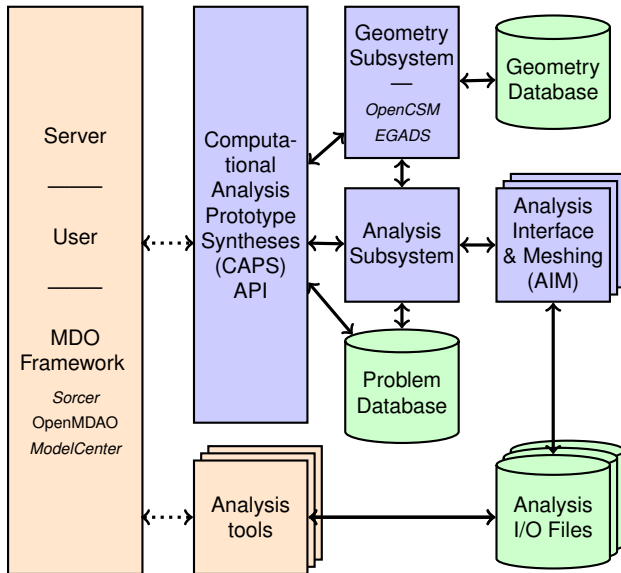


# Computational Aircraft Prototype Syntheses: The CAPS API

## A Preliminary Design Specification

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

# CAPS Infrastructure



# 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. Various data entries can be connected via *linkage ports* found in all *input* objects. There can be multiple Problems in a single execution of CAPS and each Problem is designed to be *thread safe* allowing for multi-threading of CAPS at the highest level.

## Value Object

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

## 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
- 3D – At the Body level

## VertexSet Object

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

## DataSet Object

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

# CAPS Objects

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

Body Objects are EGADS Objects (egos)

# CAPS Geometric Fidelity

This value reflects the Fidelity that will be generated for a specific analysis. These values must be agreed upon upfront. A Geometric Model may have an input *parameter* called “capsFidelity” (and will have the Attribute “capsFidelity” with the associated value on the resultant Body). When executing a multidisciplinary model there may be the requirement for the generation of multiple bodies each with the appropriate *fidelity*. Suggested correspondence:

Fidelity	Description	Fidelity	Description
1	Beam	11	MidSurface Aero
2	ASWING	12	MidSurface w/ Control Surfaces
3	LSM	13	AVL
4	Built-up Element Model	14	OML
5	Solid Structural Model	15	Detailed OML

# 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.

## capsBoundary

This string Attribute must be applied to EGADS BRep objects to indicate which CAPS Bound(s) are associated with the geometry. An Face/Body 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.



# CAPS API – Utilities

## Open CAPS Problem

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

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

**\*.caps** read the saved CAPS problem file

**\*.csm** initialize the project using the specified OpenCSM file

**\*.egads** initialize the project based on the static geometry

**pname** the input CAPS problem process name

**problem** the returned CAPS problem object

**icode** the integer return code

## Close CAPS Problem

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

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

**icode** the integer return code

# CAPS API – Utilities

## Save Problem file

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

**problem** the input CAPS problem object to write

**name** the save file name – no extension (added by this function)

**icode** the integer return code

## Information about an Object

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

**object** the input CAPS object

**name** the returned object name pointer (if any)

**type** the returned data type: Problem, Value, Analysis, Bound, VertexSet, DataSet

**stype** the returned subtype (depending on type)

**link** the returned linkage Value object (**NULL** – no link)

**parent** the returned parent object (**NULL** for a Problem or an Attribute generated User Value)

**last** the returned last owner to *touch* the object

**icode** integer return code

# CAPS API – Utilities

## Children Sizing info from a Parent Object

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

**object** the input CAPS object

**type** the data type to size: Bodies, Attributes, Value, Analysis, Bound, VertexSet, DataSet

**stype** the subtype to size (depending on type)

**size** the returned size

**icode** integer return code

## Get Child by Index

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

**object** the input parent object

**type** the object type to return: Value, Analysis, Bound, VertexSet, DataSet

**stype** the subtype to size (depending on type)

**index** the index [1-size]

**child** the returned CAPS object

**icode** integer return code

## 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 size (depending on type)
- name** a pointer to the index character string
- child** the returned CAPS object
- icode** integer return code

## Delete an Object

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

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

# CAPS API – Utilities

## Get Body by index

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

**prob** the input CAPS Problem 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

## Get Owning Information

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

**owner** the input CAPS Owning structure

**pname** the returned pointer to the process name

**pID** the returned pointer to the process ID

**userID** the returned pointer to the user ID

**datetime** the filled date/time stamp info [year, month, day, hour, minute, second]

**sNum** the sequence number (always increasing)

**icode** integer return code

# CAPS API – Utilities

## Get Error Information

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

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

## Free Error Structure

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

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

# CAPS API – Value Objects

## Create A Value Object

```
icode = caps_makeValue(capsObj problem, char *vname, enum subtype,  
                      enum vtype, int vlen, 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
1	Integer	3	Character String

**vlen** the Value length (not needed for strings)

**data** pointer to the appropriate block of memory

**units** pointer to the string declaring the units – **NULL** for unitless values

**val** the returned CAPS Value object

**icode** integer return code

# CAPS API – Value Objects

## Retrieve Values

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

**val** the input Value object

**vtype** the returned data type:

<b>0</b>	Boolean	<b>2</b>	Double	<b>4</b>	Value Object
<b>1</b>	Integer	<b>3</b>	Character String		

**vlen** the returned value length

**data** a filled pointer to the appropriate block of memory (**NULL** – don't fill)  
Can use `childByIndex` to get Value Objects

**units** the returned pointer to the string declaring the units

**nErr** the returned number of errors generated – **0** means no errors

**errs** the returned CAPS error structure – **NULL** with no errors

**icode** integer return code



# CAPS API – Value Objects

## Reset A Value Object

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

**val** the input CAPS Value object (not for GeometryOut or AnalysisOut)

**vlen** the Value length (not needed for strings)

**data** pointer to the appropriate block of memory used to reset the values

## Get Valid Value Range

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

**val** the input Value object

**limits** an returned pointer to a block of memory containing the valid range [ $2 * \text{sizeof}(\text{vtype})$  in length] – or – **NULL** if not yet filled

## Set Valid Value Range

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

**val** the input Value object (only for the User & Parameter subtypes)

**limits** a pointer to the appropriate block of memory which contains the minimum and maximum range allowed (2 in length)

**icode** integer return code

# CAPS API – Value Object

## Get Value Shape/Dimension

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

**val** the input Value object

**dim** the returned dimensionality:

**0** scalar only

**1** vector or scalar

**2** scalar, vector or 2D array

**lfixed** **0** – the length(s) can change, **1** – the length is fixed

**sfixed** **0** – the Shape can change, **1** – Shape is fixed

**ntype** **0** – NULL invalid, **1** – not NULL, **2** – is NULL

**nrow** number of rows – parent index for Value vtypes

**ncol** number of columns

Note:  $vlen = nrow * ncol$

**icode** integer return code

# CAPS API – Value Object

## Set Value Shape/Dimension

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

[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

[nrow](#) number of rows

[ncol](#) number of columns

Note: creating a [capsObj](#) with this data, first invoke `caps_makeValue` with the correct `vlen` then call this function

[icode](#) integer return code

# CAPS API – Value Object

## Transfer Values

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

**src** the source input Value object (not for Value vtype) – or –  
DataSet object

**tmethod** **0** – copy, **1** – integrate, **2** – weighted average – (**1** & **2** only for DataSet src)

**dst** the destination Value object to receive the data  
Notes:

- Must not be GeometryOut or AnalysisOut
- Shapes must be compatible
- Overwrites any Linkage

**nErr** the returned number of errors generated – **0** means no errors

**errs** the returned CAPS error structure – **NULL** with no errors

**icode** integer return code

# CAPS API – Value Object

## Establish Linkage

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

**link** linking Value object (not for Value 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
- Shapes must be compatible
- link = NULL removes any Linkage

**icode** integer return code

Note: circular linkages are not allowed!

## 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)

**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

## 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

**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

## Load Analysis into a Problem

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

**problem** a CAPS Problem object

**aname** the Analysis (and AIM plugin) name

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

**apath** the absolute filesystem path to both read and write files

this is required even if the AIM does not use the the filesystem, so that the combination of aname and apath is unique

**fidelity** the *Geometric Fidelity* value used

ignored if the Analysis only supports a single fidelity

**naobj** the number of parent analysis object(s)

**aobjs** a list of the parent analysis object(s) – may be NULL if `naobj == 0`

**analysis** the resultant Analysis object

**icode** integer return code



## Get Info about an Analysis Object

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

**analysis** the input Analysis object

**apath** a returned pointer to the string specifying the filesystem path for file I/O

**fid** the returned *Geometric Fidelity* associated with this Analysis object

**naobj** the returned number of *parent* analysis object(s)

**aobjs** a returned pointer to a list of the *parent* analysis object(s)

**nfields** the returned number of fields for DataSet filling

**fnames** a returned pointer to a list of character strings with the field/DataSet names

**ranks** a returned pointer to a list of ranks associated with each field

**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

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

## Mark Analysis as Run

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

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

## Get Dirty Analysis Object(s)

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

**problem** a CAPS Problem 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 Data

## Create a Bound – Open until completeBound

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

**problem** a CAPS Problem object

**dim** the dimensionality of the Bound (1 – 3)

**bname** the Bound name (matching the *capsBoundary* Attribute)

**bound** the resultant *open* Bound object

**icode** integer return code

## Complete a Bound

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

**bound** the CAPS Bound object to close after creating all of the VertexSets & DataSets  
make calls to *makeVertexSet* and *makeDataSet* in between these 2 functions

**icode** integer return code

## Get Information about a Bound

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

**bound** the CAPS Bound object

**state** the returned Bound state:

- 1 Open
- 0 Empty & Closed
- 1 single BRep entity
- 2 multiple BRep entities
- 2 multiple BRep entities – Error in reparameterization!

**dim** the returned dimensionality of the Bound (1 – 3)

**plims** the filled parameterization limits (2 values when dim is 1, 4 when dim is 2)

**icode** integer return code

# CAPS API – Analysis Data

## Make a VertexSet

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

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

## Get Info about a VertexSet

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

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

# CAPS API – Analysis Data

## Fill an Unconnected VertexSet

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

**vset** the input Unconnected VertexSet object

**npts** the number of points in the VertexSet

**xyzs** the point positions (3\*npts in length)

**icode** integer return code

## Create a DataSet

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

**vset** the VertexSet object – associated Bound must be *open*

**dname** a pointer to a string containing the name of the DataSet (i.e., *pressure*)

**method** the method used for data transfers: (Sensitivity, Analysis, Interpolate, Conserve, User)

**rank** the rank of the data (e.g., 1 – scalar, 3 – vector)

**dset** the returned DataSet object

**icode** integer return code

## 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	<i>Geometry</i> Positions	
xyzd	3	<i>Data</i> Positions	Not for vertex-based discretizations
param*	1/2	t or [u,v] data for <i>Geometry</i> Positions	
paramd*	1/2	t or [u,v] for <i>Data</i> Positions	Not for vertex-based discretizations
<i>GeomIn</i> *	3	Sensitivity for the <i>Geometry</i> Input <i>GeomIn</i>	can have [ <i>irow</i> , <i>icol</i> ] in name

\* Note: not valid for 3D Bounds



# CAPS API – Analysis Data

## Get Data from a DataSet

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

**dset** the DataSet object

**npts** the returned number of points in the DataSet

**rank** the returned rank of the data (e.g., 1 – scalar, 3 – vector)

**data** the returned pointer to the data (rank\*npts in length)

**units** the returned pointer to the string declaring the units

**icode** integer return code

## Get History of a DataSet

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

**dset** the DataSet object

**vset** the returned associated VertexSet object

**nhist** the returned length of the history list

**hist** the returned pointer to the list (nhist in length)

**icode** integer return code

# CAPS API – Analysis Data

## Put *User* Data into a DataSet

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

**dset** the DataSet object

**nverts** the number of points in data – must match declared `npts`

**rank** the rank of the data – must match declared `rank` (e.g., 1 – scalar, 3 – vector)

**data** a pointer to the data (`rank*nverts` in length)

**units** the pointer to the string declaring the units

**icode** integer return code

## Get DataSet Objects by Name

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

**bound** an input CAPS Bound object

**dname** a pointer to a string containing the name of the DataSet

**nobj** the returned number of objects with the name

**dsets** a returned pointer to the list of DataSet objects (*freeable*)

**icode** integer return code

## Get Triangulations for a 2D VertexSet

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

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

# Analysis Interface & Meshing

- Hides all of the individual Analysis details (and peculiarities)
- Dynamically loaded at runtime – extendibility and extensibility
  - Windows Dynamically Loaded Libraries (name.dll)
  - LINUX Shared Objects (name.so)
  - MAC Bundles, CAPS will use the so file extension
- An AIM plugin is required for each Analysis code at:
  - a specific *fidelity*
  - a specific *mode* (i.e., where the inputs may be different)
- Plugin names must be unique – loaded by the name
- † indicates memory handled by CAPS in the following functions  
i.e., CAPS will free these memory blocks when necessary

# Analysis Interface & Meshing – Initialization

## Initialization Information for the AIM

```
icode = aimInitialize(int ngIn, capsValue *gIn, int **nFid,  
                    int **fids, int *nIn, int *nOut,  
                    int *nFields, char ***fnames, int **ranks)
```

**ngIn** the number of *Geometry* Input value structures

**gIn** a pointer to the list of *Geometry* Input value structures

**nFid** the returned number of fidelities this AIM requires

**fids** the returned pointer to the geometric fidelities associated with this Analysis †

**nIn** the number of Inputs

**nOut** the number of possible Outputs

**nFields** the number of fields to responds to for DataSet filling

**fnames** a pointer to a list of character strings with the field/DataSet names †

**ranks** a pointer to a list of ranks associated with each field †

**icode** integer return code or AIM *instance* counter

## capsValue Structure

The **capsValue** Structure is simply the data found within a CAPS Value object.

# Analysis Interface & Meshing – Support

## Discrete Structure

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

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

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

**bname** the Boundary name

Note: all of the BRep entities are examined for the attribute **capsBoundary**. Any that match bname must be included when filling this **capsDiscr**.

**discr** the Discrete structure to fill

Note: the AIM *instance*, AIM *info* pointer and the dimensionality have been filled in before this function is invoked.

**icode** integer return code

# Analysis Interface & Meshing – Support

## Frees up data in a Discrete Structure

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

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

## Element in the *Mesh* – Optional

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

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

# Analysis Interface & Meshing – Input Prep

## Input Information for the AIM

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

- inst** the AIM *instance* index
- aimInfo** the AIM context (used by the Utility Functions)
- index** the Input index [1-nIn]
- ainame** a pointer to the returned Analysis Input variable name
- default** a pointer to the filled default value(s) and units – CAPS will free any allocated memory

## Parse Input data & Generate Input File(s)

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

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



# Analysis Interface & Meshing – Output Parsing

## Output Information for the AIM

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

**inst** the AIM *instance* index

**index** the Output index [1-nOut]

**aonam** a pointer to the returned Analysis Output variable name

**form** a pointer to the Value Shape & Units information – to be filled  
any actual values stored are ignored

## Calculate/Retrieve Output Information

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

**inst** the AIM *instance* index

**aimInfo** the AIM context (used by the Utility Functions)

**apath** the filesystem path where the Analysis output file(s) should be read

**index** the Output index [1-nOut] for this single result

**val** a pointer to the capsValue data to fill – CAPS will free any allocated memory

**errors** a pointer to the returned error structure where output parsing error(s) occurred  
**NULL** with no errors

## Data Transfer using the Discrete Structure – Optional

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

**discr** the input Discrete Structure

**name** the field name to that corresponds to the fill

**npts** the number of points to be filled

**rank** the rank of the data

**data** a pointer associated with the data to be filled ( $\text{rank} \times \text{npts}$  in length)

**units** the returned pointer to the string declaring the units †  
return **NULL** to indicate unitless values

**icode** integer return code

# Analysis Interface & Meshing – Data Transfers

## Interpolation on the Bound – Optional

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

- discr** the input Discrete Structure
- name** a pointer to the input DataSet name string
- eIndex** the input target element index (1 bias) in the Discrete Structure
- bary** the input Barycentric/reference position in the element eIndex
- rank** the input rank of the data
- data** values at the data (or geometry) positions
- result** the filled in results (rank in length)
- r\_bar** input  $d(\text{objective})/d(\text{result})$
- d\_bar** returned  $d(\text{objective})/d(\text{data})$
- icode** integer return code

Forward and *reverse differentiated* functions

# Analysis Interface & Meshing – Data Transfers

## Element Integration on the Bound – Optional

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

**discr** the input Discrete Structure

**name** a pointer to the input DataSet name string

**eIndex** the input target element index (1 bias) in **discr**

**rank** the input rank of the data

**data** values at the data (or geometry) positions – **NULL** length/area/volume of element

**result** the filled in results (**rank** in length)

**r\_bar** input  $d(\text{objective})/d(\text{result})$

**d\_bar** returned  $d(\text{objective})/d(\text{data})$

**icode** integer return code

Forward and *reverse differentiated* functions

# Analysis Interface & Meshing – Data Transfers

## Data Transfer to Child AIM – Optional

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

**name** the agreed-upon data name to transfer

**vtype** value data type – returned

**rank** the rank of the data – returned (negative – child should free data)

**nrow** the number of rows – returned

**ncol** the number of columns – returned

**data** a void pointer associated with the data – returned

**units** the pointer to the string declaring the units (will be free'd by child) – returned  
return **NULL** to indicate unitless values

**icode** integer return code

# Analysis Interface & Meshing – Utility Library

## Get Bodies

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

**aimInfo** the AIM context

**nBody** the returned number of EGADS Body objects that match the *Fidelity*

**bodies** the returned pointer to a list of EGADS Body objects

**icode** integer return code

## Units conversion

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

**aimInfo** the AIM context

**inUnits** the pointer to the string declaring the source units

**inValue** the value to be converted

**outUnits** the pointer to the string declaring the desired units

**outValue** the returned converted value

**icode** integer return code

## Name to Index conversion

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

**aimInfo** the AIM context

**name** the pointer to the string specifying the name to look-up

**stype** Value subtype (GEOMETRYIN, ANALYSISIN or ANALYSISOUT)

**icode** index (1 bias) or negative integer return code

## Index to Name conversion

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

**aimInfo** the AIM context

**index** the index to use (1 bias)

**stype** Value subtype (GEOMETRYIN, ANALYSISIN or ANALYSISOUT)

**name** the returned pointer to the string specifying the name

**icode** integer return code

## Data Transfer from Parent AIM(s)

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

**aimInfo** the AIM context

**name** the requested agreed-upon name to fill

**vtype** the returned value data type

**rank** the returned rank of the data (negative – data should be free'd when done)

**nrow** the returned number of rows

**ncol** the returned number of columns

**data** a returned void pointer associated with the data

**units** the returned pointer to the string declaring the units (should be free'd)

**NULL** indicates unitless values

**icode** integer return code

Notes: All parent AIMS are queried. If none properly respond, this function returns CAPS\_NOTFOUND. If multiple parents respond then this function returns CAPS\_SOURCEERR. Parents must not be *dirty*.



## Establish Linkage from Parent or Geometry

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

**aimInfo** the AIM context

**name** the requested Value object name to link

**stype** Value subtype (GEOMETRYIN, GEOMETRYOUT, ANALYSISIN or ANALYSISOUT)

**default** the pointer from aimInputs

**icode** integer return code

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

# CAPS Return Codes

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