

pyCAPS: A Python Extension Module for the Computational Aircraft Prototype Syntheses (CAPS)

Ryan Durscher
AFRL/RQVC

July 30, 2019

Contents

1	Introduction	1
1.1	Overview	1
1.2	Key differences between pyCAPS and CAPS	1
1.3	Clearance Statement	1
2	Getting Started	1
2.1	The capsProblem Class	2
2.1.1	Initialization and termination	2
2.1.2	Loading the geometry	2
2.1.3	Loading an AIM	3
2.2	Working with Geometry	3
2.2.1	Setting and getting design parameters	3
2.2.2	Viewing geometry	3
2.3	Working with an AIM	4
2.3.1	Setting and getting AIM inputs and outputs	4
2.3.2	AIM pre- and post- analysis	4
3	Namespace Index	5
3.1	Packages	5
4	Class Index	5
4.1	Class List	5
5	Namespace Documentation	5
5.1	pyCAPS Namespace Reference	5
5.1.1	Detailed Description	6
5.1.2	Function Documentation	6
6	Class Documentation	7
6.1	_capsAnalysis Class Reference	7
6.1.1	Detailed Description	8
6.1.2	Member Function Documentation	9
6.1.3	Member Data Documentation	17
6.2	_capsBound Class Reference	18
6.2.1	Detailed Description	19
6.2.2	Member Function Documentation	19
6.2.3	Member Data Documentation	21
6.3	_capsDataSet Class Reference	22
6.3.1	Detailed Description	23
6.3.2	Member Function Documentation	23

6.3.3	Member Data Documentation	24
6.4	_capsGeometry Class Reference	25
6.4.1	Detailed Description	26
6.4.2	Member Function Documentation	26
6.4.3	Member Data Documentation	31
6.5	_capsValue Class Reference	31
6.5.1	Detailed Description	32
6.5.2	Member Function Documentation	32
6.5.3	Member Data Documentation	33
6.6	_capsVertexSet Class Reference	34
6.6.1	Detailed Description	34
6.6.2	Member Data Documentation	34
6.7	CAPSError Class Reference	35
6.7.1	Detailed Description	35
6.7.2	Constructor & Destructor Documentation	35
6.7.3	Member Data Documentation	36
6.8	capsProblem Class Reference	38
6.8.1	Detailed Description	39
6.8.2	Constructor & Destructor Documentation	39
6.8.3	Member Function Documentation	40
6.8.4	Member Data Documentation	45
6.9	capsViewer Class Reference	46
6.9.1	Detailed Description	47
6.9.2	Constructor & Destructor Documentation	47
6.9.3	Member Function Documentation	48
6.9.4	Member Data Documentation	55
7	Example Documentation	55
7.1	analysis2.py	55
7.2	analysis3.py	56
7.3	analysis4.py	56
7.4	analysis5.py	57
7.5	analysis6.py	58
7.6	analysis7.py	58
7.7	exception.py	59
7.8	problem.py	59
7.9	problem1.py	59
7.10	problem2.py	60
7.11	problem3.py	60
7.12	problem4.py	61

7.13	problem5.py	61
7.14	problem6.py	62
7.15	problem7.py	62
7.16	problem8.py	62
7.17	units.py	63
7.18	value.py	64
7.19	value2.py	64
7.20	value3.py	65
7.21	value4.py	65
7.22	value6.py	66
7.23	webviewer.py	67
	Index	69

1 Introduction

1.1 Overview

[pyCAPS](#) is a Python extension module to interact with Computational Aircraft Prototype Syntheses (CAPS) routines in the Python environment. Written in Cython, [pyCAPS](#) natively handles all type conversions/casting, while logically grouping CAPS function calls together to simplify a user's experience. Additional functionality not directly available through the CAPS API (such as saving a geometric view) is also provided.

An overview of the basic [pyCAPS](#) functionality is provided in [Getting Started](#).

1.2 Key differences between [pyCAPS](#) and CAPS

- Manipulating the "owner" information for CAPS objects isn't currently supported
- CAPS doesn't natively support an array of string values (an array of strings is viewed by CAPS as a single concatenated string), however [pyCAPS](#) does. If a list of strings is provided this list is concatenated, separated by a ';' and provided to CAPS as a single string. The number of rows and columns are correctly set to match the original list. If a string is received from CAPS by [pyCAPS](#) and the rows and columns are set correctly it will be unpacked correctly considering entries are separated by a ';'. (Important) If the rows and columns aren't set correctly and the string contains a ';', the data will likely be unpacked incorrectly or raise an indexing error. (Note: not available when setting attributes on objects)

1.3 Clearance Statement

This software has been cleared for public release on 25 Jul 2018, case number 88ABW-2018-3793.

2 Getting Started

The following provides an overview of [pyCAPS](#)'s services, with the intention being to emphasis and focus on basic, core functionality. As such not all functions will be discussed. Users are encouraged to individually explore each classes documentation for a complete list of options.

2.1 The capsProblem Class

The capsProblem class is the front end of [pyCAPS](#). All other classes are intended to be initiated through the problem class. The following code details the primary function calls and uses when creating/setting up a new problem.

2.1.1 Initialization and termination

The first step to create a new capsProblem is to import the [pyCAPS](#) module; on Linux and OSX this is the pyCAPS.so file, while on Windows it is the pyCAPS.pyd file. For convenience, it is recommended that the path to this file is added to the environmental variable PYTHONPATH.

```
import pyCAPS
```

After the module is loaded, a new capsProblem class object should be instantiated (see [pyCAPS.capsProblem.__init__](#)). Note that multiple problems may be simultaneously loaded/exist in a single script.

```
myProblem = pyCAPS.capsProblem()
```

Once a problem has been created the public attributes of the capsProblem are accessible.

After all desired operations on the problem are finished, it is recommended to close the problem (see [pyCAPS.capsProblem.closeCAPS](#)).

```
myProblem.closeCAPS()
```

Putting it all together we get:

```
# Use: Initiate/close problem.

# Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
import pyCAPS

# Instantiate our CAPS problem "myProblem"
print("Initiating capsProblem")
myProblem = pyCAPS.capsProblem()

# Close our problem
print("Closing our problem")
myProblem.closeCAPS()
```

2.1.2 Loading the geometry

A geometry file is loaded into the problem using the loadCAPS() function for the problem (see [pyCAPS.capsProblem.loadCAPS](#)). In the example below a *.csm file, `./csmData/cfdMultiBody.csm`, is loaded into our created problem from above. The project name "basicTest" may be optionally set here; if no argument is provided the CAPS file provided is used as the project name. A reference to the newly created geometry class (see [pyCAPS._capsGeometry](#)) is stored and accessed through the geometry attribute ([pyCAPS.capsProblem.geometry](#))

```
myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
```

Alternatively, a reference to the geometry class is also returned during a call to loadCAPS() and can be used to also access the geometry class. Both the above and below code snippets are equivalent.

```
myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
```

2.1.3 Loading an AIM

Analysis interface modules (AIMs) are loaded in the problem using the `loadAIM()` function for the problem (see [pyCAPS.capsProblem.loadAIM](#)). In the code sample below, the "fun3dAIM" is loaded into the problem with a specified working directory and intention.

```
myProblem.loadAIM(aim = "fun3dAIM",
                  altName = "fun3d",
                  analysisDir = "FUN3DAnalysisTest",
                  capsIntent = "CFD")
```

Since the "altName" keyword is being specified in the above snippet this analysis instance will be referenced/bookkept in the problem's analysis dictionary ([pyCAPS.capsProblem.analysis](#)) as "fun3d". For example,

```
print(myProblem.analysis["fun3d"].analysisDir)
```

results in,

```
FUN3DAnalysisTest
```

Alternatively, a reference to the analysis class is also returned during a call to `loadAIM()` and can be used to also access the analysis class. Both the above and below code snippets are equivalent.

```
fun3d = myProblem.loadAIM(aim = "fun3dAIM",
                          altName = "fun3d",
                          analysisDir = "FUN3DAnalysisTest",
                          capsIntent = "CFD")

print(fun3d.analysisDir)
```

2.2 Working with Geometry

Once the geometry is loaded various functions are provided to interact with it (see [pyCAPS._capsGeometry](#)). The following sections highlight a few of the more common ones.

2.2.1 Setting and getting design parameters

Geometric design parameters may be set using the `setGeometryVal()` function ([pyCAPS._capsGeometry.setGeometryVal](#)), while the current value of the parameter may be retrieved using `getGeometryVal()` ([pyCAPS._capsGeometry.getGeometryVal](#)). In the following example (geometry from [Loading the geometry](#)) the current value for the parameter "lesweep" is first obtained. The value is then reset and increased by 20.0.

```
# Get current value of the leading edges sweep
value = myGeometry.getGeometryVal("lesweep")
print("Current sweep Value =", value)

# Set a new value for the leading edges sweep
myGeometry.setGeometryVal("lesweep", value + 20.0)

# Check to see if the value was set correctly
value = myGeometry.getGeometryVal("lesweep")
print("New Sweep Value =", value)
```

The results from this snippet look like:

```
Current sweep Value = 30.0
New Sweep Value = 50.0
```

2.2.2 Viewing geometry

Within Python a picture of the current geometry may be viewed and/or saved using the `viewGeometry()` function ([pyCAPS._capsGeometry.viewGeometry](#)). A representative example of the function is as follows,

```
myGeometry.viewGeometry(filename = "GeomViewDemo_NewSweep",
                        title="DESPMTR: lesweep = " + str(value),
                        showImage = True,
                        ignoreBndBox = True,
                        combineBodies = True,
                        showTess = False,
                        showAxes = False,
                        directory = "GeometryImages",
                        viewType = "fourview")
```

Upon execution of the above code an image of the current geometry is displayed on the screen (`showImage = True`). As seen below, all the bodies are combined (`combineBodies = True`) into a single image with four different view points (`viewType = "fourview"`). Since a filename is also provided, the imaged displayed on the screen is also saved.



Figure 1: Demo of `viewGeometry()`

2.3 Working with an AIM

Once an AIM has been loaded, various functions are provided to interact with it (see [pyCAPS._capsAnalysis](#)). The following sections highlight a few of the more common ones.

2.3.1 Setting and getting AIM inputs and outputs

AIM inputs may be set using the `setAnalysisVal()` function ([pyCAPS._capsAnalysis.setAnalysisVal](#)), while the current value of the input may be retrieved using `getAnalysisVal()` ([pyCAPS._capsAnalysis.getAnalysisVal](#)). In the following example (AIM from [Loading an AIM](#)) the current value for the input "Proj_Name" is first obtained and is then reset.

```
# Get current value of the project name
value = fun3d.getAnalysisVal("Proj_Name")
print("Current project name =", value)

# Set a new value for the project name
fun3d.setAnalysisVal("Proj_Name", "pyCAPS_Demo")

# Check to see if the value was set correctly
value = fun3d.getAnalysisVal("Proj_Name")
print("New project name =", value)
```

The results from this snippet look like:

```
Current project name = fun3d_CAPS
New project name = pyCAPS_Demo
```

Similar to `getAnalysisVal`, the `getAnalysisOutVal()` function ([pyCAPS._capsAnalysis.getAnalysisOutVal](#)) returns AIM output variables.

2.3.2 AIM pre- and post- analysis

For a given AIM, the CAPS pre- and post- analysis functions are executed in [pyCAPS](#) using the `preAnalysis()` ([pyCAPS._capsAnalysis.preAnalysis](#)) and `postAnalysis()` ([pyCAPS._capsAnalysis.postAnalysis](#)) functions.

```
# Execute pre-Analysis
fun3d.preAnalysis()

# Run AIM - os.system call, python interface, etc.....

# Execute post-Analysis
fun3d.postAnalysis()
```

3 Namespace Index

3.1 Packages

Here are the packages with brief descriptions (if available):

pyCAPS	
Python extension module for CAPS	5

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

_capsAnalysis	
Functions to interact with a CAPS analysis object	7
_capsBound	
Functions to interact with a CAPS bound object	18
_capsDataSet	
Functions to interact with a CAPS dataSet object	22
_capsGeometry	
Functions to interact with a CAPS geometry object	25
_capsValue	
Functions to interact with a CAPS value object	31
_capsVertexSet	
Functions to interact with a CAPS vertexSet object	34
CAPSError	
CAPS error exception class	35
capsProblem	
Defines a CAPS problem object	38
capsViewer	
Defines a CAPS viewer object	46

5 Namespace Documentation

5.1 pyCAPS Namespace Reference

Python extension module for CAPS.

Classes

- class [_capsAnalysis](#)
Functions to interact with a CAPS analysis object.
- class [_capsBound](#)
Functions to interact with a CAPS bound object.
- class [_capsDataSet](#)
Functions to interact with a CAPS dataSet object.
- class [_capsGeometry](#)
Functions to interact with a CAPS geometry object.
- class [_capsValue](#)
Functions to interact with a CAPS value object.
- class [_capsVertexSet](#)
Functions to interact with a CAPS vertexSet object.
- class [CAPSError](#)
CAPS error exception class.
- class [capsProblem](#)
Defines a CAPS problem object.
- class [capsViewer](#)
Defines a CAPS viewer object.

Functions

- def [capsConvert](#) (value, fromUnits, toUnits, ignoreWarning=True)
Convert a value from one unit to another using the UDUNITS-2 library.

5.1.1 Detailed Description

Python extension module for CAPS.

5.1.2 Function Documentation

5.1.2.1 capsConvert()

```
def pyCAPS.capsConvert (
    value,
    fromUnits,
    toUnits,
    ignoreWarning = True )
```

Convert a value from one unit to another using the UDUNITS-2 library.

See [units.py](#) for example use cases. Note that UDUNITS-2 is packaged with CAPS, so no additional dependencies are necessary. Please refer to the UDUNITS-2 documentation for specifics regarding the syntax for valid unit strings, [UDUNITS-2 Manual](#) or [NIST Units](#). The following table was taken from the UDUNITS-2 manual to assist in some of the unit specifications.

String Type	Using Names	Using Symbols	Comment
Simple	meter	m	
Raised	meter^2	m2	higher precedence than multiplying or dividing
Product	newton meter	N.m	
Quotient	meter per second	m/s	

String Type	Using Names	Using Symbols	Comment
Scaled	60 second	60 s	
Prefixed	kilometer	km	
Offset	kelvin from 273.15	K @ 273.15	lower precedence than multiplying or dividing
Logarithmic	lg(re milliwatt)	lg(re mW)	"lg" is base 10, "ln" is base e, and "lb" is base 2
Grouped	(5 meter)/(30 second)	(5 m)/(30 s)	

Parameters

<i>value</i>	Input value to convert. Value may be an integer, float/double, or list of integers and floats/doubles. Note that integers are automatically cast to floats/doubles
<i>fromUnits</i>	Current units of the input value (see [UDUNITS-2 Manual] for valid string).
<i>toUnits</i>	Desired units to convert the input value to (see [UDUNITS-2 Manual] for valid string).
<i>ignoreWarning</i>	Ignore UDUNITS verbose warnings (default - True). Errors during unit conversions are still reported.

Returns

Return the input value(s) in the specified units.

Exceptions

<i>TypeError</i>	Wrong type [float(s)/double(s)/integer(s)] for value or non-string value for from/to-Units.
<i>ValueError</i>	Error during unit conversion. See raised message for additional details.

6 Class Documentation

6.1 _capsAnalysis Class Reference

Functions to interact with a CAPS analysis object.

Public Member Functions

- def [setAnalysisVal](#) (self, varname, value, units=None)
Sets an ANALYSISIN variable for the analysis object.
- def [getAnalysisVal](#) (self, varname=None, kwargs)
Gets an ANALYSISIN variable for the analysis object.
- def [getAnalysisOutVal](#) (self, varname=None, kwargs)
Gets an ANALYSISOUT variable for the analysis object.
- def [getAnalysisInfo](#) (self, printInfo=True, kwargs)
Gets analysis information for the analysis object.
- def [getAttributeVal](#) (self, attributeName, kwargs)
Retrieve a list of geometric attribute values of a given name ("attributeName") for the bodies loaded into the analysis.
- def [getAttributeMap](#) (self, getInternal=False, kwargs)
Create geometric attribution map (embedded dictionaries) for the bodies loaded into the analysis.
- def [aimPreAnalysis](#) (self)
Alternative to [preAnalysis\(\)](#).
- def [preAnalysis](#) (self)

- *Run the pre-analysis function for the AIM.*
- def [aimPostAnalysis](#) (self)
 - *Alternative to [postAnalysis\(\)](#).*
- def [postAnalysis](#) (self)
 - *Run the post-analysis function for the AIM.*
- def [createOpenMDAOComponent](#) (self, inputVariable, outputVariable, kwargs)
 - *Create an OpenMDAO component object, an external code component ([ExternalCode](#)) is created if the `executeCommand` keyword argument is provided.*
- def [createTree](#) (self, filename="aimName", kwargs)
 - *Create a HTML dendrogram/tree of the current state of the analysis.*
- def [addAttribute](#) (self, name, data)
 - *Add an attribute (that is meta-data) to the analysis object.*
- def [getAttribute](#) (self, name)
 - *Get an attribute (that is meta-data) that exists on the analysis object.*
- def [getSensitivity](#) (self, inputVar, outputVar)
 - *Get sensitivity values.*
- def [aimBackDoor](#) (self, JSONin)
 - *Alternative to [backDoor\(\)](#).*
- def [backDoor](#) (self, JSONin)
 - *Make a call to the AIM backdoor function.*
- def [viewGeometry](#) (self, kwargs)
 - *View the geometry associated with the analysis.*
- def [saveGeometry](#) (self, filename="myGeometry", directory=os.getcwd(), extension=".egads")
 - *Save the current geometry used by the AIM to a file.*

Public Attributes

- [capsProblem](#)
 - *Reference to the problem object that loaded the AIM during a call to [capsProblem.loadAIM](#) .*
- [openMDAOComponent](#)
 - *OpenMDAO "component" object (see [createOpenMDAOComponent](#) for instantiation).*

Static Public Attributes

- [aimName](#)
 - *Reference name of AIM loaded for the analysis object (see [\\$aimName](#)).*
- [officialName](#)
 - *Name of the AIM loaded for the analysis object (see [\\$aim](#)).*
- [parents](#)
 - *List of parents of the AIM loaded for the analysis object (see [\\$parents](#)).*
- [analysisDir](#)
 - *Analysis directory of the AIM loaded for the analysis object (see [\\$analysis](#)).*
- [analysisIntent](#)
 - *Analysis intent of the AIM loaded for the analysis object (see [\\$capsIntent](#)).*
- [unitSystem](#)
 - *Unit system the AIM was loaded with (if applicable).*

6.1.1 Detailed Description

Functions to interact with a CAPS analysis object.

Should be created with [capsProblem.loadAIM](#) (not a standalone class)

6.1.2 Member Function Documentation

6.1.2.1 addAttribute()

```
def addAttribute (
    self,
    name,
    data )
```

Add an attribute (that is meta-data) to the analysis object.

See example [analysis7.py](#) for a representative use case.

Parameters

<i>name</i>	Name used to define the attribute.
<i>data</i>	Data value(s) for the attribute. Note that type casting is done automatically based on the determined type of the Python object.

6.1.2.2 aimBackDoor()

```
def aimBackDoor (
    self,
    JSONin )
```

Alternative to [backDoor\(\)](#).

Warning: May be deprecated in future versions.

6.1.2.3 aimPostAnalysis()

```
def aimPostAnalysis (
    self )
```

Alternative to [postAnalysis\(\)](#).

Warning: May be deprecated in future versions.

6.1.2.4 aimPreAnalysis()

```
def aimPreAnalysis (
    self )
```

Alternative to [preAnalysis\(\)](#).

Warning: May be deprecated in future versions.

6.1.2.5 backDoor()

```
def backDoor (
    self,
    JSONin )
```

Make a call to the AIM backdoor function.

Important: it is assumed that the JSON string returned (JSONout) by the AIM is freeable.

Parameters

<i>JSONin</i>	JSON string input to the backdoor function. If the value isn't a string (e.g. a Python dictionary) it will automatically be converted to JSON string.
---------------	---

Returns

Returns a Python object of the converted JSON out string from the back door function if any.

6.1.2.6 createOpenMDAOComponent()

```
def createOpenMDAOComponent (
    self,
    inputVariable,
    outputVariable,
    kwargs )
```

Create an OpenMDAO component object, an external code component (ExternalCode) is created if the executeCommand keyword argument is provided.

Note that this functionality is currently tied to version 1.7.3 of OpenMDAO (pip install openmdao==1.7.3), use of version 2.x will result in an import error.

Parameters

<i>inputVariable</i>	Input variable(s)/parameter(s) to add to the OpenMDAO component. Variables may be either analysis input variables or geometry design parameters. Note, that the setting of analysis inputs supersedes the setting of geometry design parameters; issues may arise if analysis input and geometry design variables have the same name. If the analysis parameter wanting to be added to the OpenMDAO component is part of a capsTuple the following notation should be used: "AnalysisInput:TupleKey:DictionaryKey", for example "AVL_Control:ControlSurfaceA:deflectionAngle" would correspond to the AVL_Control input variable, the ControlSurfaceA element of the input values (that is the name of the control surface being created) and finally deflectionAngle corresponds to the name of the dictionary entry that is to be used as the component parameter. If the tuple's value isn't a dictionary just "AnalysisInput:TupleKey" is needed.
<i>outputVariable</i>	Output variable(s)/parameter(s) to add to the OpenMDAO component. Only scalar output variables are currently supported
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>changeDir</i>	Automatically switch into the analysis directory set for the AIM when executing an external code (default - True).
<i>saveIteration</i>	If the generated OpenMDAO component is going to be called multiple times, the inputs and outputs from the analysis and the AIM will be automatically bookkept (= True) by moving the files to a folder within the AIM's analysis directory (analysisDir) named "Iteration_#" where # represents the iteration number (default - False). By default (= False) input and output files will be continuously overwritten. Notes: <ul style="list-style-type: none"> • If the AIM has 'parents' their generated files will not be bookkept. • If previous iteration folders already exist, the iteration folders and any other files in the directory will be moved to a folder named "Instance_#". • This bookkeeping method will likely fail if the iterations are run concurrently!
<i>executeCommand</i>	Command to be executed when running an external code. Command must be a list of command line arguments (see OpenMDAO documentation). If provided an ExternalCode object is created; if not provided or set to None a Component object is created (default - None).

Parameters

<i>inputFile</i>	Optional list of input file names for OpenMDAO to check the existence of before OpenMDAO executes the "solve_nonlinear" (default - None). This is redundant as the AIM automatically does this already.
<i>outputFile</i>	Optional list of output names for OpenMDAO to check the existence of before OpenMDAO executes the "solve_nonlinear" (default - None). This is redundant as the AIM automatically does this already.
<i>stdin</i>	Set I/O connection for the standard input of an ExternalCode component. The use of this depends on the expected AIM execution.
<i>stdout</i>	Set I/O connection for the standard output of an ExternalCode component. The use of this depends on the expected AIM execution.
<i>setSensitivity</i>	Optional dictionary containing sensitivity/derivative settings/parameters. See OpenMDAO documentation for a complete list of "deriv_options". Common values for a finite difference calculation would be <code>setSensitivity['type'] = "fd"</code> , <code>setSensitivity['form'] = "forward"</code> or <code>"backward"</code> or <code>"central"</code> , and <code>setSensitivity['step_size'] = 1.0E-6</code>

Returns

Optionally returns the reference to the OpenMDAO component object created. "None" is returned if a failure occurred during object creation.

6.1.2.7 `createTree()`

```
def createTree (
    self,
    filename = "aimName",
    kwargs )
```

Create a HTML dendrogram/tree of the current state of the analysis.

See example [analysis5.py](#) for a representative use case. The HTML file relies on the open-source JavaScript library, D3, to visualize the data. This library is freely available from <https://d3js.org/> and is dynamically loaded within the HTML file. If running on a machine without internet access a (miniaturized) copy of the library may be written to a file alongside the generated HTML file by setting the `internetAccess` keyword to False. If set to True, internet access will be necessary to view the tree.

Parameters

<i>filename</i>	Filename to use when saving the tree (default - "aimName"). Note an ".html" is automatically appended to the name (same with ".json" if <code>embedJSON = False</code>).
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>embedJSON</i>	Embed the JSON tree data in the HTML file itself (default - True). If set to False a separate file is generated for the JSON tree data.
<i>internetAccess</i>	Is internet access available (default - True)? If set to True internet access will be necessary to view the tree.
<i>analysisGeom</i>	Show the geometry currently load into the analysis in the tree (default - False).
<i>internalGeomAttr</i>	Show the internal attributes (denoted by starting with an underscore, for example <code>"_AttrName"</code>) that exist on the geometry (default - False). Note: "analysisGeom" must also be set to True.
<i>reverseMap</i>	Reverse the attribute map (default - False). See getAttributeMap for details.

6.1.2.8 `getAnalysisInfo()`

```
def getAnalysisInfo (
    self,
    printInfo = True,
    kwargs )
```

Gets analysis information for the analysis object.

See example [analysis6.py](#) for a representative use case.

Parameters

<i>printInfo</i>	Print information to screen if True.
<i>**kwargs</i>	See below.

Returns

Cleanliness state of analysis object or a dictionary containing analysis information (infoDict must be set to True)

Valid keywords:

Parameters

<i>infoDict</i>	Return a dictionary containing analysis information instead of just the cleanliness state (default - False)
-----------------	---

6.1.2.9 `getAnalysisOutVal()`

```
def getAnalysisOutVal (
    self,
    varname = None,
    kwargs )
```

Gets an ANALYSISOUT variable for the analysis object.

Parameters

<i>varname</i>	Name of CAPS value to retrieve from the AIM. If no name is provided a dictionary containing all ANALYSISOUT values is returned. See example analysis4.py for a representative use case.
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>units</i>	When set to True returns the units along with the value as specified by the analysis. When set to a string (e.g. units="ft") the returned value is converted into the specified units. (default - False).
<i>namesOnly</i>	Return only a list of variable names (no values) if creating a dictionary (default - False).

Returns

Value of "varname" or dictionary of all values. Units are also returned if applicable based on the "units" keyword (does not apply to dictionary returns).

6.1.2.10 `getAnalysisVal()`

```
def getAnalysisVal (
    self,
    varname = None,
    kwargs )
```

Gets an ANALYSISIN variable for the analysis object.

Parameters

<i>varname</i>	Name of CAPS value to retrieve from the AIM. If no name is provided a dictionary containing all ANALYSISIN values is returned. See example analysis4.py for a representative use case.
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>units</i>	When set to True returns the units along with the value as specified by the analysis. When set to a string (e.g. units="ft") the returned value is converted into the specified units. (default - False).
<i>namesOnly</i>	Return only a list of variable names (no values) if creating a dictionary (default - False).

Returns

Value of "varname" or dictionary of all values. Units are also returned if applicable based on the "units" keyword (does not apply to dictionary returns).

6.1.2.11 `getAttribute()`

```
def getAttribute (
    self,
    name )
```

Get an attribute (that is meta-data) that exists on the analysis object.

See example [analysis7.py](#) for a representative use case.

Parameters

<i>name</i>	Name of attribute to retrieve.
-------------	--------------------------------

Returns

Value of attribute "name".

6.1.2.12 `getAttributeMap()`

```
def getAttributeMap (
    self,
    getInternal = False,
    kwargs )
```

Create geometric attribution map (embedded dictionaries) for the bodies loaded into the analysis.

Dictionary layout:

- Body 1
 - Body : Body level attributes
 - Faces
 - * 1 : Attributes on the first face of the body
 - * 2 : Attributes on the second face of the body
 - * " : ...
 - Edges
 - * 1 : Attributes on the first edge of the body
 - * 2 : Attributes on the second edge of the body
 - * " : ...
 - Nodes :
 - * 1 : Attributes on the first node of the body
 - * 2 : Attributes on the second node of the body
 - * " : ...
- Body 2
 - Body : Body level attributes
 - Faces
 - * 1 : Attributes on the first face of the body
 - * " : ...
 - ...
- ...

Dictionary layout (reverseMap = True):

- Body 1
 - Attribute : Attribute name
 - * Value : Value of attribute
 - Body : True if value exist at body level, None if not
 - Faces : Face numbers at which the attribute exist
 - Edges : Edge numbers at which the attribute exist
 - Nodes : Node numbers at which the attribute exist
 - * Value : Next value of attribute with the same name
 - Body : True if value exist at body level, None if not
 - " : ...
 - * ...
 - Attribute : Attribute name
 - * Value : Value of attribute
 - " : ...
 - * ...
- Body 2
 - Attribute : Attribute name
 - * Value : Value of attribute
 - Body : True if value exist at body level, None if not
 - " : ...
 - * ...
 - ...
- ...

Parameters

<i>getInternal</i>	Get internal attributes (denoted by starting with an underscore, for example " <code>_AttrName</code> ") that exist on the geometry (default - False).
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>reverseMap</i>	Reverse the attribute map (default - False). See above table for details.
-------------------	---

Returns

Dictionary containing attribution map

6.1.2.13 `getAttributeVal()`

```
def getAttributeVal (
    self,
    attributeName,
    kwargs )
```

Retrieve a list of geometric attribute values of a given name ("`attributeName`") for the bodies loaded into the analysis.

Level in which to search the bodies is determined by the `attrLevel` keyword argument. See [analysis3.py](#) for a representative use case.

Parameters

<i>attributeName</i>	Name of attribute to retrieve values for.
<i>**kwargs</i>	See below.

Returns

A list of attribute values.

Valid keywords:

Parameters

<i>bodyIndex</i>	Specific body in which to retrieve attribute information from.
<i>attrLevel</i>	Level to which to search the body(ies). Options: 0 (or "Body") - search just body attributes 1 (or "Face") - search the body and all the faces [default] 2 (or "Edge") - search the body, faces, and all the edges 3 (or "Node") - search the body, faces, edges, and all the nodes

6.1.2.14 `getSensitivity()`

```
def getSensitivity (
    self,
    inputVar,
```

```
outputVar )
```

Get sensitivity values.

Note the AIM must have aimBackdoor function to interact with.

Parameters

<i>inputVar</i>	Input variable to retrieve the sensitivity with respect to.
<i>outputVar</i>	Output variable to retrieve the sensitivity value for.

Returns

Sensitivity value.

Note for AIM developers:

This function makes use of the caps_AIMbackdoor function. The JSONin variable provided is a JSON dictionary with the following form - {"mode": "Sensitivity", "outputVar": outputVar, "inputVar": inputVar}. Similarly, a JSON dictionary is expected to return (JSONout) with the following form - {"sensitivity": sensitivityVal}; furthermore it is assumed that JSON string returned (JSONout) is freeable.

6.1.2.15 postAnalysis()

```
def postAnalysis (
    self )
```

Run the post-analysis function for the AIM.

6.1.2.16 preAnalysis()

```
def preAnalysis (
    self )
```

Run the pre-analysis function for the AIM.

If the specified analysis directory doesn't exist it will be made automatically.

6.1.2.17 saveGeometry()

```
def saveGeometry (
    self,
    filename = "myGeometry",
    directory = os.getcwd(),
    extension = ".egads" )
```

Save the current geometry used by the AIM to a file.

Parameters

<i>filename</i>	File name to use when saving geometry file.
<i>directory</i>	Directory where to save file. Default current working directory.
<i>extension</i>	Extension type for file if filename does not contain an extension.

6.1.2.18 setAnalysisVal()

```
def setAnalysisVal (
    self,
```

```
varname,
value,
units = None )
```

Sets an ANALYSISIN variable for the analysis object.

Parameters

<i>varname</i>	Name of CAPS value to set in the AIM.
<i>value</i>	Value to set. Type casting is automatically done based on the CAPS value type
<i>units</i>	Applicable units of the current variable (default - None). Only applies to real values. specified in the AIM.

6.1.2.19 `viewGeometry()`

```
def viewGeometry (
    self,
    kwargs )
```

View the geometry associated with the analysis.

If the analysis produces a surface tessellation, then that is shown. Otherwise the bodies are shown with default tessellation parameters. Note that the geometry must be built and will not automatically be built by this function.

Parameters

<i>**kwargs</i>	See below.
------------------------	------------

Valid keywords:

Parameters

<i>portNumber</i>	Port number to start the server listening on (default - 7681).
-------------------	--

6.1.3 Member Data Documentation

6.1.3.1 `aimName`

```
aimName [static]
```

Reference name of AIM loaded for the analysis object (see [\\$altName](#)).

6.1.3.2 `analysisDir`

```
analysisDir [static]
```

Analysis directory of the AIM loaded for the analysis object (see [\\$analysis](#)).

If the directory does not exist it will be made automatically.

6.1.3.3 `capsProblem`

```
capsProblem
```

Reference to the problem object that loaded the AIM during a call to [capsProblem.loadAIM](#) .

6.1.3.4 officialName

`officialName` [static]

Name of the AIM loaded for the analysis object (see [\\$aim](#)).

6.1.3.5 openMDAOComponent

`openMDAOComponent`

OpenMDAO "component" object (see [createOpenMDAOComponent](#) for instantiation).

6.1.3.6 parents

`parents` [static]

List of parents of the AIM loaded for the analysis object (see [\\$parents](#)).

6.1.3.7 unitSystem

`unitSystem` [static]

Unit system the AIM was loaded with (if applicable).

The documentation for this class was generated from the following file:

- `/Users/haimes/svn/CAPS/trunk/pyCAPS/src/pyCAPS.pyx`

6.2 _capsBound Class Reference

Functions to interact with a CAPS bound object.

Public Member Functions

- def [getBoundInfo](#) (self, printInfo=True, kwargs)
Gets bound information for the bound object.
- def [fillVertexSets](#) (self)
Populates VertexSets for the bound.
- def [executeTransfer](#) (self, variableName=None)
Execute data transfer for the bound.
- def [viewData](#) (self, variableName=None, kwargs)
Visualize data in the bound.
- def [writeTecplot](#) (self, filename, variableName=None)
Write a Tecplot compatible file for the data in the bound.
- def [createTree](#) (self, filename="boundName", kwargs)
Create a HTML dendrogram/tree of the current state of the bound.

Public Attributes

- [boundName](#)
Bound/transfer name used to set up the data bound.

- `capsProblem`
Reference to the problem object (`pyCAPS.capsProblem`) the bound belongs to.
- `variables`
List of variables in the bound.
- `vertexSet`
Dictionary of vertex set object (`pyCAPS._capsVertexSet`) in the bound.
- `dataSetSrc`
Dictionary of "source" data set objects (`pyCAPS._capsDataSet`) in the bound.
- `dataSetDest`
Dictionary of "destination" data set objects (`pyCAPS._capsDataSet`) in the bound.

6.2.1 Detailed Description

Functions to interact with a CAPS bound object.

Should be created with `capsProblem.createDataBound` or `capsProblem.createDataTransfer` (not a standalone class).

6.2.2 Member Function Documentation

6.2.2.1 `createTree()`

```
def createTree (
    self,
    filename = "boundName",
    kwargs )
```

Create a HTML dendrogram/tree of the current state of the bound.

The HTML file relies on the open-source JavaScript library, D3, to visualize the data. This library is freely available from <https://d3js.org/> and is dynamically loaded within the HTML file. If running on a machine without internet access a (miniaturized) copy of the library may be written to a file alongside the generated HTML file by setting the `internetAccess` keyword to False. If set to True, internet access will be necessary to view the tree.

Parameters

<code>filename</code>	Filename to use when saving the tree (default - "boundName"). Note an ".html" is automatically appended to the name (same with ".json" if <code>embedJSON</code> = False).
<code>**kwargs</code>	See below.

Valid keywords:

Parameters

<code>embedJSON</code>	Embed the JSON tree data in the HTML file itself (default - True). If set to False a separate file is generated for the JSON tree data.
<code>internetAccess</code>	Is internet access available (default True)? If set to True internet access will be necessary to view the tree.

6.2.2.2 `executeTransfer()`

```
def executeTransfer (
```

```

        self,
        variableName = None )

```

Execute data transfer for the bound.

Parameters

<i>variableName</i>	Name of variable to implement the data transfer for. If no name is provided the first variable in bound is used.
---------------------	--

6.2.2.3 fillVertexSets()

```

def fillVertexSets (
    self )

```

Populates VertexSets for the bound.

Must be called to finalize the bound after all mesh generation aim's have been executed

6.2.2.4 getBoundInfo()

```

def getBoundInfo (
    self,
    printInfo = True,
    kwargs )

```

Gets bound information for the bound object.

Parameters

<i>printInfo</i>	Print information to screen if True.
<i>**kwargs</i>	See below.

Returns

State of bound object.

Valid keywords:

Parameters

<i>infoDict</i>	Return a dictionary containing bound information instead of just the state (default - False)
-----------------	--

6.2.2.5 viewData()

```

def viewData (
    self,
    variableName = None,
    kwargs )

```

Visualize data in the bound.

The function currently relies on matplotlib or the capViewer class (webviewer) to plot the data.

Parameters

<i>variableName</i>	Name of variable to visualize. If no name is provided the first variable in the bound is used.
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>viewerType</i>	What viewer should be used (default - capsViewer). Options: capsViewer or matplotlib (options are case insensitive). Important: if <code>\$filename</code> isn't set to None, the default is changed to matplotlib.
<i>portNumber</i>	Port number to start the server listening on (default - 7681).
<i>filename</i>	Save image(s) to file specified (default - None). Not available when using the webviewer
<i>colorMap</i>	Valid string for a, matplotlib::cm, colormap (default - 'Blues'). Not as options are available when using the webviewer (see capsViewer for additional details).
<i>showImage</i>	Show image(s) (default - True).

6.2.2.6 `writeTecplot()`

```
def writeTecplot (
    self,
    filename,
    variableName = None )
```

Write a Tecplot compatible file for the data in the bound.

Parameters

<i>filename</i>	Name of file to save data to.
<i>variableName</i>	Single or list of variables to write data for. If no name is provided all variables in the bound are used.

6.2.3 Member Data Documentation

6.2.3.1 `boundName`

`boundName`

Bound/transfer name used to set up the data bound.

6.2.3.2 `capsProblem`

`capsProblem`

Reference to the problem object ([pyCAPS.capsProblem](#)) the bound belongs to.

6.2.3.3 `dataSetDest`

`dataSetDest`

Dictionary of "destination" data set objects ([pyCAPS._capsDataSet](#)) in the bound.

6.2.3.4 `dataSetSrc`

`dataSetSrc`

Dictionary of "source" data set objects ([pyCAPS._capsDataSet](#)) in the bound.

6.2.3.5 variables

`variables`

List of variables in the bound.

6.2.3.6 vertexSet

`vertexSet`

Dictionary of vertex set object ([pyCAPS._capsVertexSet](#)) in the bound.

The documentation for this class was generated from the following file:

- `/Users/haimes/svn/CAPS/trunk/pyCAPS/src/pyCAPS.pyx`

6.3 _capsDataSet Class Reference

Functions to interact with a CAPS dataSet object.

Public Member Functions

- `def initDataSet (self, initValue)`
Executes caps_initDataSet on data set object to set an initial constant value needed for cyclic data transfers.
- `def getData (self)`
Executes caps_getData on data set object to retrieve data set variable, [dataSetName](#).
- `def getDataXYZ (self)`
Executes caps_getData on data set object to retrieve XYZ coordinates of the data set.
- `def getDataConnect (self)`
Executes caps_triangulate on data set's vertex set to retrieve the connectivity (triangles only) information for the data set.
- `def viewData (self, fig=None, numDataSet=1, dataSetIndex=0, kwargs)`
Visualize data set.
- `def writeTecplot (self, file=None, filename=None)`
Write data set to a Tecplot compatible data file.

Public Attributes

- `dataSetName`
Data set name (variable name).
- `capsBound`
Reference to the bound object ([pyCAPS._capsBound](#)) that data set pertains to.
- `capsVertexSet`
Reference to the vertex set object ([pyCAPS._capsVertexSet](#)) that data set pertains to.
- `dataSetMethod`
Data method: Analysis, Interpolate, Conserve.
- `dataRank`
Rank of data set.

6.3.1 Detailed Description

Functions to interact with a CAPS `dataSet` object.

Should be initiated within `pyCAPS._capsBound` (not a standalone class)

6.3.2 Member Function Documentation

6.3.2.1 `getData()`

```
def getData (
    self )
```

Executes `caps_getData` on data set object to retrieve data set variable, `dataSetName`.

Returns

Optionally returns a list of data values. Data with a rank greater than 1 returns a list of lists (e.g. data representing a displacement would return [[Node1_xDisplacement, Node1_yDisplacement, Node1_zDisplacement], [Node2_xDisplacement, Node2_yDisplacement, Node2_zDisplacement], etc.]

6.3.2.2 `getDataConnect()`

```
def getDataConnect (
    self )
```

Executes `caps_triangulate` on data set's vertex set to retrieve the connectivity (triangles only) information for the data set.

Returns

Optionally returns a list of lists of connectivity values (e.g. [[node1, node2, node3], [node2, node3, node7], etc.]) and a list of lists of data connectivity (not this is an empty list if the data is node-based) (eg. [[node1, node2, node3], [node2, node3, node7], etc.]

6.3.2.3 `getDataXYZ()`

```
def getDataXYZ (
    self )
```

Executes `caps_getData` on data set object to retrieve XYZ coordinates of the data set.

Returns

Optionally returns a list of lists of x,y, z values (e.g. [[x2, y2, z2], [x2, y2, z2], [x3, y3, z3], etc.])

6.3.2.4 `initDataSet()`

```
def initDataSet (
    self,
    initValue )
```

Executes `caps_initDataSet` on data set object to set an initial constant value needed for cyclic data transfers.

6.3.2.5 viewData()

```
def viewData (
    self,
    fig = None,
    numDataSet = 1,
    dataSetIndex = 0,
    kwargs )
```

Visualize data set.

The function currently relies on matplotlib to plot the data.

Parameters

<i>fig</i>	Figure object (matplotlib::figure) to append image to.
<i>numDataSet</i>	Number of data sets in \$fig .
<i>dataSetIndex</i>	Index of data set being added to \$fig .
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>filename</i>	Save image(s) to file specified (default - None).
<i>colorMap</i>	Valid string for a, matplotlib::cm, colormap (default - 'Blues').
<i>showImage</i>	Show image(s) (default - True).
<i>title</i>	Set a custom title on the plot (default - VertexSet= 'name', DataSet = 'name', (Var. '#')).

6.3.2.6 writeTecplot()

```
def writeTecplot (
    self,
    file = None,
    filename = None )
```

Write data set to a Tecplot compatible data file.

A triangulation of the data set will be used for the connectivity.

Parameters

<i>file</i>	Optional open file object to append data to. If not provided a filename must be given via the keyword argument \$filename .
<i>filename</i>	Write Tecplot file with the specified name.

6.3.3 Member Data Documentation

6.3.3.1 capsBound

capsBound

Reference to the bound object ([pyCAPS._capsBound](#)) that data set pertains to.

6.3.3.2 `capsVertexSet``capsVertexSet`

Reference to the vertex set object ([pyCAPS._capsVertexSet](#)) that data set pertains to.

6.3.3.3 `dataRank``dataRank`

Rank of data set.

6.3.3.4 `dataSetMethod``dataSetMethod`

Data method: Analysis, Interpolate, Conserve.

6.3.3.5 `dataSetName``dataSetName`

Data set name (variable name).

The documentation for this class was generated from the following file:

- `/Users/haimes/svn/CAPS/trunk/pyCAPS/src/pyCAPS.pyx`

6.4 `_capsGeometry` Class Reference

Functions to interact with a CAPS geometry object.

Public Member Functions

- `def buildGeometry` (self)
Manually force a build of the geometry.
- `def saveGeometry` (self, filename="myGeometry", directory=os.getcwd(), extension=".egads")
Save the current geometry to a file.
- `def setGeometryVal` (self, varname=None, value=None)
Sets a GEOMETRYIN variable for the geometry object.
- `def getGeometryVal` (self, varname=None, kwargs)
Gets a GEOMETRYIN variable for the geometry object.
- `def getGeometryOutVal` (self, varname=None, kwargs)
Gets a GEOMETRYOUT variable for the geometry object.
- `def viewGeometry` (self, kwargs)
View or take a screen shot of the geometry configuration.
- `def getAttributeVal` (self, attributeName, kwargs)
Retrieve a list of attribute values of a given name ("attributeName") for the bodies in the current geometry.
- `def getAttributeMap` (self, getInternal=False, kwargs)
Create attribution map (embedded dictionaries) of each body in the current geometry.
- `def createTree` (self, filename="myGeometry", kwargs)
Create a HTML dendrogram/tree of the current state of the geometry.

Public Attributes

- [geomName](#)

Geometry file loaded into problem.

6.4.1 Detailed Description

Functions to interact with a CAPS geometry object.

Should be created with [capsProblem.loadCAPS](#) (not a standalone class).

6.4.2 Member Function Documentation

6.4.2.1 buildGeometry()

```
def buildGeometry (
    self )
```

Manually force a build of the geometry.

The geometry will only be rebuilt if a design parameter (see [setGeometryVal](#)) has been changed.

6.4.2.2 createTree()

```
def createTree (
    self,
    filename = "myGeometry",
    kwargs )
```

Create a HTML dendrogram/tree of the current state of the geometry.

The HTML file relies on the open-source JavaScript library, D3, to visualize the data. This library is freely available from <https://d3js.org/> and is dynamically loaded within the HTML file. If running on a machine without internet access a (miniaturized) copy of the library may be written to a file alongside the generated HTML file by setting the internetAccess keyword to False. If set to True, internet access will be necessary to view the tree.

Parameters

<i>filename</i>	Filename to use when saving the tree (default - "myGeometry"). Note an ".html" is automatically appended to the name (same with ".json" if embedJSON = False).
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>embedJSON</i>	Embed the JSON tree data in the HTML file itself (default - True). If set to False a separate file is generated for the JSON tree data.
<i>internetAccess</i>	Is internet access available (default - True)? If set to True internet access will be necessary to view the tree.
<i>internalGeomAttr</i>	Show the internal attributes (denoted by starting with an underscore, for example "_AttrName") that exist on the geometry (default - False).
<i>reverseMap</i>	Reverse the attribute map (default - False). See getAttributeMap for details.

6.4.2.3 `getAttributeMap()`

```
def getAttributeMap (
    self,
    getInternal = False,
    kwargs )
```

Create attribution map (embeded dictionaries) of each body in the current geometry.

Dictionary layout:

- Body 1
 - Body : Body level attributes
 - Faces
 - * 1 : Attributes on the first face of the body
 - * 2 : Attributes on the second face of the body
 - * " : ...
 - Edges
 - * 1 : Attributes on the first edge of the body
 - * 2 : Attributes on the second edge of the body
 - * " : ...
 - Nodes :
 - * 1 : Attributes on the first node of the body
 - * 2 : Attributes on the second node of the body
 - * " : ...
- Body 2
 - Body : Body level attributes
 - Faces
 - * 1 : Attributes on the first face of the body
 - * " : ...
 - ...
- ...

Dictionary layout (`reverseMap = True`):

- Body 1
 - Attribute : Attribute name
 - * Value : Value of attribute
 - Body : True if value exist at body level, None if not
 - Faces : Face numbers at which the attribute exist
 - Edges : Edge numbers at which the attribute exist
 - Nodes : Node numbers at which the attribute exist
 - * Value : Next value of attribute with the same name
 - Body : True if value exist at body level, None if not
 - " : ...
 - * ...
 - Attribute : Attribute name
 - * Value : Value of attribute
 - " : ...
 - * ...

- Body 2
 - Attribute : Attribute name
 - * Value : Value of attribute
 - Body : True if value exist at body level, None if not
 - " : ...
 - * ...
 - ...
- ...

Parameters

<i>getInternal</i>	Get internal attributes (denoted by starting with an underscore, for example "_AttrName") that exist on the geometry (default - False).
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>reverseMap</i>	Reverse the attribute map (default - False). See above table for details.
-------------------	---

Returns

Dictionary containing attribution map

6.4.2.4 getAttributeVal()

```
def getAttributeVal (
    self,
    attributeName,
    kwargs )
```

Retrieve a list of attribute values of a given name ("attributeName") for the bodies in the current geometry. Level in which to search the bodies is determined by the attrLevel keyword argument.

Parameters

<i>attributeName</i>	Name of attribute to retrieve values for.
<i>**kwargs</i>	See below.

Returns

A list of attribute values.

Valid keywords:

Parameters

<i>bodyIndex</i>	Specific body in which to retrieve attribute information from.
------------------	--

Parameters

<i>attrLevel</i>	Level to which to search the body(ies). Options: 0 (or "Body") - search just body attributes 1 (or "Face") - search the body and all the faces [default] 2 (or "Edge") - search the body, faces, and all the edges 3 (or "Node") - search the body, faces, edges, and all the nodes
------------------	---

6.4.2.5 getGeometryOutVal()

```
def getGeometryOutVal (
    self,
    varname = None,
    kwargs )
```

Gets a GEOMETRYOUT variable for the geometry object.

Parameters

<i>varname</i>	Name of geometry (local) parameter to retrieve from the geometry. If no name is provided a dictionary containing all local variables is returned.
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>namesOnly</i>	Return only a list of parameter names (no values) if creating a dictionary (default - False).
<i>ignoreAt</i>	Ignore @ geometry variables when creating a dictionary (default - True).

Returns

Value of "varname" or dictionary of all values.

6.4.2.6 getGeometryVal()

```
def getGeometryVal (
    self,
    varname = None,
    kwargs )
```

Gets a GEOMETRYIN variable for the geometry object.

Parameters

<i>varname</i>	Name of geometry design parameter to retrieve from the geometry. If no name is provided a dictionary containing all design parameters is returned.
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>namesOnly</i>	Return only a list of parameter names (no values) if creating a dictionary (default - False).
------------------	---

Returns

Value of "varname" or dictionary of all values.

6.4.2.7 saveGeometry()

```
def saveGeometry (
    self,
    filename = "myGeometry",
    directory = os.getcwd(),
    extension = ".egads" )
```

Save the current geometry to a file.

Parameters

<i>filename</i>	File name to use when saving geometry file.
<i>directory</i>	Directory where to save file. Default current working directory.
<i>extension</i>	Extension type for file if filename does not contain an extension.

6.4.2.8 setGeometryVal()

```
def setGeometryVal (
    self,
    varname = None,
    value = None )
```

Sets a GEOMETRYIN variable for the geometry object.

Parameters

<i>varname</i>	Name of geometry design parameter to set in the geometry.
<i>value</i>	Value of geometry design parameter. Type casting is automatically done based on value indicated by CAPS.

6.4.2.9 viewGeometry()

```
def viewGeometry (
    self,
    kwargs )
```

View or take a screen shot of the geometry configuration.

The use of this function requires the **matplotlib** module. **Important:** If both showImage = True and filename is not None, any manual view changes made by the user in the displayed image will be reflected in the saved image.

Parameters

**kwargs	See below.
-----------------	------------

Valid keywords:

Parameters

<i>viewerType</i>	What viewer should be used (default - <code>capsViewer</code>). Options: <code>capsViewer</code> or <code>matplotlib</code> (options are case insensitive). Important: if <code>\$filename</code> is not <code>None</code> , the viewer is changed to <code>matplotlib</code> .
<i>portNumber</i>	Port number to start the server listening on (default - 7681).
<i>title</i>	Title to add to each figure (default - <code>None</code>).
<i>filename</i>	Save image(s) to file specified (default - <code>None</code>). Note filename should not contain '.' other than to indicate file type extension (default type = *.png). 'file' - OK, 'file2.0Test' - BAD, 'file2_0Test.png' - OK, 'file2.0Test.jpg' - BAD.
<i>directory</i>	Directory path were to save file. If the directory doesn't exist it will be made. (default - current directory).
<i>viewType</i>	Type of view for the image(s). Options: "isometric" (default), "fourview", "top" (or "-zaxis"), "bottom" (or "+zaxis"), "right" (or "+yaxis"), "left" (or "-yaxis"), "front" (or "+xaxis"), "back" (or "-xaxis").
<i>combineBodies</i>	Combine all bodies into a single image (default - <code>False</code>).
<i>ignoreBndBox</i>	Ignore the largest body (default - <code>False</code>).
<i>showImage</i>	Show image(s) (default - <code>False</code>).
<i>showAxes</i>	Show the xyz axes in the image(s) (default - <code>False</code>).
<i>showTess</i>	Show the edges of the tessellation (default - <code>False</code>).
<i>dpi</i>	Resolution in dots-per-inch for the figure (default - <code>None</code>).
<i>tessParam</i>	Custom tessellation parameters, see EGADS documentation for <code>makeTessBody</code> function. values will be scaled by the norm of the bounding box for the body (default - [0.0250, 0.0010, 15.0]).

6.4.3 Member Data Documentation

6.4.3.1 `geomName``geomName`

Geometry file loaded into problem.

Note that the directory path has been removed.

The documentation for this class was generated from the following file:

- `/Users/haimes/svn/CAPS/trunk/pyCAPS/src/pyCAPS.pyx`

6.5 `_capsValue` Class Reference

Functions to interact with a CAPS value object.

Public Member Functions

- def `setVal` (self, data)
Change the value of the object.
- def `getVal` (self)
Get the current value of object.
- def `setLimits` (self, newLimits)
Set new limits.

- def [getLimits](#) (self)
Get the current value for the limits.
- def [convertUnits](#) (self, toUnits)
Return the current value of the object in the desired, specified units.

Public Attributes

- [capsProblem](#)
Reference to the problem object that loaded the value.
- [units](#)
Units of the variable.
- [limits](#)
Acceptable limits for the value.
- [value](#)
Value of the variable.

Static Public Attributes

- [name](#)
Variable name.

6.5.1 Detailed Description

Functions to interact with a CAPS value object.

Should be created with [capsProblem.createValue](#) (not a standalone class)

6.5.2 Member Function Documentation

6.5.2.1 [convertUnits\(\)](#)

```
def convertUnits (  
    self,  
    toUnits )
```

Return the current value of the object in the desired, specified units.

Note that this neither changes the value or units of the object, only returns a converted value. See [value4.py](#) for a representative use case.

Returns

Current value of the object in the specified units.

6.5.2.2 [getLimits\(\)](#)

```
def getLimits (  
    self )
```

Get the current value for the limits.

See [value3.py](#) for a representative use case.

Returns

Current value for the limits.

6.5.2.3 `getVal()`

```
def getVal (
    self )
```

Get the current value of object.

See [value2.py](#) for a representative use case.

Returns

Current value of set for object.

6.5.2.4 `setLimits()`

```
def setLimits (
    self,
    newLimits )
```

Set new limits.

See [value3.py](#) for a representative use case.

Parameters

<i>newLimits</i>	New values to set for the limits. Should be 2 element list - [min value, max value].
------------------	--

6.5.2.5 `setVal()`

```
def setVal (
    self,
    data )
```

Change the value of the object.

See [value2.py](#) for a representative use case.

Parameters

<i>data</i>	Data value(s) for the variable. Note that data will be type casted to match the type used to original create the capsValue object.
-------------	--

6.5.3 Member Data Documentation**6.5.3.1 `capsProblem`**

[capsProblem](#)

Reference to the problem object that loaded the value.

6.5.3.2 limits

limits

Acceptable limits for the value.

Limits may be set directly. See [value3.py](#) for a representative use case.

6.5.3.3 value

value

Value of the variable.

Value may be set directly. See [value2.py](#) for a representative use case.

The documentation for this class was generated from the following file:

- `/Users/haimes/svn/CAPS/trunk/pyCAPS/src/pyCAPS.pyx`

6.6 `_capsVertexSet` Class Reference

Functions to interact with a CAPS vertexSet object.

Public Attributes

- [vertexSetName](#)
Vertex set name (analysis name the vertex belongs to).
- [capsBound](#)
Bound object ([pyCAPS._capsBound](#)) the vertex set belongs to.
- [capsAnalysis](#)
Analysis object ([pyCAPS._capsAnalysis](#)) the vertex set belongs to.

6.6.1 Detailed Description

Functions to interact with a CAPS vertexSet object.

Should be initiated within [pyCAPS._capsBound](#) (not a standalone class)

6.6.2 Member Data Documentation

6.6.2.1 capsAnalysis

capsAnalysis

Analysis object ([pyCAPS._capsAnalysis](#)) the vertex set belongs to.

6.6.2.2 capsBound

capsBound

Bound object ([pyCAPS._capsBound](#)) the vertex set belongs to.

6.6.2.3 vertexSetName

vertexSetName

Vertex set name (analysis name the vertex belongs to).

The documentation for this class was generated from the following file:

- /Users/haimes/svn/CAPS/trunk/pyCAPS/src/pyCAPS.pyx

6.7 CAPSError Class Reference

CAPS error exception class.

Inherits Exception.

Public Member Functions

- def `__init__` (self, code=None, msg=None)
Initialize the [CAPSError](#) exception.

Public Attributes

- [errorCode](#)
Error code encountered when running the CAPS.
- [errorMessage](#)
Name of error code encountered.

Static Public Attributes

- dictionary [capsError](#)
Dictionary of CAPS errors {errorCode : errorMessage}.
- dictionary [egadsError](#)
Dictionary of EGADS errors {errorCode : errorMessage}.
- dictionary [ocsmError](#)
Dictionary of OCSM errors {errorCode : errorMessage}.

6.7.1 Detailed Description

CAPS error exception class.

See [exception.py](#) for a representative use case.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 `__init__()`

```
def __init__ (
    self,
    code = None,
    msg = None )
```

Initialize the [CAPSError](#) exception.

6.7.3 Member Data Documentation

6.7.3.1 capsError

dictionary capsError [static]

Initial value:

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

Dictionary of CAPS errors {errorCode : errorName}.

6.7.3.2 egadsError

dictionary egadsError [static]

Initial value:

```
= { -31 : 'EGADS_TESSTATE',
-30 : 'EGADS_EXISTS',
-29 : 'EGADS_ATTRERR',
-28 : 'EGADS_TOPOCNT',
-27 : 'EGADS_OCSEFLT',
-26 : 'EGADS_BADSCALE',
-25 : 'EGADS_NOTORTHO',
-24 : 'EGADS_DEGEN',
-23 : 'EGADS_CONSTERR',
-22 : 'EGADS_TOPOERR',
-21 : 'EGADS_GEOMERR',
-20 : 'EGADS_NOTBODY',
-19 : 'EGADS_WRITERR',
-18 : 'EGADS_NOTMODEL',
-17 : 'EGADS_NOLOAD',
-16 : 'EGADS_RANGERR',
-15 : 'EGADS_NOTGEOM',
-14 : 'EGADS_NOTTESS',
-13 : 'EGADS_EMPTY',
-12 : 'EGADS_NOTTOPO',
```

```

-11 : 'EGADS_REFERCE',
-10 : 'EGADS_NOTXFORM',
-9 : 'EGADS_NOTCNTX',
-8 : 'EGADS_MIXCNTX',
-7 : 'EGADS_NODATA',
-6 : 'EGADS_NONAME',
-5 : 'EGADS_INDEXERR',
-4 : 'EGADS_MALLOCC',
-3 : 'EGADS_NOTOBJ',
-2 : 'EGADS_NULLOBJ',
-1 : 'EGADS_NOTFOUND',
0 : 'EGADS_SUCCESS',
1 : 'EGADS_OUTSIDE',

cEGADS.PRM_NOTCONVERGED : 'PRM_NOTCONVERGED',
cEGADS.PRM_BADNUMVERTICES : 'PRM_BADNUMVERTICES',
cEGADS.PRM_ZEROPIVOT : 'PRM_ZEROPIVOT',
cEGADS.PRM_NEGATIVEAREAS : 'PRM_NEGATIVEAREAS',
cEGADS.PRM_TOLERANCEUNMET : 'PRM_TOLERANCEUNMET',
cEGADS.PRM_NOGLOBALUV : 'PRM_NOGLOBALUV',
cEGADS.PRM_BADPARAM : 'PRM_BADPARAM',
cEGADS.PRM_BADDIVISION : 'PRM_BADDIVISION',
cEGADS.PRM_CANNOTFORMLOOP : 'PRM_CANNOTFORMLOOP',
cEGADS.PRM_WIGGLEDETECTED : 'PRM_WIGGLEDETECTED',
cEGADS.PRM_INTERNAL : 'PRM_INTERNAL' }

```

Dictionary of EGADS errors {errorCode : errorName}.

6.7.3.3 ocsmError

dictionary ocsmError [static]

Initial value:

```

= {cOCSM.OCSM_FILE_NOT_FOUND : 'OCSM_FILE_NOT_FOUND',
  cOCSM.OCSM_ILLEGAL_STATEMENT : 'OCSM_ILLEGAL_STATEMENT',
  cOCSM.OCSM_NOT_ENOUGH_ARGS : 'OCSM_NOT_ENOUGH_ARGS',
  cOCSM.OCSM_NAME_ALREADY_DEFINED : 'OCSM_NAME_ALREADY_DEFINED',
  cOCSM.OCSM_NESTED_TOO_DEEPLY : 'OCSM_NESTED_TOO_DEEPLY',
  cOCSM.OCSM_IMPROPER_NESTING : 'OCSM_IMPROPER_NESTING',
  cOCSM.OCSM_NESTING_NOT_CLOSED : 'OCSM_NESTING_NOT_CLOSED',
  cOCSM.OCSM_NOT_MODL_STRUCTURE : 'OCSM_NOT_MODL_STRUCTURE',
  cOCSM.OCSM_PROBLEM_CREATING_PERTURB : 'OCSM_PROBLEM_CREATING_PERTURB',

  cOCSM.OCSM_MISSING_MARK : 'OCSM_MISSING_MARK',
  cOCSM.OCSM_INSUFFICIENT_BODYDYS_ON_STACK : 'OCSM_INSUFFICIENT_BODYDYS_ON_STACK',
  cOCSM.OCSM_WRONG_TYPES_ON_STACK : 'OCSM_WRONG_TYPES_ON_STACK',
  cOCSM.OCSM_DID_NOT_CREATE_BODY : 'OCSM_DID_NOT_CREATE_BODY',
  cOCSM.OCSM_CREATED_TOO_MANY_BODYDYS : 'OCSM_CREATED_TOO_MANY_BODYDYS',
  cOCSM.OCSM_TOO_MANY_BODYDYS_ON_STACK : 'OCSM_TOO_MANY_BODYDYS_ON_STACK',
  cOCSM.OCSM_ERROR_IN_BODYDYS_ON_STACK : 'OCSM_ERROR_IN_BODYDYS_ON_STACK',
  cOCSM.OCSM_MODL_NOT_CHECKED : 'OCSM_MODL_NOT_CHECKED',
  cOCSM.OCSM_NEED_TESSELLATION : 'OCSM_NEED_TESSELLATION',

  cOCSM.OCSM_BODY_NOT_FOUND : 'OCSM_BODY_NOT_FOUND',
  cOCSM.OCSM_FACE_NOT_FOUND : 'OCSM_FACE_NOT_FOUND',
  cOCSM.OCSM_EDGE_NOT_FOUND : 'OCSM_EDGE_NOT_FOUND',
  cOCSM.OCSM_NODE_NOT_FOUND : 'OCSM_NODE_NOT_FOUND',
  cOCSM.OCSM_ILLEGAL_VALUE : 'OCSM_ILLEGAL_VALUE',
  cOCSM.OCSM_ILLEGAL_ATTRIBUTE : 'OCSM_ILLEGAL_ATTRIBUTE',
  cOCSM.OCSM_ILLEGAL_CSYSYSTEM : 'OCSM_ILLEGAL_CSYSYSTEM',

  cOCSM.OCSM_SKETCH_IS_OPEN : 'OCSM_SKETCH_IS_OPEN',
  cOCSM.OCSM_SKETCH_IS_NOT_OPEN : 'OCSM_SKETCH_IS_NOT_OPEN',
  cOCSM.OCSM_COLINEAR_SKETCH_POINTS : 'OCSM_COLINEAR_SKETCH_POINTS',
  cOCSM.OCSM_NON_COPLANAR_SKETCH_POINTS : 'OCSM_NON_COPLANAR_SKETCH_POINTS',
  cOCSM.OCSM_TOO_MANY_SKETCH_POINTS : 'OCSM_TOO_MANY_SKETCH_POINTS',
  cOCSM.OCSM_TOO_FEW_SPLINE_POINTS : 'OCSM_TOO_FEW_SPLINE_POINTS',
  cOCSM.OCSM_SKETCH_DOES_NOT_CLOSE : 'OCSM_SKETCH_DOES_NOT_CLOSE',
  cOCSM.OCSM_SELF_INTERSECTING : 'OCSM_SELF_INTERSECTING',
  cOCSM.OCSM_ASSERT_FAILED : 'OCSM_ASSERT_FAILED',

  cOCSM.OCSM_ILLEGAL_CHAR_IN_EXPR : 'OCSM_ILLEGAL_CHAR_IN_EXPR',
  cOCSM.OCSM_CLOSE_BEFORE_OPEN : 'OCSM_CLOSE_BEFORE_OPEN',
  cOCSM.OCSM_MISSING_CLOSE : 'OCSM_MISSING_CLOSE',
  cOCSM.OCSM_ILLEGAL_TOKEN_SEQUENCE : 'OCSM_ILLEGAL_TOKEN_SEQUENCE',
  cOCSM.OCSM_ILLEGAL_NUMBER : 'OCSM_ILLEGAL_NUMBER',
  cOCSM.OCSM_ILLEGAL_PMTR_NAME : 'OCSM_ILLEGAL_PMTR_NAME',
  cOCSM.OCSM_ILLEGAL_FUNC_NAME : 'OCSM_ILLEGAL_FUNC_NAME',
  cOCSM.OCSM_ILLEGAL_TYPE : 'OCSM_ILLEGAL_TYPE',
  cOCSM.OCSM_ILLEGAL_NARG : 'OCSM_ILLEGAL_NARG',

```



```

cOCSM.OCSM_NAME_NOT_FOUND          : 'OCSM_NAME_NOT_FOUND',
cOCSM.OCSM_NAME_NOT_UNIQUE         : 'OCSM_NAME_NOT_UNIQUE',
cOCSM.OCSM_PMTR_IS_EXTERNAL        : 'OCSM_PMTR_IS_EXTERNAL',
cOCSM.OCSM_PMTR_IS_INTERNAL        : 'OCSM_PMTR_IS_INTERNAL',
cOCSM.OCSM_PMTR_IS_CONSTANT        : 'OCSM_PMTR_IS_CONSTANT',
cOCSM.OCSM_WRONG_PMTR_TYPE         : 'OCSM_WRONG_PMTR_TYPE',
cOCSM.OCSM_FUNC_ARG_OUT_OF_BOUNDS  : 'OCSM_FUNC_ARG_OUT_OF_BOUNDS',
cOCSM.OCSM_VAL_STACK_UNDERFLOW     : 'OCSM_VAL_STACK_UNDERFLOW',
cOCSM.OCSM_VAL_STACK_OVERFLOW      : 'OCSM_VAL_STACK_OVERFLOW',

cOCSM.OCSM_ILLEGAL_BRCH_INDEX      : 'OCSM_ILLEGAL_BRCH_INDEX',
cOCSM.OCSM_ILLEGAL_PMTR_INDEX      : 'OCSM_ILLEGAL_PMTR_INDEX',
cOCSM.OCSM_ILLEGAL_BODY_INDEX      : 'OCSM_ILLEGAL_BODY_INDEX',
cOCSM.OCSM_ILLEGAL_ARG_INDEX       : 'OCSM_ILLEGAL_ARG_INDEX',
cOCSM.OCSM_ILLEGAL_ACTIVITY        : 'OCSM_ILLEGAL_ACTIVITY',
cOCSM.OCSM_ILLEGAL_MACRO_INDEX     : 'OCSM_ILLEGAL_MACRO_INDEX',
cOCSM.OCSM_ILLEGAL_ARGUMENT        : 'OCSM_ILLEGAL_ARGUMENT',
cOCSM.OCSM_CANNOT_BE_SUPPRESSED    : 'OCSM_CANNOT_BE_SUPPRESSED',
cOCSM.OCSM_STORAGE_ALREADY_USED    : 'OCSM_STORAGE_ALREADY_USED',
cOCSM.OCSM_NOTHING_PREVIOUSLY_STORED : 'OCSM_NOTHING_PREVIOUSLY_STORED',

cOCSM.OCSM_SOLVER_IS_OPEN          : 'OCSM_SOLVER_IS_OPEN',
cOCSM.OCSM_SOLVER_IS_NOT_OPEN      : 'OCSM_SOLVER_IS_NOT_OPEN',
cOCSM.OCSM_TOO_MANY_SOLVER_VARS    : 'OCSM_TOO_MANY_SOLVER_VARS',
cOCSM.OCSM_UNDERCONSTRAINED        : 'OCSM_UNDERCONSTRAINED',
cOCSM.OCSM_OVERCONSTRAINED         : 'OCSM_OVERCONSTRAINED',
cOCSM.OCSM_SINGULAR_MATRIX         : 'OCSM_SINGULAR_MATRIX',
cOCSM.OCSM_NOT_CONVERGED           : 'OCSM_NOT_CONVERGED',

cOCSM.OCSM_UDP_ERROR1              : 'OCSM_UDP_ERROR1',
cOCSM.OCSM_UDP_ERROR2              : 'OCSM_UDP_ERROR2',
cOCSM.OCSM_UDP_ERROR3              : 'OCSM_UDP_ERROR3',
cOCSM.OCSM_UDP_ERROR4              : 'OCSM_UDP_ERROR4',
cOCSM.OCSM_UDP_ERROR5              : 'OCSM_UDP_ERROR5',
cOCSM.OCSM_UDP_ERROR6              : 'OCSM_UDP_ERROR6',
cOCSM.OCSM_UDP_ERROR7              : 'OCSM_UDP_ERROR7',
cOCSM.OCSM_UDP_ERROR8              : 'OCSM_UDP_ERROR8',
cOCSM.OCSM_UDP_ERROR9              : 'OCSM_UDP_ERROR9',

cOCSM.OCSM_OP_STACK_UNDERFLOW      : 'OCSM_OP_STACK_UNDERFLOW',
cOCSM.OCSM_OP_STACK_OVERFLOW       : 'OCSM_OP_STACK_OVERFLOW',
cOCSM.OCSM_RPN_STACK_UNDERFLOW     : 'OCSM_RPN_STACK_UNDERFLOW',
cOCSM.OCSM_RPN_STACK_OVERFLOW      : 'OCSM_RPN_STACK_OVERFLOW',
cOCSM.OCSM_TOKEN_STACK_UNDERFLOW   : 'OCSM_TOKEN_STACK_UNDERFLOW',
cOCSM.OCSM_TOKEN_STACK_OVERFLOW    : 'OCSM_TOKEN_STACK_OVERFLOW',
cOCSM.OCSM_UNSUPPORTED             : 'OCSM_UNSUPPORTED',
cOCSM.OCSM_INTERNAL_ERROR          : 'OCSM_INTERNAL_ERROR' }

```

Dictionary of OCSM errors {errorCode : errorName}.

The documentation for this class was generated from the following file:

- /Users/haimes/svn/CAPS/trunk/pyCAPS/src/pyCAPS.pyx

6.8 capsProblem Class Reference

Defines a CAPS problem object.

Public Member Functions

- def **__init__** (self, libDir=None, [raiseException](#)=True)
Initialize the problem.
- def **loadCAPS** (self, capsFile, projectName=None, verbosity=None)
*Loads a *.csm, *.caps, or *.egads file into the problem.*
- def **setVerbosity** (self, verbosityLevel)
Set the verbosity level of the CAPS output.
- def **saveCAPS** (self, filename="saveCAPS.caps")
Save a CAPS problem.
- def **closeCAPS** (self)
Close a CAPS problem.

- def [dirtyAnalysis](#) (self)
Report what analyses loaded into the problem are dirty.
- def [loadAIM](#) (self, kwargs)
Load an AIM (Analysis Interface Module) into the problem.
- def [createDataTransfer](#) (self, kwargs)
Alteranative to [createDataBound](#).
- def [createDataBound](#) (self, kwargs)
Create a CAPS data bound/transfer into the problem.
- def [createValue](#) (self, name, data, units=None, limits=None, fixedLength=True, fixedShape=True)
Create a CAPS value object.
- def [autoLinkValue](#) (self, [value](#)=None)
*Create a link between a created CAPS value parameter and analysis inputs of **all** loaded AIMs, automatically.*
- def [addAttribute](#) (self, name, data)
Add an attribute (that is meta-data) to the problem object.
- def [getAttribute](#) (self, name)
Get an attribute (that is meta-data) that exists on the problem object.
- def [createTree](#) (self, filename="myProblem", kwargs)
Create a HTML dendrogram/tree of the current state of the problem.

Public Attributes

- [status](#)
Current CAPS status code.
- [raiseException](#)
Raise an exception after a CAPS error is found (default - True).
- [geometry](#)
Geometry object loaded into the problem.
- [aimGlobalCount](#)
Number of AIMs loaded into the problem.
- [analysisDir](#)
Current analysis directory which was used to load the latest AIM.
- [capsIntent](#)
Default intent used to load the AIM if capsIntent is not provided.
- [analysis](#)
Dictionary of analysis objects loaded into the problem.
- [dataBound](#)
Dictionary of data transfer/bound objects loaded into the problem.
- [value](#)
Dictionary of value objects loaded into the problem.

6.8.1 Detailed Description

Defines a CAPS problem object.

A [capsProblem](#) is the top-level object for a single mission/problem. It maintains a single set of interrelated geometric models (see [pyCAPS._capsGeometry](#)), analyses to be executed (see [pyCAPS._capsAnalysis](#)), connectivity and data (see [pyCAPS._capsBound](#)) associated with the run(s), which can be both multi-fidelity and multi-disciplinary.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 `__init__()`

```
def __init__ (
    self,
    libDir = None,
    raiseException = True )
```

Initialize the problem.

See [problem.py](#) for a representative use case.

Parameters

<i>libDir</i>	Deprecated option, no longer required.
<i>raiseException</i>	Raise an exception after a CAPS error is encountered (default - True). See raiseException .

6.8.3 Member Function Documentation

6.8.3.1 `addAttribute()`

```
def addAttribute (
    self,
    name,
    data )
```

Add an attribute (that is meta-data) to the problem object.

See example [problem7.py](#) for a representative use case.

Parameters

<i>name</i>	Name used to define the attribute.
<i>data</i>	Data value(s) for the attribute. Note that type casting is done automatically based on the determined type of the Python object.

6.8.3.2 `autoLinkValue()`

```
def autoLinkValue (
    self,
    value = None )
```

Create a link between a created CAPS value parameter and analysis inputs of **all** loaded AIMs, automatically.

Valid CAPS value, parameter objects must be created with [createValue\(\)](#). Note, only links to ANALYSISIN inputs are currently made at this time. See [value6.py](#) for a representative use case.

Parameters

<i>value</i>	Value to use when creating the link (default - None). A combination (i.e. a single or list) of value dictionary entries and/or value object instances (returned from a call to createValue()) can be used. If no value is provided, all entries in the value dictionary (value) will be used.
--------------	--

6.8.3.3 closeCAPS()

```
def closeCAPS (
    self )
```

Close a CAPS problem.

See [problem1.py](#) for a representative use case.

6.8.3.4 createDataBound()

```
def createDataBound (
    self,
    kwargs )
```

Create a CAPS data bound/transfer into the problem.

Parameters

**kwargs	See below.
-----------------	------------

Valid keywords:

Parameters

<i>capsBound</i>	Name of capsBound to use for the data bound.
<i>variableName</i>	Single or list of variables names to add.
<i>aimSrc</i>	Single or list of AIM names that will be the data sources for the bound.
<i>aimDest</i>	Single or list of AIM names that will be the data destinations during the transfer.
<i>transferMethod</i>	Single or list of transfer methods to use during the transfer.
<i>initValueDest</i>	Single or list of initial values for the destination data.

Returns

Optionally returns the reference to the data bound dictionary ([dataBound](#)) entry created for the bound class object ([pyCAPS._capsBound](#)).

6.8.3.5 createDataTransfer()

```
def createDataTransfer (
    self,
    kwargs )
```

Alteranative to [createDataBound](#).

Enforces that at least 2 AIMs must be already loaded into the problem. See [createDataBound](#) for details.

6.8.3.6 createTree()

```
def createTree (
    self,
    filename = "myProblem",
    kwargs )
```

Create a HTML dendrogram/tree of the current state of the problem.

See example [problem6.py](#) for a representative use case. The HTML file relies on the open-source JavaScript library, D3, to visualize the data. This library is freely available from <https://d3js.org/> and is dynamically loaded within the HTML file. If running on a machine without internet access a (miniaturized) copy of the library may be

written to a file alongside the generated HTML file by setting the `internetAccess` keyword to `False`. If set to `True`, internet access will be necessary to view the tree.

Parameters

<i>filename</i>	Filename to use when saving the tree (default - "myProblem"). Note an ".html" is automatically appended to the name (same with ".json" if <code>embedJSON</code> = <code>False</code>).
<i>**kwargs</i>	See below.

Valid keywords:

Parameters

<i>embedJSON</i>	Embed the JSON tree data in the HTML file itself (default - <code>True</code>). If set to <code>False</code> a separate file is generated for the JSON tree data.
<i>internetAccess</i>	Is internet access available (default - <code>True</code>)? If set to <code>True</code> internet access will be necessary to view the tree.
<i>analysisGeom</i>	Show the geometry for each analysis entity (default - <code>False</code>).
<i>internalGeomAttr</i>	Show the internal attributes (denoted by starting with an underscore, for example "_AttrName") that exist on the geometry (default - <code>False</code>).
<i>reverseMap</i>	Reverse the geometry attribute map (default - <code>False</code>).

6.8.3.7 createValue()

```
def createValue (
    self,
    name,
    data,
    units = None,
    limits = None,
    fixedLength = True,
    fixedShape = True )
```

Create a CAPS value object.

Only a value of subtype in `PARAMETER` is currently supported. See [value.py](#) for a representative use case. Value objects are stored the [value](#) dictionary.

Parameters

<i>name</i>	Name used to define the value. This will be used as the keyword entry in the value dictionary.
<i>data</i>	Data value(s) for the variable. Note that type casting is done automatically based on the determined type of the Python object. Be careful with the Integers and Floats/Reals, for example 10 would be type casted as an Integer, while 10.0 would be a float - this small discrepancy may lead to type errors when linking values to analysis inputs.
<i>units</i>	Units associated with the value (default - <code>None</code>).
<i>limits</i>	Valid/acceptable range for the value (default - <code>None</code>).
<i>fixedLength</i>	Should the length of the value object be fixed (default - <code>True</code>)? For example if the object is initialized with a value of <code>[1, 2]</code> it can not be changed to <code>[1, 2, 3]</code> .
<i>fixedShape</i>	Should the shape of the value object be fixed (default - <code>True</code>)? For example if the object is initialized with a value of 1 it can not be changed to <code>[1, 2]</code> or <code>[[1, 2, 3], [4, 5, 6]]</code> can not be changed to <code>[[1, 2], [4, 5]]</code> .

Returns

Optionally returns the reference to the value dictionary ([value](#)) entry created for the value ([pyCAPS._caps←Value](#)).

6.8.3.8 dirtyAnalysis()

```
def dirtyAnalysis (
    self )
```

Report what analyses loaded into the problem are dirty.

Returns

Optionally returns a list of names of the dirty analyses. An empty list is returned if no analyses are dirty.

6.8.3.9 getAttribute()

```
def getAttribute (
    self,
    name )
```

Get an attribute (that is meta-data) that exists on the problem object.

See example [problem7.py](#) for a representative use case.

Parameters

<i>name</i>	Name of attribute to retrieve.
-------------	--------------------------------

Returns

Value of attribute "name"

6.8.3.10 loadAIM()

```
def loadAIM (
    self,
    kwargs )
```

Load an AIM (Analysis Interface Module) into the problem.

See examples [problem3.py](#) and [problem4.py](#) for typical representative use cases.

Parameters

<i>**kwargs</i>	See below.
------------------------	------------

Valid keywords:

Parameters

<i>aim</i>	Name of the requested AIM.
<i>altName</i>	Alternative name to use when referencing AIM inside the problem (dictionary key in analysis). The name of the AIM, <i>aim</i> , will be used if no \$altName is provided (see remarks).

Parameters

<i>analysisDir</i>	Directory for AIM analysis. If none is provided the directory of the last loaded AIM will be used; if no AIMs have been load the current working directory is used. CAPS requires that an unique directory be specified for each instance of AIM.
<i>capsIntent</i>	Analysis intention in which to invoke the AIM.
<i>parents</i>	Single or list of parent AIM names to initilize the AIM with.
<i>copyAIM</i>	Name of AIM to copy. Creates the new AIM instance by duplicating an existing AIM. Analysis directory (\$analysisDir) and \$altName should be provided and different from the AIM being copied. See example analysis2.py for a representative use case.
<i>copyParents</i>	When copying an AIM, should the same parents also be used (default - True).

Returns

Optionally returns the reference to the analysis dictionary ([analysis](#)) entry created for the analysis class object ([pyCAPS._capsAnalysis](#)).

Remarks

If no [\\$altName](#) is provided and an AIM with the name, [\\$aim](#), has already been loaded, an alternative name will be automatically specified with the syntax [\\$aim_\[instance number\]](#). If an [\\$altName](#) is provided it must be unique compared to other instances of the loaded aim.

6.8.3.11 loadCAPS()

```
def loadCAPS (
    self,
    capsFile,
    projectName = None,
    verbosity = None )
```

Loads a *.csm, *.caps, or *.egads file into the problem.

See [problem1.py](#), [problem2.py](#), and [problem8.py](#) for example use cases.

Parameters

<i>capsFile</i>	CAPS file to load. Options: *.csm, *.caps, or *.egads. If the filename has a *.caps extension the pyCAPS analysis, bound, and value objects will be re-populated (see remarks).
<i>projectName</i>	CAPS project name. projectName=capsFile if not provided.
<i>verbosity</i>	Level of output verbosity. See setVerbosity .

Returns

Optionally returns the reference to the [geometry](#) class object ([pyCAPS._capsGeometry](#)).

Remarks

Caveats of loading an existing CAPS file:

- Can currently only load *.caps files generated from [pyCAPS](#) originally.
- OpenMDAO objects won't be re-populated for analysis objects

6.8.3.12 saveCAPS()

```
def saveCAPS (
    self,
    filename = "saveCAPS.caps" )
```

Save a CAPS problem.

See [problem8.py](#) for example use case.

Parameters

<i>filename</i>	File name to use when saving CAPS problem.
-----------------	--

6.8.3.13 setVerbosity()

```
def setVerbosity (
    self,
    verbosityLevel )
```

Set the verbosity level of the CAPS output.

See [problem5.py](#) for a representative use case.

Parameters

<i>verbosityLevel</i>	Level of output verbosity. Options: 0 (or "minimal"), 1 (or "standard") [default], and 2 (or "debug").
-----------------------	--

6.8.4 Member Data Documentation**6.8.4.1 aimGlobalCount**

```
aimGlobalCount
```

Number of AIMs loaded into the problem.

6.8.4.2 analysis

```
analysis
```

Dictionary of analysis objects loaded into the problem.

Set via [loadAIM](#).

6.8.4.3 analysisDir

```
analysisDir
```

Current analysis directory which was used to load the latest AIM.

6.8.4.4 capsIntent

```
capsIntent
```

Default intent used to load the AIM if capsIntent is not provided.

6.8.4.5 dataBound

`dataBound`

Dictionary of data transfer/bound objects loaded into the problem.

Set via [createDataBound](#) or [createDataTransfer](#).

6.8.4.6 geometry

`geometry`

Geometry object loaded into the problem.

Set via [loadCAPS](#).

6.8.4.7 raiseException

`raiseException`

Raise an exception after a CAPS error is found (default - True).

Disabling (i.e. setting to False) may have unexpected consequences; in general the value should be set to True.

6.8.4.8 value

`value`

Dictionary of value objects loaded into the problem.

Set via [createValue](#).

The documentation for this class was generated from the following file:

- `/Users/haimes/svn/CAPS/trunk/pyCAPS/src/pyCAPS.pyx`

6.9 capsViewer Class Reference

Defines a CAPS viewer object.

Public Member Functions

- `def __init__ (self, browserName=None, portNumber=7681, oneBias=True, html=None)`
Initialize the viewer object.
- `def startServer (self, enableCheck=True, unifyBoundingBox=True)`
Start the server.
- `def addVertex (self, xyz, name=None)`
Add a vertex set.
- `def addIndex (self, connectivity, name=None)`
Add a element connectivity set.
- `def addLineIndex (self, connectivity, name=None)`
Add a element connectivity set with the intention of creating graphical lines on triangular primitives.
- `def addPointIndex (self, connectivity, name=None)`
Add a point connectivity set with the intention of creating graphical points on line or triangular primitives.
- `def addColor (self, colorData, minColor=None, maxColor=None, numContour=0, reverseMap=False, color↔Map="blues", name=None)`
Add color/scalar data set.
- `def addLineColor (self, colorData, minColor=None, maxColor=None, numContour=0, reverseMap=False, colorMap="blues", name=None)`

- Add color/scalar data set for the graphical lines on triangular primitives.*

 - def [addPointColor](#) (self, colorData, minColor=None, maxColor=None, numContour=0, reverseMap=False, colorMap="blues", name=None)
- Add color/scalar data set for the graphical points on line or triangular primitives.*

 - def [createPrimitive](#) (self, type, items=None, name=None, lines=True)

Create a generic graphic primitive.
- def [createPoint](#) (self, items=None, name=None)

Create a point graphic primitive.
- def [createLine](#) (self, items=None, name=None, turnOn=True)

Create a line graphic primitive.
- def [createTriangle](#) (self, items=None, name=None, turnOn=True, lines=True)

Create a triangular graphic primitive.
- def [clearItems](#) (self)

Clear the "current" list of "wvData items (objects)".
- def [addDataSet](#) (self, dataSet)

Add a [_capsDataSet](#) (see [_capsDataSet](#)) object(s).
- def [addBound](#) (self, bound, dataSetType="both")

Alias to [addDataBound](#).
- def [addDataBound](#) (self, bound, dataSetType="both")

Add a [_capsBound](#) (see [_capsBound](#)) object(s).
- def [addTecplot](#) (self, filename)

Add a Tecplot file.
- def [addUnstructMesh](#) (self, meshFile)

Add a unstructured mesh file.
- def [addGeometry](#) (self, geomFile, flags=0, dictFile=None)

Add a geometry file.

Static Public Attributes

- [colorMap](#)
- Ryan: how do I set pSize = 6 ?!?!?*
- [numTess](#)
- Display an array of EGADS bodies.*

6.9.1 Detailed Description

Defines a CAPS viewer object.

A [capsViewer](#) object is a Pythonized version of Bob Haimes's "wv: A General Web-based 3D Viewer" API. wv's goal is "to generate a visual tool targeted for the 3D needs found within the MDAO process. A WebBrowser-based approach is considered, in that it provides the broadest possible platform for deployment."

6.9.2 Constructor & Destructor Documentation

6.9.2.1 `__init__()`

```
def __init__ (
    self,
    browserName = None,
    portNumber = 7681,
    oneBias = True,
    html = None )
```

Initialize the viewer object.

See [webviewer.py](#) for an example use case.

Parameters

<i>browserName</i>	Name of browser to load (default - None). If left as None the system level default browser will be used.
<i>portNumber</i>	Port number to start the server listening on (default - 7681).
<i>oneBias</i>	Flag to indicate the index biasing of the data. Options: 1 bias (default - True) or 0 bias (False) indexed.
<i>html</i>	Specify an alternative HTML file to launch when starting the server, see startServer (default - None or env \$CAPS_START)

6.9.3 Member Function Documentation

6.9.3.1 `addColor()`

```
def addColor (
    self,
    colorData,
    minColor = None,
    maxColor = None,
    numContour = 0,
    reverseMap = False,
    colorMap = "blues",
    name = None )
```

Add color/scalar data set.

See [webviewer.py](#) for an example use case.

Parameters

<i>colorData</i>	A list of color/scalar data to be applied at each node (e.g. [node1_Color, node2_Color, node3_Color, etc.]). Cell-centered coloring isn't currently supported. Alternatively, a Hex string (e.g. '#ff0000' == the color red) may be used to set all nodes to a constant color.
<i>minColor</i>	Minimum color value to use for scaling (default - None). If None, the minimum value is determined automatically from the colorData provided. When plotting multiple color/data sets a minimum value should be provided to ensure that all data is scaled the same.
<i>maxColor</i>	Maximum color value to use for scaling (default - None). If None, the maximum value is determined automatically from the colorData provided. When plotting multiple color/data sets a maximum value should be provided to ensure that all data is scaled the same.
<i>numContour</i>	Number of contour levels to use in the color map (default - 0 for a continuous color map).
<i>reverseMap</i>	Reverse or invert the color map (default - False).
<i>colorMap</i>	Name of color map to use of visualization. Options: "blues" (default), "reds", "greys" (or "grays"), "blue_red" (or "bwr"), "jet", "rainbow", "hot", "cool".
<i>name</i>	Name of color set (default - None).

Returns

Optionally returns a reference to the "vvData item (object)" in which the data is stored.

6.9.3.2 addDataBound()

```
def addDataBound (
    self,
    bound,
    dataSetType = "both" )
```

Add a [_capsBound](#) (see [_capsBound](#)) object(s).

Parameters

<i>bound</i>	A single or list of instances of _capsBound objects.
<i>dataSetType</i>	Specifies which type of data sets in the bound should be added, source or destination. Options: "source", "destination, or "both" (default) - values are case insensitive.

6.9.3.3 addDataSet()

```
def addDataSet (
    self,
    dataSet )
```

Add a [_capsDataSet](#) (see [_capsDataSet](#)) object(s).

A graphic primitive will be created automatically for each data set. Note, however, if multiple data sets share the same vertex set (see [_capsVertexSet](#)) the data in the repeated vertex sets will be appended into a single primitive.

Parameters

<i>dataSet</i>	A single or list of instances of _capsDataSet objects.
----------------	--

6.9.3.4 addGeometry()

```
def addGeometry (
    self,
    geomFile,
    flags = 0,
    dictFile = None )
```

Add a geometry file.

Files are loaded by extension: *.igs, *.iges, *.stp, *.step, *.brep (for native OpenCASCADE files), *.egads (for native files with persistent Attributes, splits ignored), or *.csm (ESP file)

Parameters

<i>geomFile</i>	Name of geometry file to load.
<i>flags</i>	Options (default - None) : 1 - Don't split closed and periodic entities 2 - Split to maintain at least C1 in BSPLINES 4 - Don't try maintaining Units on STEP read (always millimeters)
<i>dictFile</i>	Dictionary file name when loading *.csm files, if applicable (default - None)

6.9.3.5 addIndex()

```
def addIndex (
    self,
    connectivity,
    name = None )
```

Add a element connectivity set.

See [webviewer.py](#) for an example use case.

Parameters

<i>connectivity</i>	A list of lists of connectivity information (e.g. [[node1, node2, node3], [node2, node3, node7, node8], etc.]). Elements that >3 edges will internally be decomposed into triangles. Note that whether or not the connectivity information is one biased was specified during capsViewer initialization (see capsViewer.__init__), no further checks are currently made.
<i>name</i>	Name of connectivity set (default - None).

Returns

Optionally returns a reference to the "wvData item (object)" in which the data is stored.

6.9.3.6 addLineColor()

```
def addLineColor (
    self,
    colorData,
    minColor = None,
    maxColor = None,
    numContour = 0,
    reverseMap = False,
    colorMap = "blues",
    name = None )
```

Add color/scalar data set for the graphical lines on triangular primitives.

([addLineIndex](#)). See [webviewer.py](#) for an example use case.

Parameters

<i>colorData</i>	A list of color/scalar data to be applied at each node (e.g. [node1_Color, node2_Color, node3_Color, etc.]). Alternatively, a Hex string (e.g. '#ff0000' == the color red) may be used to set all nodes to a constant color.
<i>minColor</i>	Minimum color value to use for scaling (default - None). If None, the minimum value is determined automatically from the colorData provided. When plotting multiple color/data sets a minimum value should be provided to ensure that all data is scaled the same.
<i>maxColor</i>	Maximum color value to use for scaling (default - None). If None, the maximum value is determined automatically from the colorData provided. When plotting multiple color/data sets a maximum value should be provided to ensure that all data is scaled the same.
<i>numContour</i>	Number of contour levels to use in the color map (default - 0 for a continuous color map).
<i>reverseMap</i>	Reverse or invert the color map (default - False).
<i>colorMap</i>	Name of color map to use of visualization. Options: "blues" (default), "reds", "greys" (or "grays"), "blue_red" (or "bwr"), "jet", "rainbow", "hot", "cool".
<i>name</i>	Name of color set (default - None).

Returns

Optionally returns a reference to the "vvData item (object)" in which the data is stored.

6.9.3.7 addLineIndex()

```
def addLineIndex (
    self,
    connectivity,
    name = None )
```

Add a element connectivity set with the intention of creating graphical lines on triangular primitives.

See [webviewer.py](#) for an example use case.

Parameters

<i>connectivity</i>	A list of lists of connectivity information (e.g. [[node1, node2, node3], [node2, node3, node7, node8], etc.]). Note that whether or not the connectivity information is one biased was specified during capsViewer initialization (see capsViewer.__init__), no further checks are currently made.
<i>name</i>	Name of line set (default - None).

Returns

Optionally returns a reference to the "vvData item (object)" in which the data is stored.

6.9.3.8 addPointColor()

```
def addPointColor (
    self,
    colorData,
    minColor = None,
    maxColor = None,
    numContour = 0,
    reverseMap = False,
    colorMap = "blues",
    name = None )
```

Add color/scalar data set for the graphical points on line or triangular primitives.

([addPointIndex](#)). See [webviewer.py](#) for an example use case.

Parameters

<i>colorData</i>	A list of color/scalar data to be applied at each node (e.g. [node1_Color, node2_Color, node3_Color, etc.]). Alternatively, a Hex string (e.g. '#ff0000' == the color red) may be used to set all nodes to a constant color.
<i>minColor</i>	Minimum color value to use for scaling (default - None). If None, the minimum value is determined automatically from the colorData provided. When plotting multiple color/data sets a minimum value should be provided to ensure that all data is scaled the same.
<i>maxColor</i>	Maximum color value to use for scaling (default - None). If None, the maximum value is determined automatically from the colorData provided. When plotting multiple color/data sets a maximum value should be provided to ensure that all data is scaled the same.
<i>numContour</i>	Number of contour levels to use in the color map (default - 0 for a continuous color map).
<i>reverseMap</i>	Reverse or invert the color map (default - False).
<i>colorMap</i>	Name of color map to use of visualization. Options: "blues" (default), "reds", "greys" (or "grays"), "blue_red" (or "bwr"), "jet", "rainbow", "hot", "cool".

Parameters

<i>name</i>	Name of color set (default - None).
-------------	-------------------------------------

Returns

Optionally returns a reference to the "wvData item (object)" in which the data is stored.

6.9.3.9 addPointIndex()

```
def addPointIndex (
    self,
    connectivity,
    name = None )
```

Add a point connectivity set with the intention of creating graphical points on line or triangular primitives.

Parameters

<i>connectivity</i>	A list of connectivity information (e.g. [node1, node2, node3, etc.]). Note that whether or not the connectivity information is one biased was specified during capsViewer initialization (see capsViewer.__init__), no further checks are currently made.
<i>name</i>	Name of point set (default - None).

Returns

Optionally returns a reference to the "wvData item (object)" in which the data is stored.

6.9.3.10 addTecplot()

```
def addTecplot (
    self,
    filename )
```

Add a Tecplot file.

Parameters

<i>filename</i>	Name of Tecplot file to load.
-----------------	-------------------------------

6.9.3.11 addUnstructMesh()

```
def addUnstructMesh (
    self,
    meshFile )
```

Add a unstructured mesh file.

Note that in most cases the mesh's name (see `capsUnstructMesh.$name`) will be used for the name of the primitive.

Parameters

<i>meshFile</i>	Name of unstructured mesh file to load or an instance of a <code>capsUnstructMesh</code> (see <code>capsUnstructMesh</code>) object.
-----------------	---

6.9.3.12 addVertex()

```
def addVertex (
    self,
    xyz,
    name = None )
```

Add a vertex set.

See [webviewer.py](#) for an example use case.

Parameters

<i>xyz</i>	A list of lists of x,y, z values (e.g. [[x1, y1, z1], [x2, y2, z2],[x3, y3, z3], etc.])
<i>name</i>	Name of vertex set (default - None).

Returns

Optionally returns a reference to the "wvData item (object)" in which the data is stored.

6.9.3.13 clearItems()

```
def clearItems (
    self )
```

Clear the "current" list of "wvData items (objects)".

6.9.3.14 createLine()

```
def createLine (
    self,
    items = None,
    name = None,
    turnOn = True )
```

Create a line graphic primitive.

Parameters

<i>items</i>	List of "wvData items (objects)" used to create the triangle (default - None). If none are provided all current items added will be used!
<i>name</i>	Name of graphic primitive (default - GPrim_#, where # is 1 + number of primitives previously loaded). Important: If specified, the name must be unique.

Returns

Optionally returns a reference to the graphicPrimitive object.

6.9.3.15 createPoint()

```
def createPoint (
    self,
```



```

        items = None,
        name = None )

```

Create a point graphic primitive.

Parameters

<i>items</i>	List of "vvData items (objects)" used to create the triangle (default - None). If none are provided all current items added will be used!
<i>name</i>	Name of graphic primitive (default - GPrim_#, where # is 1 + number of primitives previously loaded). Important: If specified, the name must be unique.

Returns

Optionally returns a reference to the graphicPrimitive object.

6.9.3.16 createPrimitive()

```

def createPrimitive (
    self,
    type,
    items = None,
    name = None,
    lines = True )

```

Create a generic graphic primitive.

Parameters

<i>type</i>	Type of primitive to create. Options: "point", "line", or "triangle" (case insensitive)
<i>items</i>	List of "vvData items (objects)" used to create the triangle (default - None). If none are provided all current items added will be used!
<i>name</i>	Name of graphic primitive (default - GPrim_#, where # is 1 + number of primitives previously loaded). Important: If specified, the name must be unique.
<i>lines</i>	Lines in the primitive on or off

6.9.3.17 createTriangle()

```

def createTriangle (
    self,
    items = None,
    name = None,
    turnOn = True,
    lines = True )

```

Create a triangular graphic primitive.

See [webviewer.py](#) for an example use case.

Parameters

<i>items</i>	List of "vvData items (objects)" used to create the triangle (default - None). If none are provided all current items added will be used!
<i>name</i>	Name of graphic primitive (default - GPrim_#, where # is 1 + number of primitives previously loaded). Important: If specified, the name must be unique.
<i>lines</i>	Lines in the primitive on or off

Returns

Optionally returns a reference to the `graphicPrimitive` object.

6.9.3.18 startServer()

```
def startServer (
    self,
    enableCheck = True,
    unifyBoundingBox = True )
```

Start the server.

The port number specified when initializing the viewer object will be used (see [capsViewer.__init__](#)). Note that the server will continue to run as long as the browser is still connected, once the connection is broken the `capViewer` object should be deleted! See [webviewer.py](#) for an example use case.

Parameters

<i>enableCheck</i>	Enable checks to ensure all primitives have the same number colors and that the limits for each color are initially the same.
<i>unifyBoundingBox</i>	Unify the bounding boxes of all primitives (so all images are within the viewing area)

6.9.4 Member Data Documentation**6.9.4.1 numTess**

```
numTess [static]
```

Display an array of EGADS bodies.

Parameters

<i>numBody</i>	Number of bodies in array.
<i>body</i>	Array of EGADS body objects. Display an array of EGADS tessellations
<i>numTess</i>	Number of tessellations in array.
<i>tess</i>	Array of EGADS tessellation objects.

The documentation for this class was generated from the following file:

- `/Users/haimes/svn/CAPS/trunk/pyCAPS/src/pyCAPS.pyx`

7 Example Documentation

7.1 analysis2.py

Duplicate an AIM using the `$copyAIM` keyword argument of the `pyCAPS.capsProblem.loadAIM()` function.

```
1 # Use: Duplicate an AIM
2
3 from __future__ import print_function
4
5 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
6 import pyCAPS
```

```

7
8 # Instantiate our CAPS problem "myProblem"
9 print("Initiating capsProblem")
10 myProblem = pyCAPS.capsProblem()
11
12 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
13 # project name "basicTest" may be optionally set here; if no argument is provided
14 # the CAPS file provided is used as the project name.
15 print("Loading file into our capsProblem")
16 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
17
18 # Load FUN3D aim
19 fun3d = myProblem.loadAIM(aim = "fun3dAIM",
20                           altName = "fun3d",
21                           analysisDir = "FUN3DAnalysisTest",
22                           capsIntent = "CFD")
23
24 # Print out memory
25 print(fun3d)
26
27 print("Duplicate")
28 # Duplicate the fun3d aim using the caps_dupAnalysis function
29 fun3d2 = myProblem.loadAIM(altName = "fun3d2",
30                            analysisDir = "FUN3DAnalysisTest2",
31                            copyAIM = "fun3d")
32 # Print out memory
33 print(fun3d2)
34
35 # Close our problems
36 print("Closing our problem")
37 myProblem.closeCAPS()
38

```

7.2 analysis3.py

Example use case of the `pyCAPS._capsAnalysis.getAttributeVal()` function

```

1 # Use: Retrieve attributes from the geometry loaded into an analysis object
2
3 # Make print statement Python 2-3 agnostic
4 from __future__ import print_function
5
6 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
7 import pyCAPS
8
9 # Instantiate our CAPS problem "myProblem"
10 myProblem = pyCAPS.capsProblem()
11
12 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem.
13 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm")
14
15 # Load FUN3D aim
16 fun3d = myProblem.loadAIM(aim = "fun3dAIM",
17                           altName = "fun3d",
18                           analysisDir = "FUN3DAnalysisTest",
19                           capsIntent = "CFD")
20
21 # Retrieve "capsGroup" attributes for geometry loaded into fun3dAIM.
22 # Only search down to the body level
23 attributeList = fun3d.getAttributeVal("capsGroup", attrLevel = "Body")
24 print(attributeList)
25
26 # Provide a wrong attribute level - issue warning and default to "Face"
27 attributeList = fun3d.getAttributeVal("capsGroup", attrLevel = 5)
28 print(attributeList)
29
30 # Retrieve "capsGroup" attributes for geometry loaded into fun3dAIM.
31 # Only search down to the node level
32 attributeList = fun3d.getAttributeVal("capsGroup", attrLevel = "Node")
33 print(attributeList)
34
35 # Close our problem
36 myProblem.closeCAPS()

```

7.3 analysis4.py

Example use cases for interacting the `pyCAPS._capsAnalysis.getAnalysisVal()` and `pyCAPS._capsAnalysis.getAnalysisOutVal()` functions.

```

1 from __future__ import print_function
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
11 # project name "basicTest" may be optionally set here; if no argument is provided
12 # the CAPS file provided is used as the project name.
13 print("Loading file into our capsProblem")
14 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
15
16 # Load FUN3D aim
17 fun3d = myProblem.loadAIM(aim = "fun3dAIM",
18                           altName = "fun3d",
19                           analysisDir = "FUN3DAnalysisTest",
20                           capsIntent = "CFD")
21
22
24 valueDict = fun3d.getAnalysisVal()
25
26 print("Variable Dict = ", valueDict)
27
28
29
31 valueList = fun3d.getAnalysisVal(namesOnly = True)
32
33 print("Variable List =", valueList)
34
35
36
37
39 valueList = fun3d.getAnalysisOutVal(namesOnly = True)
40
41 print("Variable List =", valueList)
42
43
44
46 valueDict = fun3d.getAnalysisOutVal()
47
48 print("Variable Dict = ", valueDict)
49
50
51 # Close our problems
52 print("Closing our problem")
53 myProblem.closeCAPS()
54

```

7.4 analysis5.py

Example use cases for the `pyCAPS._capsAnalysis.createTree()` function.

```

1 from __future__ import print_function
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
11 # project name "basicTest" may be optionally set here; if no argument is provided
12 # the CAPS file provided is used as the project name.
13 print("Loading file into our capsProblem")
14 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
15
16 # Load FUN3D aim
17 fun3d = myProblem.loadAIM(aim = "fun3dAIM",
18                           altName = "fun3d",
19                           analysisDir = "FUN3DAnalysisTest",
20                           capsIntent = "CFD")
21
22 inviscid = {"bcType" : "Inviscid", "wallTemperature" : 1.1}
23 fun3d.setAnalysisVal("Boundary_Condition", [("Skin", inviscid),
24                                             ("SymmPlane", "SymmetryY"),
25                                             ("Farfield", "farfield")])
26
27 # Create tree
28 fun3d.createTree(internetAccess = False, embedJSON = True, analysisGeom = True, reverseMap = True)

```

```

29
30
31 # Close our problems
32 print("Closing our problem")
33 myProblem.closeCAPS()
34

```

7.5 analysis6.py

Example use cases for interacting the `pyCAPS._capsAnalysis.getAnalysisInfo()` function.

```

1 from __future__ import print_function
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
11 # project name "basicTest" may be optionally set here; if no argument is provided
12 # the CAPS file provided is used as the project name.
13 print("Loading file into our capsProblem")
14 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
15
16 # Load Tetgen aim
17 tetgen = myProblem.loadAIM(aim = "tetgenAIM",
18                             altName = "tetgen",
19                             analysisDir = "TetGenAnalysisTest",
20                             capsIntent = "CFD")
21
22 # Analysis information
23 state = tetgen.getAnalysisInfo()
24 print("Cleanliness state", state) # Cleanliness status
25
26 # Get analysis information dictionary
27 infoDict = tetgen.getAnalysisInfo(printinfo = False, infoDict = True)
28
29 # Print dictionary
30 print(infoDict)
31
32 # Close our problems
33 print("Closing our problem")
34 myProblem.closeCAPS()
35

```

7.6 analysis7.py

Example use cases for interacting the `pyCAPS._capsAnalysis.addAttribute()` and `pyCAPS._capsAnalysis.getAttribute()` functions.

```

1 from __future__ import print_function
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
11 # project name "basicTest" may be optionally set here; if no argument is provided
12 # the CAPS file provided is used as the project name.
13 print("Loading file into our capsProblem")
14 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
15
16 # Load Tetgen aim
17 tetgen = myProblem.loadAIM(aim = "tetgenAIM",
18                             altName = "tetgen",
19                             analysisDir = "TetGenAnalysisTest",
20                             capsIntent = "CFD")
21
22 # Add attribute
23 tetgen.addAttribute("testAttr", [1, 2, 3])
24
25 # Add another attribute
26 tetgen.addAttribute("testAttr_2", "anotherAttribute")

```

```

27
28 myValue = tetgen.getAttribute("testAttr")
29 print("Value = ", myValue)
30
31 myValue = tetgen.getAttribute("testAttr_2")
32 print("Value = ", myValue)
33
34 # Close our problems
35 print("Closing our problem")
36 myProblem.closeCAPS()
37

```

7.7 exception.py

Example of raised error ([pyCAPS.CAPSError](#)) handling in [pyCAPS](#).

```

1 # Use: Test autolinking of capsValue objects
2
3 from __future__ import print_function
4
5 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
6 import pyCAPS
7
8 # Instantiate our CAPS problem "myProblem"
9 print("Initiating capsProblem")
10 myProblem = pyCAPS.capsProblem()
11
12 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem.
13 print("Loading file into our capsProblem")
14 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm")
15
16 # Try to load another geometry into the problem - this is forbidden
17 try:
18     myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm")
19
20 except pyCAPS.CAPSError as e:
21     print("We caught the error!")
22
23     print("Error code =", e.errorCode)
24     print("Error name =", e.errorName)
25
26     # Do something based on error captured.
27
28 except:
29     print ("We did not catch the error!")
30

```

7.8 problem.py

Basic example use case of the initiation ([pyCAPS.capsProblem.__init__](#)) and termination ([pyCAPS.capsProblem.closeCAPS](#))

```

1 # Use: Initiate/close problem.
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Close our problem
11 print("Closing our problem")
12 myProblem.closeCAPS()

```

7.9 problem1.py

Example use case for the [pyCAPS.capsProblem.loadCAPS\(\)](#) function in which are multiple problems with geometry are created.

```

1 # Use: Create a multiple problems with geometry
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)

```

```

4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblems")
8 myProblem = pyCAPS.capsProblem()
9
10 # Create another problem
11 myProblemNew = pyCAPS.capsProblem()
12
13 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
14 # project name "basicTest" may be optionally set here; if no argument is provided
15 # the CAPS file provided is used as the project name.
16 print("Loading file into our capsProblems")
17
18
19 myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
20
21
22 myProblemNew.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
23
24 # Close our problems
25 print("Closing our problems")
26 myProblem.closeCAPS()
27 myProblemNew.closeCAPS()
28

```

7.10 problem2.py

Example use case for the `pyCAPS.capsProblem.loadCAPS()` function - compare "geometry" attribute with returned geometry object.

```

1 # Use: Load geometry into the problem - compare geometry with returned geometry object
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
11 # project name "basicTest" may be optionally set here; if no argument is provided
12 # the CAPS file provided is used as the project name.
13 print("Loading file into our capsProblem")
14
15
16 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
17
18
19 print(myGeometry)
20 print(myProblem.geometry)
21
22 # Close our problems
23 print("Closing our problem")
24 myProblem.closeCAPS()
25

```

7.11 problem3.py

Example use cases for the `pyCAPS.capsProblem.loadAIM()` function.

```

1 # Use: Load a analysis module (AIM) into the problem
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
11 # project name "basicTest" may be optionally set here; if no argument is provided
12 # the CAPS file provided is used as the project name.
13 print("Loading file into our capsProblem")
14 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
15
16 # Load FUN3D aim
17

```

```

18 myProblem.loadAIM(aim = "fun3dAIM",
19                   altName = "fun3d",
20                   analysisDir = "FUN3DAnalysisTest",
21                   capsIntent = "CFD")
22
23
24
25 print(myProblem.analysis["fun3d"].analysisDir)
26
27
28 # Close our problems
29 print("Closing our problem")
30 myProblem.closeCAPS()
31

```

7.12 problem4.py

Example use cases for the `pyCAPS.capsProblem.loadAIM()` function - compare analysis dictionary entry with returned analysis object.

```

1 # Use: Load a analysis module (AIM) into the problem - compare analysis dictionary with returned analysis
  object
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
11 # project name "basicTest" may be optionally set here; if no argument is provided
12 # the CAPS file provided is used as the project name.
13 print("Loading file into our capsProblem")
14 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
15
16 # Load FUN3D aim
17
18 fun3d = myProblem.loadAIM(aim = "fun3dAIM",
19                           altName = "fun3d",
20                           analysisDir = "FUN3DAnalysisTest",
21                           capsIntent = "CFD")
22
23 print(fun3d.analysisDir)
24
25
26 print(fun3d)
27 print(myProblem.analysis["fun3d"])
28
29 # Close our problems
30 print("Closing our problem")
31 myProblem.closeCAPS()
32

```

7.13 problem5.py

Basic example for setting the verbosity of a problem using `pyCAPS.capsProblem.setVerbosity()` function.

```

1 #Use case set verbosity of the problem
2 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
3 import pyCAPS
4
5 # Instantiate our CAPS problem "myProblem"
6 print("Initiating capsProblem")
7 myProblem = pyCAPS.capsProblem()
8
9 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
10 # project name "basicTest" may be optionally set here; if no argument is provided
11 # the CAPS file provided is used as the project name.
12 print("Loading file into our capsProblem")
13 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest", verbosity="debug")
14
15
16 # Change verbosity to minimal - 0 (integer value)
17 myProblem.setVerbosity("minimal")
18
19 # Change verbosity to standard - 1 (integer value)
20 myProblem.setVerbosity("standard")

```



```

21
22 # Change verbosity to back to minimal using integer value - 0
23 myProblem.setVerbosity(0)
24
25 # Change verbosity to back to debug using integer value - 2
26 myProblem.setVerbosity(2)
27
28 # Give wrong value
29 myProblem.setVerbosity(10)
30
31
32 # Close our problems
33 print("Closing our problem")
34 myProblem.closeCAPS()
35

```

7.14 problem6.py

Example use case for the `pyCAPS.capsProblem.createTree()` function.

```

1 # Use: Check creating a tree on the problem
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 myProblem = pyCAPS.capsProblem()
8
9 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem.
10 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", verbosity = "debug")
11
12 # Create problem tree
13 myProblem.createTree()
14
15 # Close our problems
16 myProblem.closeCAPS()

```

7.15 problem7.py

Example use cases for interacting the `pyCAPS.capsProblem.addAttribute()` and `pyCAPS.capsProblem.getAttribute()` functions.

```

1 # Use: Check setting and getting an Attribute on a problem object
2
3 from __future__ import print_function
4
5 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
6 import pyCAPS
7
8 # Instantiate our CAPS problem "myProblem"
9 myProblem = pyCAPS.capsProblem()
10
11 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem.
12 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", verbosity = "debug")
13
14 # Add attribute
15 myProblem.addAttribute("testAttr", [1, 2, 3])
16
17 # Add another attribute
18 myProblem.addAttribute("testAttr_2", "anotherAttribute")
19
20 myValue = myProblem.getAttribute("testAttr")
21 print("Value = ", myValue)
22
23 myValue = myProblem.getAttribute("testAttr_2")
24 print("Value = ", myValue)
25
26 # Close our problems
27 myProblem.closeCAPS()

```

7.16 problem8.py

Example use case for the `pyCAPS.capsProblem.loadCAPS()` and `pyCAPS.capsProblem.saveCAPS()` functions - using a CAPS file to initiate a new problem.

```

1 # Use: Load a analysis module (AIM) into the problem, save the problem, then use that CAPS file
2 # to initiate a new problem
3
4 from __future__ import print_function
5
6 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
7 import pyCAPS
8
9 # Instantiate our CAPS problem "myProblem"
10 print("Initiating capsProblem")
11 myProblem = pyCAPS.capsProblem()
12
13 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
14 # project name "basicTest" may be optionally set here; if no argument is provided
15 # the CAPS file provided is used as the project name.
16 print("Loading file into our capsProblem")
17 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
18
19 # Load FUN3D aim
20 myAnalysis = myProblem.loadAIM(aim = "fun3dAIM",
21                               altName = "fun3d",
22                               analysisDir = "FUN3DAnalysisTest",
23                               capsIntent = "CFD")
24
25 # Run pre and post to get the analysis in a "clean" state
26 myAnalysis.preAnalysis()
27 myAnalysis.postAnalysis()
28
29 # Add a value/parameter to problem
30 myValye = myProblem.createValue("Mach", 0.5)
31
32 # Save our problem - use default name
33 print("Saving out problem")
34 myProblem.saveCAPS()
35
36 # Close our problem
37 print("Closing our problem")
38 myProblem.closeCAPS()
39
40 #Create a new problem
41 myProblem = pyCAPS.capsProblem()
42
43 # Reload the saved problem
44 print("Reload the saved problem")
45 myGeometry = myProblem.loadCAPS("saveCAPS.caps")
46
47 # Get a copy of the analysis object
48 myAnalysis = myProblem.analysis["fun3d"]
49
50 # Get analysis
51 myAnalysis.getAnalysisInfo()
52
53 # Check other features of analysis object
54 print(myAnalysis.aimName)
55 print(myAnalysis.analysisDir)
56
57 # Get a copy of the value object
58 myValue = myProblem.value["Mach"]
59 print("Value =", myValue.value)
60

```

7.17 units.py

Example use cases for `pyCAPS.capsConvert()` function.

```

1 # Make print statement Python 2-3 agnostic
2 from __future__ import print_function
3
4 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
5 from pyCAPS import capsConvert
6
7 # Convert mile to feet
8 value = capsConvert(19, "mile", "ft")
9 print(value, "ft")
10
11 # Convert gallon to milliliter
12 value = capsConvert(5.0, "gallon", "ml")
13 print(value, "ml")
14
15 # Convert feet to kilograms - expect failure as this doesn't make sense
16 try:
17     value = capsConvert(1, "ft", "kg")
18 except:

```

```

19     print("Error occurred")
20
21 # Convert pounds to kilograms
22 value = capsConvert(10.0, "lb", "kg")
23 print(value, "kg")
24
25 # Convert slug to kilogram
26 value = capsConvert([1.0, 2.0, 4.0, 10], "slug", "kg")
27 print(value, "kg")
28
29 # Convert BTU (British Thermal Unit) to Joules
30 value = capsConvert([[1.0, 2.0], [4.0, 10]], "btu", "J")
31 print(value, "J")
32
33 # Convert foot per second to meter per hour
34 value = capsConvert(1, "foot per second", "m/h")
35 print(value, "m/h")
36
37 # Convert foot per second to meter per hour - scale ft/s by 1/6
38 value = capsConvert(1, "(5 foot)/(30 second)", "m/h")
39 print(value, "m/h")

```

7.18 value.py

Example use cases for `pyCAPS.capsProblem.createValue()` function.

```

1 from __future__ import print_function
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem.
11 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
12
13 # Create a value
14 myValue = myProblem.createValue("Altitude", [0.0, 30000.0, 60000.0], units="ft", limits=[0.0, 70000])
15
16 print("Name = ", myValue.name)
17 print("Value = ", myValue.value)
18 print("Units = ", myValue.units)
19 print("Limits = ", myValue.limits)
20
21 # Close our problems
22 print("Closing our problem")
23 myProblem.closeCAPS()

```

7.19 value2.py

Example use cases for interacting the a `pyCAPS._capsValue` object for setting the value.

```

1 from __future__ import print_function
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem.
11 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
12
13 # Create a value
14 myValue = myProblem.createValue("Altitude", [0.0, 30000.0, 60000.0], units="ft", limits=[0.0, 70000])
15
16 print("Name = ", myValue.name)
17 print("Value = ", myValue.value)
18 print("Units = ", myValue.units)
19 print("Limits = ", myValue.limits)
20
21 # Change the value
22 myValue.value = [0.0, 40000.0, 50000]
23 print("New Value = ", myValue.value)
24
25 # Or change the value using the setVal() function - equivalent to the directly setting the value

```

```

26 myValue.setVal([0.0, 45000.0, 55000])
27 print("New Value = ", myValue.value)
28
29 # Get the value using the getVal() function - equivalent to directly calling the value
30 print("Current Value = ", myValue.getVal())
31
32 # Close our problems
33 print("Closing our problem")
34 myProblem.closeCAPS()
35

```

7.20 value3.py

Example use cases for interacting the a [pyCAPS._capsValue](#) object for setting the limits.

```

1 from __future__ import print_function
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem.
11 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
12
13 # Create a value
14 myValue = myProblem.createValue("Altitude", [0.0, 30000.0, 60000.0], units="ft", limits=[0.0, 70000])
15
16 print("Name = ", myValue.name)
17 print("Value = ", myValue.value)
18 print("Units = ", myValue.units)
19 print("Limits = ", myValue.limits)
20
21 # Change the limits
22 myValue.limits = [0.0, 65000]
23 print("New Limits = ", myValue.limits)
24
25 # or change the limits using the setLimits() function - equivalent to the directly setting the limits
26 myValue.setLimits([0.0, 69000])
27 print("New Limits = ", myValue.limits)
28
29 # Get the limits using the getLimits() function - equivalent to directly calling the limits
30 print("Current Limits = ", myValue.getLimits())
31
32 # Close our problems
33 print("Closing our problem")
34 myProblem.closeCAPS()
35

```

7.21 value4.py

Example use cases for interacting the with [pyCAPS._capsValue.convertUnits\(\)](#) function.

```

1 from __future__ import print_function
2
3 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
4 import pyCAPS
5
6 # Instantiate our CAPS problem "myProblem"
7 print("Initiating capsProblem")
8 myProblem = pyCAPS.capsProblem()
9
10 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem.
11 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
12
13 # Create a value
14 myValue = myProblem.createValue("Altitude", [0.0, 30000.0, 60000.0], units="ft")
15 print("Name = ", myValue.name)
16 print("Value = ", myValue.value)
17 print("Units = ", myValue.units)
18
19 # Convert the units from feet to meters
20 convertValue = myValue.convertUnits("m")
21 print("Converted Value = ", convertValue, "(m)")
22
23 # Create a new value
24 myValue = myProblem.createValue("Freq", [[0.0, 1, 2], [.25, .5, .75]], units="Hz")

```

```

25 print("Name = ", myValue.name)
26 print("Value = ", myValue.value)
27 print("Units = ", myValue.units)
28
29 # Convert the units from 1/s to 1/min
30 convertValue = myValue.convertUnits("1 per minute")
31 print("Converted Value = ", convertValue, "(1/min)")
32
33 # Create a new value
34 myValue = myProblem.createValue("Energy", 1.0, units="Btu")
35 print("Name = ", myValue.name)
36 print("Value = ", myValue.value)
37 print("Units = ", myValue.units)
38
39 # Convert the units from Btu to Joules
40 convertValue = myValue.convertUnits("J")
41 print("Converted Value = ", convertValue, "(J)")
42
43 # Close our problems
44 print("Closing our problem")
45 myProblem.closeCAPS()

```

7.22 value6.py

Example use cases for `pyCAPS.capsProblem.autoLinkValue()` function.

```

1 # Use: Test autolinking of capsValue objects
2
3 from __future__ import print_function
4
5 # Import pyCAPS module (Linux and OSX = pyCAPS.so file; Windows = pyCAPS.pyd file)
6 import pyCAPS
7
8 # Instantiate our CAPS problem "myProblem"
9 print("Initiating capsProblem")
10 myProblem = pyCAPS.capsProblem()
11
12 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem.
13 myGeometry = myProblem.loadCAPS("./csmData/cfdMultiBody.csm", "basicTest")
14
15 myAnalysis = myProblem.loadAIM(aim = "fun3dAIM",
16                               analysisDir = "FUN3DAnalysisTest",
17                               capsIntent = "CFD")
18
19 # Create a value
20 myValue = myProblem.createValue("Alpha", -10.0, units="degree")
21 print("Alpha Value = ", myValue.value)
22
23 # Create another value
24 myValue = myProblem.createValue("Mach", 0.25)
25 print("Mach Value = ", myValue.value)
26
27 # Autolink all capsValues with all AIMs loaded into the problem
28 myProblem.autoLinkValue()
29
30 # Values are updated in a lazy manner - the value shouldn't reflect the linked value yet
31 print("Mach = ", myAnalysis.getAnalysisVal("Mach"))
32 print("Alpha = ", myAnalysis.getAnalysisVal("Alpha"))
33
34 # Create a new value
35 myValue = myProblem.createValue("Beta", 1.3, units="degree")
36 print("Beta Value = ", myValue.value)
37
38 # Autolink just the new value with all AIMs loaded into the problem
39 myProblem.autoLinkValue(myValue) # or could have used - myProblem.autoLinkValue("Beta")
40
41 # Run preAnalysis to force the value update
42 myAnalysis.preAnalysis()
43
44 # Re-print the values - the values should reflect the linked values
45 print("Again... Mach = ", myAnalysis.getAnalysisVal("Mach"))
46 print("Again... Alpha = ", myAnalysis.getAnalysisVal("Alpha"))
47 print("Again... Beta = ", myAnalysis.getAnalysisVal("Beta"))
48
49 # Close our problems
50 print("Closing our problem")
51 myProblem.closeCAPS()

```

7.23 webviewer.py

Basic example use case for creating visualizing data using CAPS's web viewer [pyCAPS.capsViewer](#)

```

1 from pyCAPS import capsViewer
2
3 # Initialize viewer
4 viewer = capsViewer()
5
6 # Define nodes
7 nodes = [ [0, 0, 0],
8           [1, 1, 0],
9           [0, 1, 0],
10          [2, 0, 0],
11          [2, 2, 0],
12          [3, 2, 0],
13          [5, 0, 0],
14          [7, 3, 0]]
15
16 # Define element connectivity
17 connectivity = [ [1, 2, 3],
18                 [1, 4, 2],
19                 [4, 5, 2],
20                 [6, 5, 4],
21                 [4, 7, 8, 6]]
22
23 # Define scalar data at nodes
24 color = [1, 1, 2, 2, 4, 2, 3, 4]
25
26 # Add nodes
27 viewer.addVertex( nodes)
28
29 # Add element connectivity
30 viewer.addIndex( connectivity)
31
32 # Add color/scalar data values at nodes
33 viewer.addColor( color)
34
35 # Add second set of values
36 color = [0, 30, 40, 50, 60, 90, 10, 100]
37 viewer.addColor( color, name = "Color 2")
38
39 # Add "mesh" connectivity - enables plotting the elemental mesh
40 viewer.addLineIndex(connectivity)
41
42 # Set mesh color - a single value set a monotone color
43 viewer.addLineColor(1, colorMap = "grey")
44
45 # Take all current "items" that have been added and create a triangle graphic primitive
46 viewer.createTriangle()
47
48 # Start the webserver
49 viewer.startServer()
50
51 # Optionally during each "add" function call we could have kept track of the "items"
52 # being created and explicitly used them to create the triangle graphic primitive.
53 #
54 # For example:
55 #
56 # Setup empty list to hold wvItems
57 itemList = []
58
59 itemList.append(viewer.addVertex( nodes )
60
61 itemList.append(viewer.addIndex( connectivity ))
62
63 itemList.append(viewer.addColor( color ))
64
65 itemList.append(viewer.addLineIndex( connectivity ))
66
67 itemList.append(viewer.addLineColor( 1, "grey" ))
68
69 #viewer.createTriangle(itemList)
70

```


Index

- `__init__`
 - `pyCAPS::CAPSError`, 35
 - `pyCAPS::capsProblem`, 39
 - `pyCAPS::capsViewer`, 47
 - `_capsAnalysis`, 7
 - `_capsBound`, 18
 - `_capsDataSet`, 22
 - `_capsGeometry`, 25
 - `_capsValue`, 31
 - `_capsVertexSet`, 34
- `addAttribute`
 - `pyCAPS::_capsAnalysis`, 9
 - `pyCAPS::capsProblem`, 40
- `addColor`
 - `pyCAPS::capsViewer`, 48
- `addDataBound`
 - `pyCAPS::capsViewer`, 49
- `addDataSet`
 - `pyCAPS::capsViewer`, 49
- `addGeometry`
 - `pyCAPS::capsViewer`, 49
- `addIndex`
 - `pyCAPS::capsViewer`, 50
- `addLineColor`
 - `pyCAPS::capsViewer`, 50
- `addLineIndex`
 - `pyCAPS::capsViewer`, 51
- `addPointColor`
 - `pyCAPS::capsViewer`, 51
- `addPointIndex`
 - `pyCAPS::capsViewer`, 52
- `addTecplot`
 - `pyCAPS::capsViewer`, 52
- `addUnstructMesh`
 - `pyCAPS::capsViewer`, 52
- `addVertex`
 - `pyCAPS::capsViewer`, 53
- `aimBackDoor`
 - `pyCAPS::_capsAnalysis`, 9
- `aimGlobalCount`
 - `pyCAPS::capsProblem`, 45
- `aimName`
 - `pyCAPS::_capsAnalysis`, 17
- `aimPostAnalysis`
 - `pyCAPS::_capsAnalysis`, 9
- `aimPreAnalysis`
 - `pyCAPS::_capsAnalysis`, 9
- `analysis`
 - `pyCAPS::capsProblem`, 45
- `analysisDir`
 - `pyCAPS::_capsAnalysis`, 17
 - `pyCAPS::capsProblem`, 45
- `autoLinkValue`
 - `pyCAPS::capsProblem`, 40
- `backDoor`
 - `pyCAPS::_capsAnalysis`, 9
- `boundName`
 - `pyCAPS::_capsBound`, 21
- `buildGeometry`
 - `pyCAPS::_capsGeometry`, 26
- `CAPSError`, 35
- `capsAnalysis`
 - `pyCAPS::_capsVertexSet`, 34
- `capsBound`
 - `pyCAPS::_capsDataSet`, 24
 - `pyCAPS::_capsVertexSet`, 34
- `capsConvert`
 - `pyCAPS`, 6
- `capsError`
 - `pyCAPS::CAPSError`, 36
- `capsIntent`
 - `pyCAPS::capsProblem`, 45
- `capsProblem`, 38
 - `pyCAPS::_capsAnalysis`, 17
 - `pyCAPS::_capsBound`, 21
 - `pyCAPS::_capsValue`, 33
- `capsVertexSet`
 - `pyCAPS::_capsDataSet`, 25
- `capsViewer`, 46
- `clearItems`
 - `pyCAPS::capsViewer`, 53
- `closeCAPS`
 - `pyCAPS::capsProblem`, 40
- `convertUnits`
 - `pyCAPS::_capsValue`, 32
- `createDataBound`
 - `pyCAPS::capsProblem`, 41
- `createDataTransfer`
 - `pyCAPS::capsProblem`, 41
- `createLine`
 - `pyCAPS::capsViewer`, 53
- `createOpenMDAComponent`
 - `pyCAPS::_capsAnalysis`, 10
- `createPoint`
 - `pyCAPS::capsViewer`, 53
- `createPrimitive`
 - `pyCAPS::capsViewer`, 54
- `createTree`
 - `pyCAPS::_capsAnalysis`, 11
 - `pyCAPS::_capsBound`, 19
 - `pyCAPS::_capsGeometry`, 26
 - `pyCAPS::capsProblem`, 41
- `createTriangle`
 - `pyCAPS::capsViewer`, 54
- `createValue`
 - `pyCAPS::capsProblem`, 42
- `dataBound`
 - `pyCAPS::capsProblem`, 46

- dataRank
 - pyCAPS::_capsDataSet, 25
- dataSetDest
 - pyCAPS::_capsBound, 21
- dataSetMethod
 - pyCAPS::_capsDataSet, 25
- dataSetName
 - pyCAPS::_capsDataSet, 25
- dataSetSrc
 - pyCAPS::_capsBound, 21
- dirtyAnalysis
 - pyCAPS::capsProblem, 43
- egadsError
 - pyCAPS::CAPSError, 36
- executeTransfer
 - pyCAPS::_capsBound, 19
- fillVertexSets
 - pyCAPS::_capsBound, 20
- geomName
 - pyCAPS::_capsGeometry, 31
- geometry
 - pyCAPS::capsProblem, 46
- getAnalysisInfo
 - pyCAPS::_capsAnalysis, 12
- getAnalysisOutVal
 - pyCAPS::_capsAnalysis, 12
- getAnalysisVal
 - pyCAPS::_capsAnalysis, 13
- getAttribute
 - pyCAPS::_capsAnalysis, 13
 - pyCAPS::capsProblem, 43
- getAttributeMap
 - pyCAPS::_capsAnalysis, 13
 - pyCAPS::_capsGeometry, 26
- getAttributeVal
 - pyCAPS::_capsAnalysis, 15
 - pyCAPS::_capsGeometry, 28
- getBoundInfo
 - pyCAPS::_capsBound, 20
- getData
 - pyCAPS::_capsDataSet, 23
- getDataConnect
 - pyCAPS::_capsDataSet, 23
- getDataXYZ
 - pyCAPS::_capsDataSet, 23
- getGeometryOutVal
 - pyCAPS::_capsGeometry, 29
- getGeometryVal
 - pyCAPS::_capsGeometry, 29
- getLimits
 - pyCAPS::_capsValue, 32
- getSensitivity
 - pyCAPS::_capsAnalysis, 15
- getVal
 - pyCAPS::_capsValue, 33
- initDataSet
 - pyCAPS::_capsDataSet, 23
- limits
 - pyCAPS::_capsValue, 34
- loadAIM
 - pyCAPS::capsProblem, 43
- loadCAPS
 - pyCAPS::capsProblem, 44
- numTess
 - pyCAPS::capsViewer, 55
- ocsmError
 - pyCAPS::CAPSError, 37
- officialName
 - pyCAPS::_capsAnalysis, 18
- openMDAOCComponent
 - pyCAPS::_capsAnalysis, 18
- parents
 - pyCAPS::_capsAnalysis, 18
- postAnalysis
 - pyCAPS::_capsAnalysis, 16
- preAnalysis
 - pyCAPS::_capsAnalysis, 16
- pyCAPS::_capsAnalysis
 - addAttribute, 9
 - aimBackDoor, 9
 - aimName, 17
 - aimPostAnalysis, 9
 - aimPreAnalysis, 9
 - analysisDir, 17
 - backDoor, 9
 - capsProblem, 17
 - createOpenMDAOCComponent, 10
 - createTree, 11
 - getAnalysisInfo, 12
 - getAnalysisOutVal, 12
 - getAnalysisVal, 13
 - getAttribute, 13
 - getAttributeMap, 13
 - getAttributeVal, 15
 - getSensitivity, 15
 - officialName, 18
 - openMDAOCComponent, 18
 - parents, 18
 - postAnalysis, 16
 - preAnalysis, 16
 - saveGeometry, 16
 - setAnalysisVal, 16
 - unitSystem, 18
 - viewGeometry, 17
- pyCAPS::_capsBound
 - boundName, 21
 - capsProblem, 21
 - createTree, 19
 - dataSetDest, 21
 - dataSetSrc, 21

- executeTransfer, 19
- fillVertexSets, 20
- getBoundInfo, 20
- variables, 22
- vertexSet, 22
- viewData, 20
- writeTecplot, 21
- pyCAPS::_capsDataSet
 - capsBound, 24
 - capsVertexSet, 25
 - dataRank, 25
 - dataSetMethod, 25
 - dataSetName, 25
 - getData, 23
 - getDataConnect, 23
 - getDataXYZ, 23
 - initDataSet, 23
 - viewData, 23
 - writeTecplot, 24
- pyCAPS::_capsGeometry
 - buildGeometry, 26
 - createTree, 26
 - geomName, 31
 - getAttributeMap, 26
 - getAttributeVal, 28
 - getGeometryOutVal, 29
 - getGeometryVal, 29
 - saveGeometry, 30
 - setGeometryVal, 30
 - viewGeometry, 30
- pyCAPS::_capsValue
 - capsProblem, 33
 - convertUnits, 32
 - getLimits, 32
 - getVal, 33
 - limits, 34
 - setLimits, 33
 - setVal, 33
 - value, 34
- pyCAPS::_capsVertexSet
 - capsAnalysis, 34
 - capsBound, 34
 - vertexSetName, 34
- pyCAPS::CAPSError
 - __init__, 35
 - capsError, 36
 - egadsError, 36
 - ocsmError, 37
- pyCAPS::capsProblem
 - __init__, 39
 - addAttribute, 40
 - aimGlobalCount, 45
 - analysis, 45
 - analysisDir, 45
 - autoLinkValue, 40
 - capsIntent, 45
 - closeCAPS, 40
 - createDataBound, 41
 - createDataTransfer, 41
 - createTree, 41
 - createValue, 42
 - dataBound, 46
 - dirtyAnalysis, 43
 - geometry, 46
 - getAttribute, 43
 - loadAIM, 43
 - loadCAPS, 44
 - raiseException, 46
 - saveCAPS, 44
 - setVerbosity, 45
 - value, 46
- pyCAPS::capsViewer
 - __init__, 47
 - addColor, 48
 - addDataBound, 49
 - addDataSet, 49
 - addGeometry, 49
 - addIndex, 50
 - addLineColor, 50
 - addLineIndex, 51
 - addPointColor, 51
 - addPointIndex, 52
 - addTecplot, 52
 - addUnstructMesh, 52
 - addVertex, 53
 - clearItems, 53
 - createLine, 53
 - createPoint, 53
 - createPrimitive, 54
 - createTriangle, 54
 - numTess, 55
 - startServer, 55
- pyCAPS, 5
 - capsConvert, 6
- raiseException
 - pyCAPS::capsProblem, 46
- saveCAPS
 - pyCAPS::capsProblem, 44
- saveGeometry
 - pyCAPS::_capsAnalysis, 16
 - pyCAPS::_capsGeometry, 30
- setAnalysisVal
 - pyCAPS::_capsAnalysis, 16
- setGeometryVal
 - pyCAPS::_capsGeometry, 30
- setLimits
 - pyCAPS::_capsValue, 33
- setVal
 - pyCAPS::_capsValue, 33
- setVerbosity
 - pyCAPS::capsProblem, 45
- startServer
 - pyCAPS::capsViewer, 55
- unitSystem

- pyCAPS::_capsAnalysis, [18](#)
- value
 - pyCAPS::_capsValue, [34](#)
 - pyCAPS::capsProblem, [46](#)
- variables
 - pyCAPS::_capsBound, [22](#)
- vertexSet
 - pyCAPS::_capsBound, [22](#)
- vertexSetName
 - pyCAPS::_capsVertexSet, [34](#)
- viewData
 - pyCAPS::_capsBound, [20](#)
 - pyCAPS::_capsDataSet, [23](#)
- viewGeometry
 - pyCAPS::_capsAnalysis, [17](#)
 - pyCAPS::_capsGeometry, [30](#)
- writeTecplot
 - pyCAPS::_capsBound, [21](#)
 - pyCAPS::_capsDataSet, [24](#)