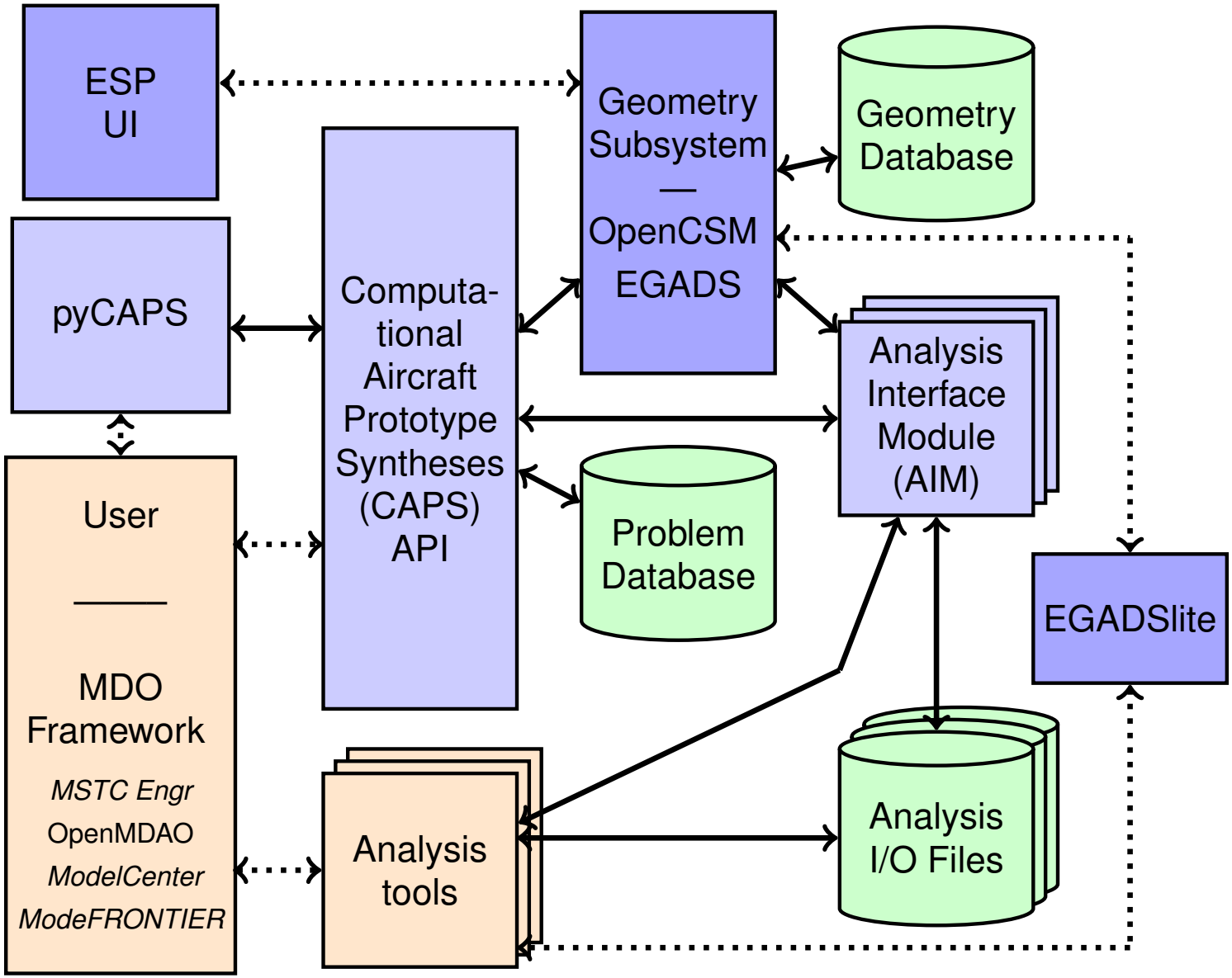

# Computational Aircraft Prototype Syntheses: The CAPS API

## Enhanced CAPS (EnCAPS) Specification for ESP Rev 1.19

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 derivatives 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 derivatives associated with these inputs. This is because the meaning and use of rows and columns are now malleable. There are now internal *slots* for derivatives with respect to GeometryOut Value Objects, which are internally *registered* when `caps_getDot` is called. 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 reinstated 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 |
| capsAnalysis | | capsProblem |
| capsValue | AnalysisIn, AnalysisOut | capsAnalysis |
| capsBound | | capsProblem |
| capsVertexSet | Connected, Unconnected | capsBound |
| capsDataSet | FieldOut, FieldIn, User, GeomSens, TessSens, Builtin | 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_makeAnalysis` 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_makeAnalysis` 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.

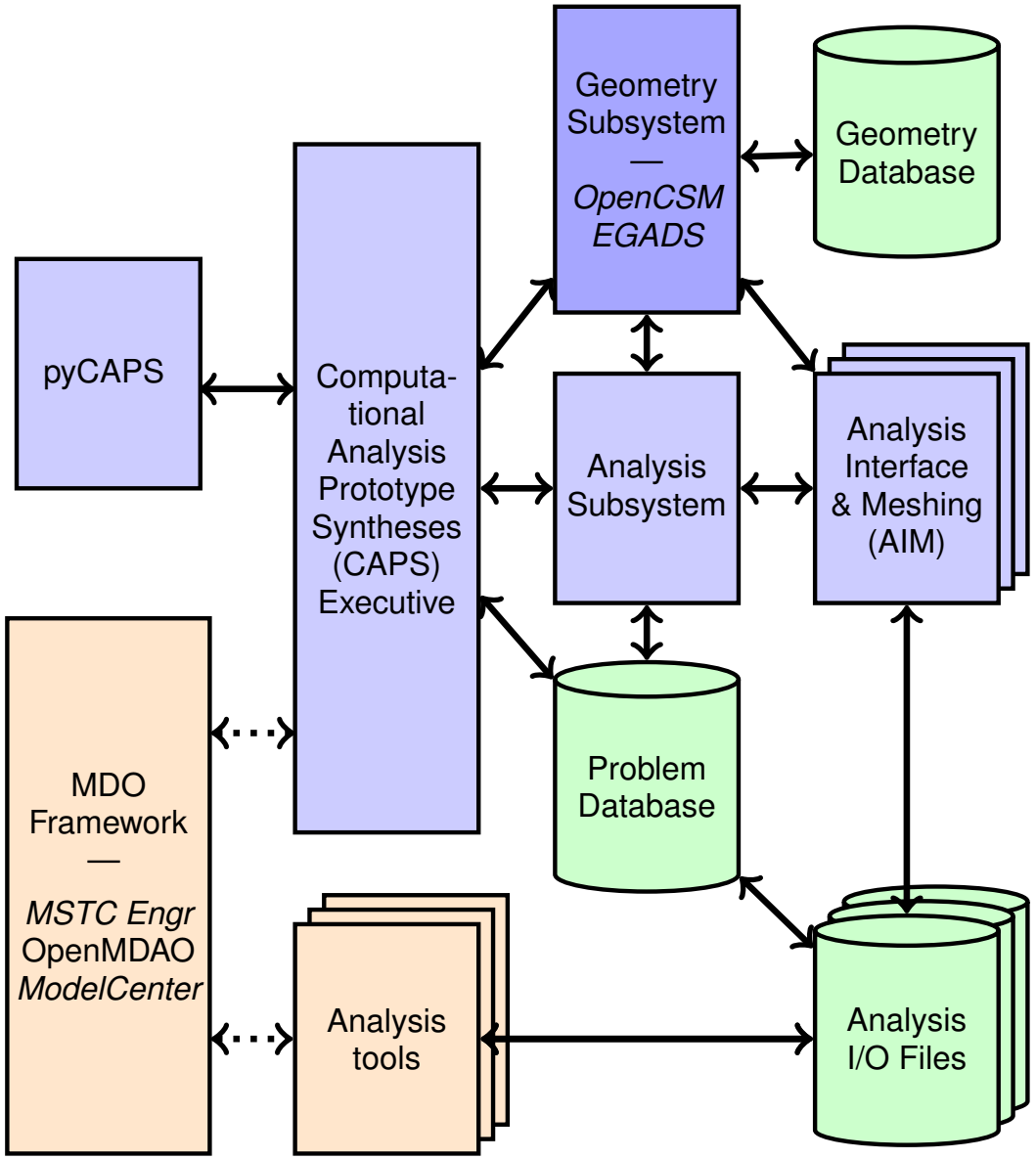
Note: Bound names should not cross dimensional lines.

## 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*

## CAPS Execution Phases

CAPS has 4 modes for starting the session:

- Scratch – This is for development (and not production). It will remove any existing data in the *Scratch* directory of the Problem's path.
- Initial – This *phase* is started by a call to `caps_open` that points to a nonexistent directory. The initialization can either be from a CSM, geometry file, an `OpenCSM` or `EGADS` Model.
- Continuation – This occurs when CAPS has not fully completed a *phase* either do to an interruption or not reaching `caps_close` (where the *phase* is marked as completed). In this case the CAPS application or pyCAPS script can be run from the beginning, but *recycling* of results is used to quickly get to the position where the *phase* terminated.
- Starting from a completed *phase*.

This is controlled by the Problem Object's initialization using `caps_open`.

## CAPS Directory Structure

At the top level prName (of `caps_open`) you will find *phase* subdirectories. Note that *Scratch* is not as protected as the others.

In each *phase* subdirectory you may see:

- capsRestart.cpc – A CSM saved state file – or –
- capsRestart.egads – An EGADS saved geometry file (for nonparametric runs).
- capsRestart – This subdirectory contains the CAPS restart data.
- capsClosed – An indication that the *phase* has been closed (`caps_close` has been called marking completion).
- capsLock – An indication that another application is executing in this subdirectory.
- AIMnames – any number of directories each related to an AIM instance in the running CAPS Problem.

# CAPS API – Information

## Get CAPS revision

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

major the returned major revision

minor the returned minor revision number

## Check State of CAPS Problem Phase

```
icode = caps_phaseState(const char *prName, const char *phName,
                        int *bitFlag)
```

prName the path ending with the CAPS problem name

phName the queried *phase* name (NULL is equivalent to *Scratch*)

bitFlag the returned state (additive): 1 – locked, 2 – closed

icode the integer return code

These functions may be called before CAPS *proper* is started via the invocation of `caps_open`.

# CAPS API – Initiate CAPS

## Open CAPS Problem Phase

```
icode = caps_open(const char *prName, const char *phName, int flag,
                  void *ptr, int outLevel, capsObj *problem,
                  int *nErr, capsErrs **errs)
```

| | |
|---|---|
| prName | the path ending with the CAPS problem name (no spaces)<br>if exists the stored data initializes the problem, otherwise the directory is created |
| phName | the current *phase* name (NULL is equivalent to *Scratch*) |
| flag | 0 – ptr is a filename, 1 – ptr is an `OpenCSM` Model Structure, 2 – ptr is a Model ego,<br>3 – ptr is the starting *phase* name, 4* – continuation (ptr can be NULL) |
| ptr | input path/filename (flag == 0) – based on file extension:<br>*.csm initialize the project using the specified OpenCSM file<br>*.egads initialize the project based on the static geometry<br>– or – pointer to `OpenCSM`/`EGADS` Model – left open after `caps_close` |
| outLevel | 0 - minimal, 1 - standard (default), 2 - debug |
| problem | the returned CAPS problem Object |
| nErr | the returned number of errors generated – 0 means no errors |
| errs | the returned CAPS error structure – NULL with no errors |
| icode | the integer return code |

* Note: A continuation can only occur on the same setup as it was initialized (`ESP` revision, version of OpenCASCADE and machine architecture).

# CAPS API – Utilites

## Do not use CAPS signal handling

```
caps_externSignal()
```

Must be called before `caps_open`. Calling program is responsible for invoking `caps_rmLock()` on any abort, which deletes the capsLock file.

## Get Problem root

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

| | |
|---|---|
| problem | the input CAPS Problem Object |
| fullPath | the file path to find the root of the Problem/Phase directory structure<br>if on Windows it will contain the drive |
| icode | integer return code |

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

# CAPS API – Utilities

## Close CAPS Problem

```
icode = caps_close(capsObj problem, int complete, const char *phName)
```

| | |
|---|---|
| problem | the input CAPS problem is written to disk and closed; memory cleanup is performed |
| complete | 0 – the *phase* is not complete, 1 – the *phase* is completed and should not be modified |
| phName | Phase Name of the Scatch phase is closed as complete |
| icode | the integer return code |

Note: If `caps_open` was initialized with an `OpenCSM` or `EGADS` Model, it is left open.

## Information about an Object

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

| | |
|---|---|
| object | the input CAPS Object |
| name | the returned Object name pointer (if any) |
| otype | the returned Object type: Problem, Value, Analysis, Bound, VertexSet, DataSet |
| stype | the returned subtype (depending on otype) |
| 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

## Delete an Object

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

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

## Number of Children in a Parent Object

```
icode = caps_size(capsObj object, enum capsoType type,
                  enum capssType stype, int *size, int *nErr,
                  capsErrs **errs)
```

| | |
|---|---|
| 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 |
| 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 Child by Index

```
icode = caps_childByIndex(capsObj object, enum capsoType type,
                          enum capssType sty, int ind, capsObj *child)
```

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

## Get Child by Name

```
icode = caps_childByName(capsObj object, enum capsoType type,
                         enum capssType stype, const 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 |

# 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 index, ego *body,
                         char **unit)
```

| | |
|---|---|
| obj | the input CAPS Problem or Analysis Object |
| index | the index [1-`size`] – see `caps_size`, page 19 |
| body | the returned EGADS Body Object |
| units | pointer to the string declaring the length units – NULL for unitless values |
| icode | integer return code |

## Save Problem file – Obsolete

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

# CAPS API – Utilities

## Get Error Information

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

| | |
|---|---|
| errors | the input CAPS Error structure |
| eindex | the index into `error` (1 bias) |
| errObj | the offending CAPS Object |
| eType | the returned error type (CINFO, CWARN, CERROR or CSTAT) |
| 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(const 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(const 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.

# CAPS API – I/O

## Write out Geometry

```
icode = caps_writeGeometry(capsObj obj, int flag, const char *fName,
                           int *nErr, capsErrs **errs)
```

| | |
|---|---|
| obj | the input CAPS Problem/Analysis Object |
| flag | the write flag: 0 – no additional output, 1 – also write Tessellation Objects for EGADS output (only for Analysis Objects) |
| fName | the name of the file to write – typed by extension (case insensitive):<br>iges/igs – IGES File<br>step/stp – STEP File<br>brep – OpenCASCADE File<br>egads – EGADS file (which includes attribution) |
| 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: The *EGADS Tessellation Objects* used by the Analysis Object are written in the EGADS output file along with the geometry of the Bodies.

# 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(const capsObj prob, const 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's Memory

```
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(const capsOwn owner, char **pname, char **pID,
                       char **userID, int *nLines, char ***lines,
                       short *datetime, CAPSLONG *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 – 6 in length:<br>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, const char *vname,
                       enum capssType stype, enum capsvType vtype,
                       int nrow, int ncol, const void *data,
                       int *partial, const char *units, capsObj *val)
```

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

| | | | | | |
|---|---|---|---|---|---|
| 0 | Boolean | 2 | Double | 4 | *String* Tuple |
| 1 | Integer | 3 | String | | |

| | |
|---|---|
| nrow | number of rows |
| ncol | number of columns – `vlen = nrow * ncol` |
| data | pointer to the appropriate block of memory<br>must be a pointer to a contiguous block of memory for strings (each zero terminated)<br>must be a pointer to a *capsTuple* structure(s) when `vtype` is a Tuple |
| partial | integer vector/array containing specific `ntype` indications |
| units | string pointer declaring the units for vtype 2 – NULL for unitless values<br>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 capsvType *vtype, int *nrow,
                      int *ncol, const void **data,
                      const int **partial, const 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/ Deriv* |
| 1 | Integer | 3 | 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)<br>Can use `caps_childByIndex` (page 20) to get Value Objects |
| partial | a returned integer vector/array containing specific `ntype` indications<br>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<br>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 capsvType vtype, int nrow,
                      int ncol, const void *data, const int *partial,
                      const 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 | String | 5 | AIM pointer |

| | |
|---|---|
| nrow | number of rows |
| ncol | number of columns – `vlen = nrow * ncol` |
| data | pointer to the appropriate block of memory used to reset the values<br>must point to a contiguous block of memory for `vlen` strings (each zero terminated) |
| partial | an integer vector/array of length `vlen` containing specific `ntype` indications<br>ignored for `vlen` = 1 or `ntype` is 'NULL invalid' – may be NULL<br>if non-NULL `ntype` is set to 'partial' – must be filled with 'not NULL' or 'is NULL'<br>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, capsvType *vtype,
                       const void **limits, const char **units)
```

| | |
|---|---|
| val | the input Value Object |
| vtype | the data type:<br>1 Integer \| 2 Double |
| limits | an returned pointer to a block of memory containing the valid range [2*sizeof(vtype) in length] – or – NULL if not yet filled |
| units | a string units of the limits |

## Set Valid Value Range

```
icode = caps_setLimits(capsObj val, capsvType vtype, void *limits,
                       const char *units, int *nErr, capsErrs **errs)
```

| | |
|---|---|
| val | the input Value Object (only for the User & Parameter subtypes) |
| vtype | the data type of the limits pointer:<br>1 Integer \| 2 Double |
| limits | a pointer to the appropriate block of memory which contains the minimum and maximum range allowed (2 in length) |
| units | a string units of the limits |
| nErr | the returned number of errors generated – 0 means no errors |
| errs | the returned CAPS error structure – NULL with no errors |

# CAPS API – Value Object

## Get Value Properties (replaces `caps_getValueShape`)

```
icode = caps_getValueProps(capsObj val, int *dim, int *pmtr,
                           enum capsFixed *lfix, enum capsFixed *sfix,
                           enum capsNull *ntype)
```

| | |
|---|---|
| val | the input Value Object |
| dim | the returned dimensionality:<br>0 scalar only<br>1 vector or scalar<br>2 scalar, vector or 2D array |
| pmtr | the returned flag: 0 – normal, 1 – GeometryIn type → `OCSM_CFGPMTR`,<br>2 – GeometryIn type → `OCSM_CONPMTR` |
| lfix | 0 – the length(s) can change, 1 – the length is fixed |
| sfix | 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 |

# CAPS API – Value Object

## Set Value Properties (replaces `caps_setValueShape`)

```
icode = caps_setValueProps(capsObj val, int dim, enum capsFixed lfix,
                           enum capsFixed sfix, enum capsNull ntype,
                           int *nErr, capsErrs **errs)
```

| | |
|---|---|
| val | the input Value Object (only for the User & Parameter subtypes) |
| dim | the dimensionality:<br>0 scalar only<br>1 vector or scalar<br>2 scalar, vector or 2D array |
| lfix | 0 – the length(s) can change, 1 – the length is fixed |
| sfix | 0 – the Shape can change, 1 – Shape is fixed |
| ntype | 0 – NULL invalid, 1 – not NULL, 2 – is NULL |
| nErr | the returned number of errors generated – 0 means no errors |
| errs | the returned CAPS error structure – NULL with no errors |

## Units conversion

```
icode = caps_convertValue(capsObj val, double inVal,
                          const char *inUnit, double *outVal)
```

| | |
|---|---|
| val | a Value Object |
| inVal | the source value to be converted |
| inUnit | the pointer to the string declaring the source units |
| outVal | the returned converted value in the units of the `val` Value Object |

# CAPS API – Value Object

## Transfer Values

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

| | |
|---|---|
| src | the source input Value Object (not for `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<br>Notes:<br>• Must not be GeometryOut or AnalysisOut<br>• Shapes must be compatible<br>• 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 between Value Objects

```
icode = caps_linkValue(capsObj link, enum capstMethod tmethod,
                       capsObj trgt, int *nErr, capsErrs **errs)
```

| | |
|---|---|
| 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`<br>Notes:<br>• Must not be GeometryOut or AnalysisOut<br>• Shapes must be compatible<br>• `link` = NULL – removes 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 |

Note: circular linkages are not allowed!

# CAPS API – Value Object / Derivatives

## Get a list of the Derivatives available

```
icode = caps_hasDeriv(capsObj val, int *ndot, char ***names)
```

| | |
|---|---|
| val | the input CAPS Value Object (*DoubleDeriv* type only) |
| ndot | the returned length of the number of dots available |
| names | the returned pointer to the list of derivative names (`ndot` in length – freeable) |
| icode | integer return code |

## Get Derivative values

```
icode = caps_getDeriv(capsObj val, const char *name, int *len,
                      int *rank, double **dot, int *nErr,
                      capsErrs **errs)
```

| | |
|---|---|
| val | the input CAPS Value Object (*DoubleDeriv* type only) |
| 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) |
| 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 |

*DoubleDeriv* types only exist for GeometryOut and certain AnalysisOut Value

# CAPS API – Units

## Convert value between units

```
icode = caps_convert(int count, const char  *inUnit, double  *inVal,
                                const char *outUnit, double *outVal)
```

| | |
|---|---|
| count | length of `inVal` and `outUnit` arrays |
| inUnit | a string representing the units of `inVal` |
| inVal | the input values to be converted |
| outUnit | a string representing the desired units of `inVal` |
| outVal | the output values in units of `outUnit` (may be same pointer as `inVal`) |
| icode | integer return code |

## Multiply units

```
icode = caps_unitMultiply(const char *unitL, const char *unitR,
                          char **outUnit)
```

| | |
|---|---|
| unitL | a input string representing units |
| unitR | a input string representing units |
| outUnit | a string representing the resulting units from multiplying `unitL` and `unitR` |
| icode | integer return code |

# CAPS API – Units

## Divide units

```
icode = caps_unitDivide(const char *unitL, const char *unitR,
                        char **outUnit)
```

| | |
|---|---|
| unitL | a input string representing units |
| unitR | a input string representing units |
| outUnit | a string representing the resulting units from dividing `unitL` and `unitR` |
| icode | integer return code |

## Raise units

```
icode = caps_unitRaise(const char *unit, int power, char **outUnit)
```

| | |
|---|---|
| unit | a input string representing units |
| power | power to raise `unit` |
| outUnit | a string representing the resulting units from raising `unit` to `power` |
| icode | integer return code |

## Invert units

```
icode = caps_unitInvert(const char *unit, char **outUnit)
```

| | |
|---|---|
| unit | a input string representing units |
| outUnit | a string representing the resulting units from inverting `unit` |
| icode | integer return code |

# CAPS API – Units

## Offset units

```
icode = caps_unitOffset(const char *unit, double off, char **outUnit)
```

| | |
|---|---|
| unit | a input string representing units |
| off | offset to apply to `unit` |
| outUnit | a string representing the resulting units from offsetting `unit` by `off` |
| icode | integer return code |

## Valid unit string

```
icode = caps_unitParse(const char *unit)
```

| | |
|---|---|
| unit | a input string representing units |
| icode | integer return code (`CAPS_SUCCESS` if valid, `CAPS_UNITERR` otherwise) |

# CAPS API – Units

## Valid unit conversion

```
icode = caps_unitConvertable(const char *unitL, const char *unitR)
```

- unitL — a input string representing units
- unitR — a input string representing units
- icode — integer return code (`CAPS_SUCCESS` unitL is convertible to unitR, `CAPS_UNITERR` otherwise)

## Unit comparison

```
icode = caps_unitCompare(const char *unitL, const char *unitR,
                         int *compare)
```

- unitL — a input string representing units
- unitR — a input string representing units
- compare — signed difference between `unitL` and `unitR`
- icode — integer return code

# 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)<br>Attribute name is the Value Object name |
| icode | integer return code |

# CAPS API – Attributes

## Set an Attribute

```
icode = caps_setAttr(capsObj object, const 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
  2D arrays and Tuples are not supported; 1D arrays will have rows only
- **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, const char *aname,
                           int *nIn, int *nOut, int *exec)
```

| | |
|---|---|
| problem | a CAPS Problem Object |
| aname | the Analysis (and AIM plugin) name |
| 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 |

Note: this causes the the DLL/Shared-Object to be loaded (if not already resident)

## Get Bodies

```
icode = caps_getBodies(capsObj aobj, int *nBody, ego **bodies,
                       int *nErr, capsErrs **errs)
```

| | |
|---|---|
| aobj | 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 (length – nBody) |
| 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

## Query Analysis Input Information

```
icode = caps_getInput(capsObj problem, const 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, const 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

## Create a new Analysis Object (replaces `caps_load`)

```
icode = caps_makeAnalysis(capsObj problem, const char *aname,
                          const char *name, const char *uSys,
                          char *intent, int exec, capsObj *analysis,
                          int *nErr, capsErrs **errs)
```

| | |
|---|---|
| problem | a CAPS Problem Object |
| aname | the Analysis (AIM plugin) name |
| name | the unique supplied name for this instance (can be NULL) |
| uSys | 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. This causes the DLL/Shared-Object aname to be loaded (if not already resident)
2. The parent/child relationship has been removed and should be replaced with linked AnalysisIn and AnalysisOut Objects to form the dependency
3. The path is gone and is now handled internally
4. 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, const char *name, capsObj *obj)
```

| | |
|---|---|
| from | an existing CAPS Analysis Object |
| name | the name of the duplicate Analysis Object |
| obj | the resultant Analysis Object |

## Reset the Analysis

```
icode = caps_resetAnalysis(capsObj aobj, int *nErr, capsErrs **errs)
```

| | |
|---|---|
| aobj | the input Analysis Object – this clears out the analysis directory |
| nErr | the returned number of errors generated – 0 means no errors |
| errs | the returned CAPS error structure – NULL with no errors |

## Get Dirty Analysis Object(s)

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

| | |
|---|---|
| obj | 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*) |

Note: Listed from most *stale* to most recent – the order in which to execute.

# CAPS API – Analysis

## Get Info about an Analysis Object

```
icode = caps_analysisInfo(capsObj aobj, char **dir, char **uSys,
                          int *major, int *minor, char **intent,
                          int *nfields, char ***fnames, int **frank,
                          int **fInOut, int *exec, int *status)
```

| | |
|---|---|
| aobj | the input Analysis Object |
| dir | a returned pointer to the string specifying the directory for file I/O<br>name (or aname augmented with the instance number) of `caps_makeAnalysis` |
| uSys | returned pointer to string describing the unit system used by the AIM (can be NULL) |
| major | the returned AIM major version number |
| minor | the returned AIM minor version number |
| 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 |
| frank | a returned pointer to a list of ranks associated with each field |
| fInOut | a returned pointer to a list of field flags (FIELDIN - input, FIELDOUT - output) |
| 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<br>3 – both Geometry & Analysis inputs are *dirty* , 4 – new geometry,<br>5 – *post Analysis* required, 6 – Execution & *post Analysis* required |
| icode | integer return code |

# CAPS API – Analysis

## Generate Analysis Inputs

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

| | |
|---|---|
| analysis | the Analysis (or Problem) Object<br>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(const 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 |
| 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 |

Note: this clears all Analysis Output Objects to force reloads/recomputes

# CAPS API – Analysis Data

## Create a Bound

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

| | |
|---|---|
| problem | the CAPS Problem Object |
| dim | the dimensionality of the Bound (1 – 3) |
| bname | the character string associated with "capsBound" attribute on bodies |
| bound | the returned new CAPS Bound Object |
| icode | integer return code |

## Get Information about a Bound

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

| | |
|---|---|
| bound | the CAPS Bound Object |
| state | the returned Bound state:<br>-1 Open<br>0 Empty & Closed<br>1 single BRep entity<br>2 multiple BRep entities<br>-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) |

# CAPS API – Analysis Data

## Make a VertexSet

```
icode = caps_makeVertexSet(capsObj bound, capsObj analysis,
                           const char *vname, capsObj *vset,
                           int *nErr, capsErrs **errs)
```

| | |
|---|---|
| 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 |
| 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 |

## 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 |

## Close a Bound

```
icode = caps_closeBound(capsObj bound)
```

| | |
|---|---|
| bound | an input *open* CAPS Bound Object to close |
| icode | integer return code |

## Output a VertexSet for Plotting/Debugging

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

| | |
|---|---|
| vset | the VertexSet Object |
| filename | the VertexSet filename (should have the extension ".vs") |

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

This will be 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 *FieldOut* 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, const char *dname,
                         enum capsfType ftype, int rank,
                         capsObj *dset, int *nErr, capsErrs **errs)
```

| | |
|---|---|
| vset | the VertexSet Object – associated Bound must be *open* |
| dname | a pointer to a string containing the name of the DataSet (i.e., *pressure*) |
| ftype | the type of data field: (FieldIn, FieldOut, GeomSens, TessSens, User) |
| rank | the rank of the data for a User field (e.g., 1 – scalar, 3 – vector), ignored otherwise |
| dset | the returned DataSet 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 |

## Get DataSet Information

```
icode = caps_dataSetInfo(capsObj dset, enum capsfType *ftype,
                         capsObj *link, enum capsdMethod *dmethod)
```

| | |
|---|---|
| dset | the input DataSet Object |
| ftype | the returned type of data field: (FieldIn, FieldOut, BuiltIn, GeomSens, TessSens, User) |
| link | the returned linked DataSet Object (for ftype of FieldIn) – can be NULL |
| dmethod | the returned linked DataSet Method (only valid for ftype of FieldIn) |
| icode | integer return code |

# CAPS API – Analysis Data

## Get Data from a DataSet

```
icode = caps_getData(capsObj dset, int *npt, int *rank, double **data,
                     char **units, int *nErr, capsErrs **errs)
```

| | |
|---|---|
| dset | the DataSet Object |
| npt | 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 |
| 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 – Analysis Data

## Establish Linkage between DataSet Objects

```
icode = caps_linkDataSet(capsObj link, enum capsdMethod dmethod,
                         capsObj trgt, int *nErr, capsErrs **errs)
```

| | |
|---|---|
| link | linking DataSet Object, must be FieldOut |
| dmethod | 0 – Interpolate, 1 – Conserve |
| trgt | the target DataSet Object which will get its data from `link`, must be FieldIn or User |
| nErr | the returned number of errors generated – 0 means no errors |
| errs | the returned CAPS error structure – NULL with no errors |

## Initialize DataSet for cyclic/incremental startup

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

| | |
|---|---|
| dset | the DataSet Object (Field type must be FieldIn) |
| rank | the rank of the data (e.g., 1 – scalar, 3 – vector) |
| startup | the pointer to the constant *startup* data (`rank` in length) |
| nErr | the returned number of errors generated – 0 means no errors |
| errs | the returned CAPS error structure – NULL with no errors |

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 DataSet Objects by Name

```
icode = caps_getDataSets(capsObj bound, const 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 |

## Put *User* Data into a DataSet

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

| | |
|---|---|
| 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 |
| 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 – 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 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). As a result, the meshing information for an AIM maybe required prior to calling the `aimPreAnalysis`.

The VertexSets are filled with calls the AIM to fill the `aimDiscr` structure (basically the VertexSet), which means the meshing information must be available via a link or generated in aimDiscr.

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

# Bounds and the use of Intermediate Results

## Fluid/Structure Interaction Pseudocode

```
caps_load egadsTess aim -> msobj
caps_load TetGen aim -> mfobj
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", FieldOut) -> dpfobj
caps_makeDataSet(vsobj, "Pressure", FieldIn ) -> dpsobj
caps_makeDataSet(vsobj, "Displace", FieldOut) -> ddsobj
caps_makeDataSet(vfobj, "Displace", FieldIn ) -> ddfobj
caps_linkDataSet(dpfobj, Conserve, dpsobj)
caps_linkDataSet(ddsobj, Conserve, ddfobj)
caps_initDataSet(ddfobj, 3, zeros)            /* Note #1 */
caps_closeBound(bobj)

caps_preAnalysis(msobj)
caps_postAnalysis(msobj)                      /* generate structures mesh */
caps_preAnalysis(mfobj)
caps_postAnalysis(mfobj)                      /* generate fluids mesh */

for (iter = 0; iter < nIter; iter++) {
        caps_preAnalysis(fobj)                /* Note #2 */
        /* execute fluids analysis */
        caps_postAnalysis(fobj)

        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_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.
2. The lines in red and will mark Analysis *dirty* when the DataSet is filled.
3. `caps_fillVertexSets` has been removed.
4. `caps_getData` is no longer required . Current scripts will still function. Calls to `caps_preAnalysis` will trigger the transfer to the linked FieldIn DataSet associated with that AIM.

# API index