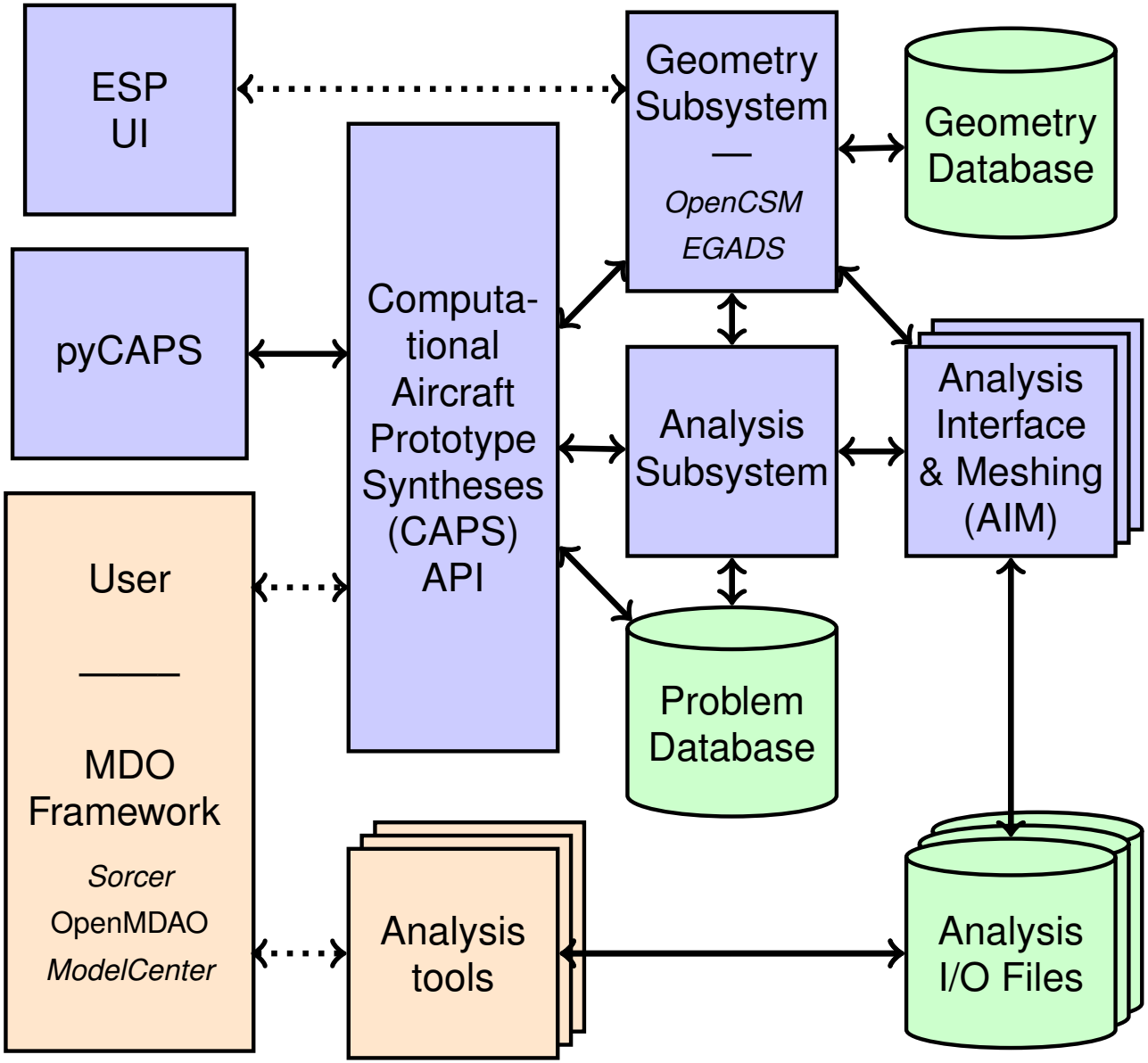
Computational Aircraft Prototype Syntheses: The CAPS API

Enhanced CAPS (EnCAPS) Specification

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

Note: Sections in red are changes in `CAPS` from Revision 1.18.

CAPS Infrastructure in ESP



CAPS Enhancements

Changing Thrusts

CAPS was originally designed to run concurrently with an MDO framework. This has turned out to be rarely the method of execution. In addition there were always issues in restarting from where the runs left off (due to the amount of state info stored in AIMs, the difficulty in getting to the correct place in the control program and the scattering of files). Also if MDO frameworks are not used, then additional execution support is required within the CAPS environment. So the enhancements include:

- Restarting runs the same script (or control program) *recycling* previous data.
- AIM reload. The AIMs ended up maintaining too much internal *state*, which made restarting almost impossible (requiring either rerunning or writing out the state). The AIMs need recasting not to hold on to extraneous data.
- A file structure where the *Problem Database* contains all of the *Analysis I/O Files* (seen in the block diagram on the previous slide).
- Better support for Analysis execution, which embraces asynchronous CAPS running when the Analysis is not run directly in the AIM.
- More emphasis on tracking data and decisions during the session.
- Enhanced handling of sensitivities from both geometry construction and analysis output.
- Removal of Value Object of Value Objects.

CAPS Enhancements

Variable Dimension GeometryIn Value Objects

Now that OpenCSM supports the ability to change the size of its *Design* and *Configuration Parameters* (GeometryIn Value Objects), this complicates dealing with Sensitivities associated with these inputs. This is because the meaning and use of rows and columns are now malleable. In the proposed Enhanced `CAPS` code there are internal *slots* for sensitivities with respect to GeometryOut Value Objects, which are *registered* via calls to `caps_sensitivity`. This is done via specifying which row/column is in play. The same is true for DataSet Objects, which request sensitivity information.

Note that when a changing a GeometryIn Value Object that effects the size of other GeometryIn Value Objects:

1. You can get which other GeometryIn Value Objects are effected when calling `caps_setValue` (see `nGIval` and `GIvals`).
2. Any GeometryOut Value *slots* associated with changed size GeometryIn Objects are invalidated and removed. These would need to get reregistered if still needed.
3. Any DataSets associated with the changed-size GeometryIn Value Objects are also removed and need to be reinstaed if still required.

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 *(sub)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 it's 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 Body Filtering

Filtering the active CSM Bodies occurs at two different stages, once in the CAPS framework, and once in the AIMs. The filtering in the CAPS framework creates sub-groups of Bodies from the CSM stack that are passed to the specified AIM. Each AIM instance is then responsible for selecting the appropriate Bodies from the list it has received.

The filtering is performed by using two Body attributes: “capsAIM” and “capsIntent”.

Filtering within AIM Code

Each AIM can adopt it’s own filtering scheme for down-selecting how to use each Body it receives. The “capsIntent” string is accessible to the AIM, but it is for information only.

CAPS Body Filtering

CSM AIM targeting: "capsAIM"

The CSM script generates Bodies which are designed to be used by specific AIMs. The AIMs that the Body is designed for is communicated to the CAPS framework via the "capsAIM" string attribute. This is a semicolon-separated string with the list of AIM names. Thus, the CSM author can give a clear indication to which AIMs should use the Body. For example, a body designed for a CFD calculation could have:

ATTRIBUTE capsAIM $su2AIM;fun3dAIM;cart3dAIM

CAPS AIM Instantiation: "capsIntent"

The "capsIntent" Body attribute is used to disambiguate which AIM instance should receive a given Body targeted for the AIM. An argument to `caps_load` accepts a semicolon-separated list of keywords when an AIM is instantiated in CAPS/pyCAPS. Bodies from the "capsAIM" selection with a matching string attribute "capsIntent" are passed to the AIM instance. The attribute "capsIntent" is a semicolon-separated list of keywords. If the string to `caps_load` is NULL, all Bodies with a "capsAIM" attribute that matches the AIM name are given to the AIM instance.

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.

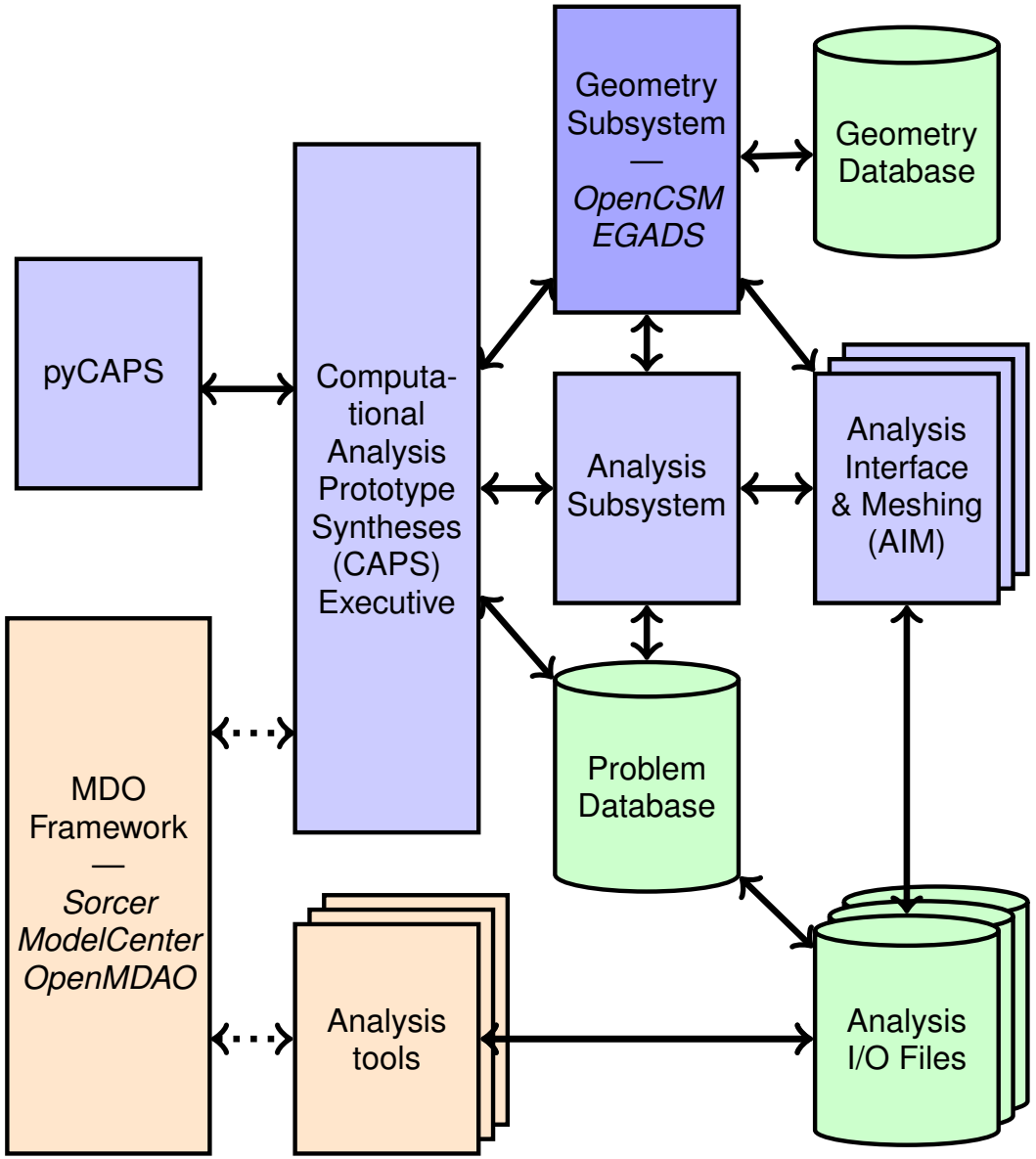
Note: Bound names should not cross dimensional lines.

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.

CAPS Execution



Setup (or read) the Problem:

- Initialize Problem with *csm* (or *static*) file GeomIn and GeomOut parameters
- Specify *mission* parameters
- Make Analysis instances AnalysisIn and AnalysisOut params
- Create *Bounds*, *VetrexSets* & *DataSets*
- Establish linkages between parameters

Run the Problem:

- Adjust the appropriate parameters
- Regenerate Geometry (if *dirty* – lazy)
- Call for Analysis Input file generation
- AIM Execute runs each *solver*
- Inform CAPS that an Analysis has run fills AnalysisOut params & *DataSets* (lazy)
- Generate *Objective Function*

Save the Problem Database when exiting for possible restart

CAPS API – Utilities

Get CAPS revision

```
caps_revision(int *major, int *minor)
```

major	the returned major revision
minor	the returned minor revision number

Open CAPS Problem

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

pname	the full-path ending with the CAPS problem name if exists the stored data initializes the problem, otherwise the directory is created
name	the full-path input file name (not needed when restarting) – based on file extension: *.csm initialize the project using the specified OpenCSM file *.egads initialize the project based on the static geometry
problem	the returned CAPS problem Object
icode	the integer return code

Save Problem file – Obsolete

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

CAPS API – Utilities

Close CAPS Problem

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

problem	the input CAPS problem is written to disk and closed; memory cleanup is performed
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/history 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!

CAPS API – Utilities

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

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

CAPS API – Utilites

Get Problem root

```
icode = caps_getRootPath(capsObj problem, char **fullPath)
```

problem	the input CAPS Problem Object
fullPath	the file path to find the root of the Problem's directory structure if on Windows it will contain the drive
icode	integer return code

Note: All other uses of *path* is relative to this point.

Reset the Object

```
icode = caps_reset(capsObj object, int *nErr, capsErrs **errs)
```

object	the input CAPS Problem/Analysis Object – this is equivalent to a clean slate restart
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 – 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 – I/O

Write Geometry Parameter File

```
icode = caps_writeParameters(capsObj problem, char *fileName)
```

problem	the input CAPS Problem Object
fileName	the name of the parameter file to write
icode	integer return code

Note: This outputs an OpenCSM Design Parameter file.

Read Geometry Parameter File

```
icode = caps_readParameters(capsObj problem, char *fileName)
```

problem	the input CAPS Problem Object
fileName	the name of the parameter file to read
icode	integer return code

Note: This reads an OpenCSM Design Parameter file and overwrites (makes *dirty*) the current state for the GeometryIn Values in the file.

Write out Geometry

```
icode = caps_writeGeometry(capsObj obj, int flag, char *fileName)
```

obj	the input CAPS Problem/Analysis Object
flag	the write flag: 0 – no additional output, 1 – write *.eto* files for EGADS output only for Analysis Objects
fileName	the name of the file to write – typed by extension (case insensitive): iges/igs – IGES File step/stp – STEP File brep – OpenCASCADE File egads – EGADS file (which includes attribution)
icode	integer return code

Note: The *EGADS Tessellation Object* files are named with the `fileName` (minus the extension) and postfixed with the Body index (bias 1) followed by the *eto* extension when there is more than a single Body.

CAPS API – History/Owner

Get History of an Object

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

obj	the input CAPS Object
nhist	the returned length of the history list
hist	the returned pointer to the list of History entities (`nhist` in length)
icode	integer return code

Add History entity to an Object

```
icode = caps_addHistory(capsObj obj, capsOwn hist)
```

obj	the input CAPS Object
hist	a CAPS Owner structure to add to the history for the Object
icode	integer return code

CAPS API – History/Owner

Set History/Owner Data

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

prob	the input CAPS Problem Object
pname	a pointer to the process name character string
nLines	the number of comment lines to describe the history entity
lines	a pointer to a list of character strings with the description
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`

Free Owner Information

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

owner	a pointer to the CAPS Owner structure to free up the internal strings

CAPS API – History/Owner

Get History/Owner Information

```
icode = caps_ownerInfo(capsOwn owner, char **pname, char **pID,
                       char **userID, int *nLines, char ***lines,
                       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
nLines	the returned number of comment lines to describe the history entity
lines	a returned pointer to a list of character strings with the description
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 – Value Object

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 if vtype is 3 and units is "PATH" – slashes are converted automatically
val	the returned CAPS Value Object
icode	integer return code

CAPS API – Value Object

Retrieve Values

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

val	the input Value Object
vtype	the returned data type:

0	Boolean	2	Double	4	*String* Tuple	6	Double *w/ Dot*
1	Integer	3	*Character* String	5	AIM pointer		

nrow	returned number of rows
ncol	returned number of columns – `vlen = nrow * ncol`
data	a filled pointer to the appropriate block of memory (NULL – don't fill)
partial	a returned integer vector/array containing specific `ntype` indications NULL is returned except for `ntype` is 'partial' – filled with 'not NULL' or 'is NULL'
units	the returned pointer to the string declaring the units if vtype is 3 and units "PATH" – slashes are converted automatically
nErr	the returned number of errors generated (Analysis Out) – 0 means no errors
errs	the returned CAPS error structure (Analysis Out) – NULL with no errors
icode	integer return code

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

CAPS API – Value Object

Reset A Value Object

```
icode = caps_setValue(capsObj val, enum vtype, int nrow, int ncol,
                      void *data, int *partial, char *units,
                      int *nErr, capsErrs **errs)
```

val	the input CAPS Value Object (not for GeometryOut or AnalysisOut)
vtype	the 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 used to reset the values
partial	an integer vector/array of length `vlen` containing specific `ntype` indications ignored for `vlen` = 1 or `ntype` is 'NULL invalid' – may be NULL if non-NULL `ntype` is set to 'partial' – must be filled with 'not NULL' or 'is NULL' See `caps_getValueProp`
units	the string declaring the units for `data`
nErr	the returned number of errors generated (Geometry In) – 0 means no errors
errs	the returned CAPS error structure (Geometry In) – NULL with no errors
icode	integer return code

CAPS API – Value Object

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
- **icode** integer return code

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 Properties

```
icode = caps_getValueProps(capsObj val, int *dim, int *flag,
                           enum *lfixed, enum *sfixed, enum *ntype)
```

val	the input Value Object
dim	the returned dimensionality: 0 scalar only 1 vector or scalar 2 scalar, vector or 2D array
flag	the returned flag: 0 – normal, 1 – GeometryIn type → `OCSM_CFGPMTR`
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, 3 – partial NULL
icode	integer return code

Note: this function replaces `caps_getValueShape`

CAPS API – Value Object

Set Value Properties

```
icode = caps_setValueProps(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

Note: this function replaces `caps_setValueShape`

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 `AIM pointer` 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

Free memory in Value Structure

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

value	a pointer to the Value structure to be cleaned up

CAPS API – Value Object

Establish Linkage

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

link	linking Value Object (not for `Tuple` vtype 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 • (Sub)shapes must be compatible • `link` = NULL removes any Linkage
irow	the row to link in the target (or 0 for all)
icol	the column to link in the target (or 0 for all) if both `irow` and `icol` are 0 the entire Object is linked
icode	integer return code

Note: circular linkages are not allowed!

CAPS API – Value Object / Sensitivities

Get a list of the Derivatives available

```
icode = caps_hasDot(capsObj obj, int *ndot, char ***names)
```

obj	the input CAPS Value Object
ndot	the returned length of the number of dots available
names	the returned pointer to the list of derivative names (`ndot` in length – freeable) derivatives derived from vectors/arrays will have "[n]" or "[n,m]" appended
icode	integer return code

Get Derivative values

```
icode = caps_getDot(capsObj obj, char *name, int *len, int *rank,
                    double **dot)
```

obj	the input CAPS Value Object
name	the input name of the derivative
len	the returned length of the data (the length of the Value Object `vlen`)
rank	the returned rank of the derivative
dot	the returned pointer to the derivative information (`len` x `rank` in length)
icode	integer return code

These will only function for GeometryOut/AnalysisOut *Double /w Dot* Value Objects

CAPS API – Sensitivities

Compute Geometric Sensitivities

```
icode = caps_sensitivity(capsObj geomIn, int irow, int icol,
                         int funFlag, int *nErr, capsErrs **errors)
```

geomIn	the input GeometryIn Value Object to compute the sensitivity
irow	the input row to use (for vector/arrays)
icol	the input column to use (for vector/array Value Objects)
funFlag	input *register* flag – 0 register slot, 1 call AIMs, 2 remove slot
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

Side effects:

1. Makes a slot in all GeometryOut Value Objects to store the results with `regFlag` of 0
2. Invokes all AIMs with a defined `aimSensitivity` function when called with `regFlag` of 1
3. This function (`regFlag` is 1), `caps_getDot` or a DataSet request for the sensitivity will fill the appropriate slots in the DataSets and GeometryOuts
4. Any GeometryOut that is a function of the mass properties will have its value slightly changed (to be consistent with the derivative calculation)

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 *exec)
```

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
exec	returned execution flag: 0 – no exec, 1 – AIM Execute exists
icode	integer return code

Get Bodies

```
icode = caps_getBodies(capsObj analysis, int *nBody, ego **bodies)
```

analysis	the Analysis Object
nBody	the returned number of EGADS Body Objects that match the Analysis' `intent`
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

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

Load Analysis into a Problem

```
icode = caps_load(capsObj problem, char *aname, char *unitSys,
                  char *intent, int exec, capsObj *analysis,
                  int *nErr, capsErrs **errs)
```

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)
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
intent	the *intent* character string used to pass Bodies to the AIM, NULL – no filtering
exec	the execution flag: 0 – no exec, 1 – AIM Execute performs analysis, 2 – Auto Exec
analysis	the resultant Analysis Object
nErr	the returned number of errors generated – 0 means no errors
errors	the returned CAPS error structure – NULL with no errors

Notes:

1. The parent/child relationship has been removed and should be replaced with linked AnalysisIn and AnalysisOut Objects to form the dependency
2. The path is gone and is now handled internally
3. If `exec` is 2 and the AIM has `aimExecute`, `aimExecute` automatically runs after `caps_preAnalysis` and if the execution is not asynchronous `aimPostAnalysis` is automatically run. Any errors can be retrieved via a call to `caps_checkAnalysis`.

CAPS API – Analysis

Initialize Analysis from another Analysis Object

```
icode = caps_dupAnalysis(capsObj from, capsObj *analysis)
```

from	an existing CAPS Analysis Object
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 **aname,
                          char **unitSys, char **intent, int *nfields,
                          char ***fnames, int **ranks, int *exec,
                          int *status)
```

analysis	the input Analysis Object
aname	a returned pointer to the string specifying the directory for file I/O
unitSys	returned pointer to string describing the unit system used by the AIM (can be NULL)
intent	the returned pointer to the *intent* character string used to pass Bodies to the AIM
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	returned execution flag: 0 – no exec, 1 – AIM Execute runs analysis, 2 – Auto Exec
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

Execute – required if AIM does execution or *AutoExec*

```
icode = caps_runAnalysis(capsObj analysis, int *status,
                         int *nErr, capsErrs **errors)
```

analysis	the Analysis Object
status	the returned status (0 – done, 1 – running)
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

Has Analysis Completed?

```
icode = caps_checkAnalysis(capsObj analysis, int *phase, int *nErr,
                           capsErrs **errors)
```

analysis	the Analysis Object
phase	the returned phase where errors were generated: 0 – no errors, 1 – during the check, 2 – execution, 3 – post (if automatic)
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_SUCCESS – completed, CAPS_RUNNING – not done)

Mark Analysis as Run

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

analysis	the Analysis Object Note: this clears all Analysis Output Objects to force reloads/recomputes
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 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 VertexSets for cyclic/incremental transfers

```
icode = caps_fillVertexSets(capsObj bound, int *nErr, capsErrs **errs)
```

bound	an input *closed* CAPS Bound 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

Note: Causes the filling of the VertexSets owned by the Bound by forcing the invocation of the appropriate `aimDiscr` functions in the AIM. Under normal circumstances this is deferred to the last postAnalysis call of the collected VertexSets.

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

CAPS API – Analysis Data

Output a VertexSet for Plotting/Debugging

```
icode = caps_outputVertexSet(capsObj vset, char *filename)
```

vset	the VertexSet Object
filename	the VertexSet filename (should have the extension ".vs")
icode	integer return code

The `CAPS` application **vVS** can be used to interactively view the file generated by this function.

This will de deprecated because CAPS viewing will be integrated

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

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

Initialize DataSet for cyclic/incremental startup

```
icode = caps_initDataSet(capsObj dset, int rank, double *startup)
```

dset	the DataSet Object (Method must be Interpolate or Conserve)
rank	the rank of the data (e.g., 1 – scalar, 3 – vector)
startup	the pointer to the constant *startup* data (`rank` in length)

Note: invocations of `caps_getData` and `aim_getDataSet` will return this data (and a length of 1) until properly filled.

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

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

CAPS API – Analysis (AIM) Debug

Backdoor AIM Specific Communication (Retained but should be undocumented)

```
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

Note: Look at the specific AIM documentation to determine if it will respond and to what JSONin commands.

CAPS Return Codes

CAPS_RUNNING	1	CAPS_UNITERR	-320
CAPS_SUCCESS	0	CAPS_NULLBLIND	-321
CAPS_BADRANK	-301	CAPS_SHAPEERR	-322
CAPS_BADDSETNAME	-302	CAPS_LINKERR	-323
CAPS_NOTFOUND	-303	CAPS_MISMATCH	-324
CAPS_BADINDEX	-304	CAPS_NOTPROBLEM	-325
CAPS_NOTCHANGED	-305	CAPS_RANGEERR	-326
CAPS_BADTYPE	-306	CAPS_DIRTY	-327
CAPS_NULLVALUE	-307	CAPS_HIERARCHERR	-328
CAPS_NULLNAME	-308	CAPS_STATEERR	-329
CAPS_NULLOBJ	-309	CAPS_SOURCEERR	-330
CAPS_BADOBJECT	-310	CAPS_EXISTS	-331
CAPS_BADVALUE	-311	CAPS_IOERR	-332
CAPS_PARAMBNDERR	-312	CAPS_DIRERR	-333
CAPS_NOTCONNECT	-313	CAPS_NOTIMPLEMENT	-334
CAPS_NOTPARMTRIC	-314	CAPS_EXECERR	-335
CAPS_READONLYERR	-315	CAPS_CLEAN	-336
CAPS_FIXEDLEN	-316	CAPS_BADINTENT	-337
CAPS_BADNAME	-317	CAPS_NOTNEEDED	-339
CAPS_BADMETHOD	-318	CAPS_NOSENSITVTY	-340
CAPS_CIRCULARLINK	-319	CAPS_NOBODIES	-341

Bounds and the use of Intermediate Results

The Population of the VertexSets

Bounds needed to be fully populated (i.e., the VertexSets need to be filled for all analyses) before they can be used. This is due to the requirement to have all points available to ensure that there is a single UV space (either by construction or by re-parameterization).

By default this is done in the "post" phase of the last analysis in the Bound to be updated, which makes it basically impossible to have an intermediate result for the first iteration (such as in Fluid/Structure Interaction). This issue is mitigated by using the function `caps_fillVertexSets` before the first analysis is invoked. What this does is call the AIM to fill the aimDiscr structure (basically the VertexSet) before the "pre" phase but requires the mesh (or performs the meshing) at that time.

NOTE: An analysis AIM that supports aimDiscr and also generates meshes "on the fly" must be able to generate meshes and call `aim_setTess` from both aimDiscr and aimPreAnalysis (whenever and wherever the mesh gets generated).

Bounds and the use of Intermediate Results

Fluid/Structure Interaction Pseudocode

```
caps_load TetGen aim -> mobj
caps_load fluids aim -> fobj
caps_load structures -> sobj
caps_makeBound "srf" -> bobj
caps_makeVertexSet(bobj, fobj) -> vfobj
caps_makeVertexSet(bobj, sobj) -> vsobj
caps_makeDataSet(vfobj, "Pressure", Analysis, 1) -> dpfobj
caps_makeDataSet(vsobj, "Pressure", Conserve, 1) -> dpsobj
caps_makeDataSet(vsobj, "Displace", Analysis, 3) -> ddsobj
caps_makeDataSet(vfobj, "Displace", Conserve, 3) -> ddfobj
caps_completeBound(bobj)

caps_preAnalysis(mobj)
caps_postAnalysis(mobj)                          /* generate fluids mesh */
caps_fillVertexSets(bobj)                        /* Note #1 */
caps_initDataSet(ddfobj, 3, zeros)               /* Note #2 */

for (iter = 0; iter < nIter; iter++) {
        caps_getData(ddfobj, ...)                /* Note #3 */
        caps_preAnalysis(fobj)
        /* execute fluids analysis */
        caps_postAnalysis(fobj)

        caps_getData(dpsobj, ...)                /* Note #3 */
        caps_preAnalysis(sobj)
        /* execute structures analysis */
        caps_postAnalysis(sobj)
}
```

Bounds and the use of Intermediate Results

Pseudocode Notes

The fluids AIM requires the "Displace" values during its "pre" phase, just as the structural analysis AIM requires "Pressure" (i.e., loads) during its "pre" phase to fill in all the inputs.

1. `caps_fillVertexSets` calls aimDiscr in the fluids AIM, so that AIM must transfer the data from the TetGen AIM to populate the aimDiscr structure. The structures AIM can still do the tessellation in its aimDiscr function, but it will be invoked before any "pre" phase. Care must be taken so that any tessellation input data can be taken from the AIM inputs.
2. `caps_initDataSet` gets called to set the first displacement data to zeros, in that no structural analysis will have been run at start, but is needed by the fluids.
3. `caps_getData` is currently required to actually do the interpolation/conservative data transfer (i.e., it cannot be done in the AIM by the invocation of `aim_getDataSet`). This will be changed in the future, so these calls will not be required, but current scripts and code will still function.
4. The lines in red cause `aimUsesDataSet` to be invoked to determine if the DataSet is required by the Analysis (and will make it *dirty*).