the history list
hist	the returned pointer to the list (`nhist` in length)
icode	integer return code

CAPS API – Analysis Data

Put *User* Data into a DataSet

```
icode = caps_setData(capsObj dset, int nverts, int rank, double *data,
                     char *units)
```

dset	the DataSet Object
nverts	the number of points in data – must match declared `npts`
rank	the rank of the data – must match declared `rank` (e.g., 1 – scalar, 3 – vector)
data	a pointer to the data (`rank*nverts` in length)
units	the pointer to the string declaring the units
icode	integer return code

Get DataSet Objects by Name

```
icode = caps_getDataSets(capsObj bound, char *dname, int *nobj,
                         capsObj **dsets)
```

bound	an input CAPS Bound Object
dname	a pointer to a string containing the name of the DataSet
nobj	the returned number of Objects with the name
dsets	a returned pointer to the list of DataSet Objects (*freeable*)
icode	integer return code

CAPS API – Analysis Data

Get Triangulations for a 2D VertexSet

```
icode = caps_triangulate(capsObj vset, int *nGtris, int **Gtris,
                         int *nDtris, int **Dtris)
```

vset	the input CAPS Connected VertexSet Object
nGtris	the returned number of *Geometry*-based Triangles
Gtris	the returned pointer to a list of indices (bias 1) referencing *Geometry*-based points (3*`nGtris` in length) – *freeable*
nDtris	the returned number of *Data*-based Triangles (0 if discretization is vertex based)
Dtris	the returned pointer to a list of indices (bias 1) referencing *Data*-based points (3*`nDtris` in length) – *freeable*
icode	integer return code

Analysis Interface & Meshing

- Hides all of the individual Analysis details (and peculiarities)
- Dynamically loaded at runtime – extendibility and extensibility

Windows	Dynamically Loaded Libraries (`name.dll`)
LINUX	Shared Objects (`name.so`)
MAC	*Bundles*, CAPS will use the `so` file extension

- An AIM plugin is required for each Analysis code at:
  - a specific *intent*
  - a specific *mode* (i.e., where the inputs may be different)
- Plugin names must be unique – loaded by the name

- † indicates memory handled by CAPS in the following functions i.e., CAPS will free these memory blocks when necessary

Analysis Interface & Meshing – Initialization

Initialization Information for the AIM

```
icode = aimInitialize(int ngIn, capsValue *gIn, int *qeFlg,
                      char *unitSys, int *nIntent, int **intents,
                      int *nIn, int *nOut, int *nFields,
                      char ***fnames, int **ranks)
```

ngIn	the number of *Geometry* Input value structures
gIn	a pointer to the list of *Geometry* Input value structures
qeFlg	on Input: 1 indicates a query and not an analysis instance; on Output: 1 specifies that the AIM executes the analysis
unitSys	a pointer to a character string declaring the unit system – can be NULL
nIntent	the returned number of *intentions* for this AIM requires
intents	the returned pointer to the intents associated with this Analysis †
nIn	the number of Inputs (minimum of 1)*
nOut	the number of possible Outputs*
nFields	the number of fields to responds to for DataSet filling
fnames	a pointer to a list of character strings with the field/DataSet names †
ranks	a pointer to a list of ranks associated with each field †

*nIn & nOut should not depend on the intent

Analysis Interface & Meshing – Support

Discrete Structure

The CAPS *Discrete* data structure holds the spatial discretization information for a Bound. It defines reference positions for the location of the vertices that support the geometry and optionally the positions for the data locations (if these differ). This structure can contain a homogeneous or heterogeneous collection of element types and optionally specifies match positions for conservative data transfers.

Fill-in the Discrete data for a Bound Object– Optional

```
icode = aimDiscr(char *tname, capsDiscr *discr)
```

tname	the Bound name Note: all of the BRep entities are examined for the attribute **capsBound**. Any that match `tname` must be included when filling this `capsDiscr`.
discr	the Discrete structure to fill Note: the AIM *instance*, AIM *info* pointer and the dimensionality have been filled in before this function is invoked.
icode	integer return code

Analysis Interface & Meshing – Support

Frees up data in a Discrete Structure

```
icode = aimFreeDiscr(capsDiscr *discr)
```

discr	the Discrete Structure to have its members freed if NULL, this flags that all internal data stored in the AIM should be cleaned up!
icode	integer return code

Element in the *Mesh* – Optional

```
icode = aimLocateElement(capsDiscr *discr, double *params,
                         double *param, int *eIndex, double *bary)
```

discr	the input Discrete Structure
params	the input global *parametric* space (at all of the *geometry* support positions) rank is the dimensionality (t for 1D, $[u, v]$ for 2D and $[x, y, z]$ for 3D)
param	the input requested parametric position in `params` (dimensionality in length)
eIndex	the returned element index in the `discr` where the position was found (1 bias)
bary	the resultant Barycentric/reference position in the element `eIndex`
icode	integer return code

Analysis Interface & Meshing – Input Prep

Input Information for the AIM

```
icode = aimInputs(int inst, void *aimInfo, int index, char **ainame,
                  capsValue *default)
```

inst	the AIM *instance* index
aimInfo	the AIM context – NULL if called from `caps_getInput`
index	the Input index [1-`nIn`]
ainame	a pointer to the returned Analysis Input variable name
default	a pointer to the filled default value(s) and units – CAPS will free any allocated memory

Parse Input data & Generate Input File(s)

```
icode = aimPreAnalysis(int inst, void *aimInfo, char *apath,
                       capsValue *inputs, capsErrs **errs)
```

inst	the AIM *instance* index
aimInfo	the AIM context (used by the Utility Functions)
apath	the filesystem path where the input file(s) are to be written
inputs	the complete suite of Analysis inputs (`nIn` in length)
errs	a pointer to the returned structure where input error(s) occurred – NULL no errors

Analysis Interface & Meshing – Output Parsing

Output Information for the AIM

```
icode = aimOutputs(int inst, void *aimInfo, int index, char **aonam,
                   capsValue *form)
```

inst	the AIM *instance* index
aimInfo	the AIM context (used by the Utility Functions)
index	the Output index [1-`nOut`]
aonam	a pointer to the returned Analysis Output variable name
form	a pointer to the Value Shape & Units information – to be filled any actual values stored are ignored/freed

Calculate/Retrieve Output Information

```
icode = aimCalcOutput(int inst, void *aimInfo, char *apath, int index,
                      capsValue *val, capsErrs **errors)
```

inst	the AIM *instance* index
aimInfo	the AIM context (used by the Utility Functions)
apath	the filesystem path where the Analysis output file(s) should be read
index	the Output index [1-`nOut`] for this single result
val	a pointer to the capsValue data to fill – CAPS will free any allocated memory
errors	a pointer to the returned error structure where output parsing error(s) occurred NULL with no errors

Analysis Interface & Meshing – Data Transfers

Data Transfer using the Discrete Structure – Optional

```
icode = aimTransfer(capsDiscr *discr, char *name, int npts,
                    int rank, double *data, char **units)
```

discr	the input Discrete Structure
name	the field name to that corresponds to the fill
npts	the number of points to be filled
rank	the rank of the data
data	a pointer associated with the data to be filled (`rank*npts` in length)
units	the returned pointer to the string declaring the units † return NULL to indicate unitless values
icode	integer return code

Analysis Interface & Meshing – Data Transfers

Interpolation on the Bound – Optional

```
icode = aimInterpolation(capsDiscr *discr, char *name, int eIndex,
                         double *bary, int rank, double *data,
                         double *result)
icode = aimInterpolateBar(capsDiscr *discr, char *name, int eIndex,
                          double *bary, int rank, double *r_bar,
                          double *d_bar)
```

discr	the input Discrete Structure
name	a pointer to the input DataSet name string
eIndex	the input target element index (1 bias) in the Discrete Structure
bary	the input Barycentric/reference position in the element `eIndex`
rank	the input rank of the data
data	values at the data (or geometry) positions
result	the filled in results (`rank` in length)
r_bar	input d(objective)/d(result)
d_bar	returned d(objective)/d(data)
icode	integer return code

Forward and *reverse differentiated* functions

Analysis Interface & Meshing – Data Transfers

Element Integration on the Bound – Optional

```
icode = aimIntegration(capsDiscr *discr, char *name, int eIndex,
                       int rank, double *data, double *result)
icode = aimIntegrateBar(capsDiscr *discr, char *name, int eIndex,
                        int rank, double *r_bar, double *d_bar)
```

discr	the input Discrete Structure
name	a pointer to the input DataSet name string
eIndex	the input target element index (1 bias) in `discr`
rank	the input rank of the data
data	values at the data (or geometry) positions – NULL length/area/volume of element
result	the filled in results (`rank` in length)
r_bar	input d(objective)/d(result)
d_bar	returned d(objective)/d(data)
icode	integer return code

Forward and *reverse differentiated* functions

Analysis Interface & Meshing – Data Transfers

Data Transfer to Child AIM – Optional

```
icode = aimData(char *name, enum *vtype, int *rank, int *nrow,
                int *ncol, void **data, char **units)
```

name	the agreed-upon data name to transfer
vtype	value data type – returned
rank	the rank of the data – returned (negative – child should free `data`)
nrow	the number of rows – returned
ncol	the number of columns – returned
data	a void pointer associated with the data – returned
units	the pointer to the string declaring the units (will be free'd by child) – returned

AIM specific Communication – Optional

```
icode = aimBackdoor(int inst, void *aimInfo, char *JSONin,
                    char **JSONout)
```

inst	the AIM *instance* index
aimInfo	the AIM context
JSONin	a pointer to a character string that represents the inputs.
JSONout	a returned pointer to a character string that is the output of the request.

Analysis Interface & Meshing – Utility Library

- provides useful functions for the AIM programmer
- note that all function names begin with `aim_`
- if any of these functions are used, then the library must be included in the AIM so/DLL build

Bodies/Nodes Available through aim_getBodies

1. If NO Bodies/Nodes on the stack have a capsIntent assigned, ALL Bodies/Nodes are provided to the AIM that match the AIM's acceptable list of intention combos (set at `aimInitialize`). `intentCombo` of `caps_load` is ignored and acts as if set to ALL.
2. If Bodies/Nodes on the stack HAVE capsIntent attributes assigned and any wildcard `intentCombo` is specified, CAPS will internally filter the bodies and provide the AIM with Bodies/Nodes that match the intention combinations defined in `aimInitialize`. A warning is raised if no Bodies/Nodes have combinations that match with the expected values for the AIM.
3. If Bodies/Nodes on the stack HAVE capsIntent attributes assigned and a `intentCombo` (not with a wildcard) is provided, CAPS will internally filter the Bodies/Nodes and only supply the AIM with the Bodies/Nodes that match the combination.

Analysis Interface & Meshing – Utility Library

Get Bodies

```
icode = aim_getBodies(void *aimInfo, int *nBody, ego **bodies)
```

aimInfo	the AIM context
nBody	the returned number of EGADS Body Objects that match the `intentCombo`
bodies	the returned pointer to a list of EGADS Body/Node Objects, Tessellation Objects (set by `aim_setTess`) follow (length – 2*nBody)
icode	integer return code

Units conversion

```
icode = aim_convert(void *aimInfo, char *inUnits, double inValue,
                    char *outUnits, double *outValue)
```

aimInfo	the AIM context
inUnits	the pointer to the string declaring the source units
inValue	the value to be converted
outUnits	the pointer to the string declaring the desired units
outValue	the returned converted value
icode	integer return code

Analysis Interface & Meshing – Utility Library

Name to Index conversion

```
icode = aim_getIndex(void *aimInfo, char *name, enum stype)
```

aimInfo	the AIM context
name	the pointer to the string specifying the name to look-up
stype	Value subtype (GEOMETRYIN, ANALYSISIN or ANALYSISOUT)
icode	index (1 bias) or negative integer return code

Index to Name conversion

```
icode = aim_getName(void *aimInfo, int index, enum stype, char **name)
```

aimInfo	the AIM context
index	the index to use (1 bias)
stype	Value subtype (GEOMETRYIN, ANALYSISIN or ANALYSISOUT)
name	the returned pointer to the string specifying the name
icode	integer return code

Analysis Interface & Meshing – Utility Library

Data Transfer from Parent AIM(s)

```
icode = aim_getData(void *aimInfo, char *name, enum *vtype, int *rank,
                    int *nrow, int *ncol, void **data, char **units)
```

aimInfo	the AIM context
name	the requested agreed-upon name to fill
vtype	the returned value data type
rank	the returned rank of the data (negative – `data` should be free'd when done)
nrow	the returned number of rows
ncol	the returned number of columns
data	a returned void pointer associated with the data
units	the returned pointer to the string declaring the units (should be free'd) NULL indicates unitless values
icode	integer return code

Notes: All parent AIMs are queried. If none properly respond, this function returns `CAPS_NOTFOUND`. If multiple parents respond then this function returns `CAPS_SOURCEERR`. Parents must not be *dirty*.

Analysis Interface & Meshing – Utility Library

Establish Linkage from Parent or Geometry

```
icode = aim_link(void *aimInfo, char *name, enum stype,
                 capsValue *default)
```

aimInfo	the AIM context
name	the requested Value Object name to link
stype	Value subtype (GEOMETRYIN, GEOMETRYOUT, ANALYSISIN or ANALSYSOUT)
default	the pointer from `aimInputs`
icode	integer return code

Note: For ANALYSISIN or ANALYSISOUT subtypes all parent Analyses are queried. If none is found in the parent hierarchy, this function returns `CAPS_NOTFOUND`. The query is performed from the *oldest* ancestor down. The first match is used.

Analysis Interface & Meshing – Utility Library

Get Geometry State WRT the Analysis

```
icode = aim_newGeometry(void *aimInfo)
```

aimInfo	the AIM context
icode	CAPS_SUCCESS for new, CAPS_CLEAN if not regenerated since last here

Set Tessellation for a Body

```
icode = aim_setTess(void *aimInfo, ego object)
```

aimInfo	the AIM context
object	the EGADS Tessellation Object to use for the associated Body –or – the Body Object to remove and delete an existing tessellation Note that *the Body Object is part of the Tessellation Object*
icode	integer return code

An error is raised when trying to set a Tessellation Object when one exists.

If the Problem is STATIC then the AIM (or CAPS application) is responsible for deleting the Tessellation Object. Otherwise removal of the Tessellation Object is controlled internally during Body operations. If a Tessellation Object is removed (no longer associated with the Body) then CAPS deletes the Tessellation Object.

Analysis Interface & Meshing – Utility Library

Get Discretization Structure

```
icode = aim_getDiscr(void *aimInfo, char *bname, capsDiscr **discr)
```

aimInfo	the AIM context
bname	the Bound name
discr	pointer to the returned Discrete structure
icode	integer return code

Get Data from Existing DataSet

```
icode = aim_getDataSet(capsDiscr *discr, char *dname, enum *method,
                       int *npts, int *rank, double **data)
```

discr	the input Discrete Structure
dname	the requested DataSet name
method	the returned method used for data transfers
npts	the returned number of points in the DataSet
rank	the returned rank of the DataSet
data	a returned pointer to the data within the DataSet
icode	integer return code

Analysis Interface & Meshing – Utility Library

Get Bound Names

```
icode = aim_getBounds(void *aimInfo, int *nBname, char ***bnames)
```

aimInfo	the AIM context
nBname	returned number of Bound names
tnames	returned pointer to list of Bound names (freeable)

Get Unit System

```
icode = aim_unitSys(void *aimInfo, char **unitSys)
```

aimInfo	the AIM context
unitSys	a returned pointer to a character string declaring the unit system – can be NULL
icode	integer return code

Analysis Interface & Meshing – Utility Library

Setup for Sensitivities

```
icode = aim_setSensitivity(void *aimInfo, char *GIname, int *irow,
                           int *icol)
```

aimInfo	the AIM context
GIname	the pointer to the string that matches the *Geometry Input* Parameter name
irow	the parameter row to use – 1 bias
icol	the parameter column to use – 1 bias

Notes: (1) `aim_setTess` must have been invoked sometime before calling this function to set the tessellations for the Bodies of interest.

(2) Call `aim_setSensitivity` before call(s) to `aim_getSensitivity`.

Analysis Interface & Meshing – Utility Library

Get Sensitivities based on Tessellation Components

```
icode = aim_getSensitivity(void *aimInfo, ego tess, int ttype,
                           int index, int *npts, double **dxyz)
```

aimInfo	the AIM context
tess	the EGADS Tessellation Object
ttype	topological type – 0 - NODE, 1 - EDGE, 2 - FACE *Configuration Sensitivities* – -1 - EDGE, -2 - FACE
index	the index in the Body (associated with the tessellation) based on the *type*
npts	the returned number of sensitivities (number of tessellation points)
dxyz	a pointer to the returned sensitivities – 3*npts in length (*freeable*)
icode	integer return code

Note: Call `aim_setSensitivity` before call(s) to `aim_getSensitivity`.

Analysis Interface & Meshing – Utility Library

Get Global Tessellation Sensitivities

```
icode = aim_sensitivity(void *aimInfo, char *GIname, int irow,
                        int icol, ego tess, int *npts, double **dxyz)
```

aimInfo	the AIM context
GIname	the pointer to the string that matches the *Geometry Input* Parameter name
irow	the parameter row to use – 1 bias
icol	the parameter column to use – 1 bias
tess	the EGADS Tessellation Object
npts	the returned number of sensitivities (number of global vertices)
dxyz	a pointer to the returned sensitivities – 3*npts in length (*freeable*)
icode	integer return code

Note: Used to get the tessellation sensitivities for the entire Tessellation Object. The number of points is the global number of vertices in the tessellation.

CAPS Return Codes

CAPS_SUCCESS	0	CAPS_CIRCULARLINK	-319
CAPS_BADRANK	-301	CAPS_UNITERR	-320
CAPS_BADDSETNAME	-302	CAPS_NULLBLIND	-321
CAPS_NOTFOUND	-303	CAPS_SHAPEERR	-322
CAPS_BADINDEX	-304	CAPS_LINKERR	-323
CAPS_NOTCHANGED	-305	CAPS_MISMATCH	-324
CAPS_BADTYPE	-306	CAPS_NOTPROBLEM	-325
CAPS_NULLVALUE	-307	CAPS_RANGEERR	-326
CAPS_NULLNAME	-308	CAPS_DIRTY	-327
CAPS_NULLOBJ	-309	CAPS_HIERARCHERR	-328
CAPS_BADOBJECT	-310	CAPS_STATEERR	-329
CAPS_BADVALUE	-311	CAPS_SOURCEERR	-330
CAPS_PARAMBNDERR	-312	CAPS_EXISTS	-331
CAPS_NOTCONNECT	-313	CAPS_IOERR	-332
CAPS_NOTPARMTRIC	-314	CAPS_DIRERR	-333
CAPS_READONLYERR	-315	CAPS_NOTIMPLEMENT	-334
CAPS_FIXEDLEN	-316	CAPS_EXECERR	-335
CAPS_BADNAME	-317	CAPS_CLEAN	-336
CAPS_BADMETHOD	-318	CAPS_BADINTENT	-337