

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

Ryan Durscher and Marshall Galbraith
AFRL/RQVC MIT/ACDL

May 28, 2021

0.1 Introduction	1
0.1.1 Overview	1
0.1.2 Key differences between pyCAPS and CAPS	1
0.1.3 Clearance Statement	1
0.2 Hierarchical Index	1
0.2.1 Class Hierarchy	1
0.3 Class Index	2
0.3.1 Class List	2
0.4 Class Documentation	2
0.4.1 Analysis Class Reference	2
0.4.1.1 Detailed Description	3
0.4.1.2 Member Function Documentation	3
0.4.2 AnalysisGeometry Class Reference	6
0.4.2.1 Detailed Description	7
0.4.2.2 Member Function Documentation	7
0.4.3 AnalysisSequence Class Reference	10
0.4.3.1 Detailed Description	11
0.4.3.2 Member Function Documentation	11
0.4.4 AttrSequence Class Reference	14
0.4.4.1 Detailed Description	14
0.4.4.2 Member Function Documentation	14
0.4.5 Bound Class Reference	15
0.4.5.1 Detailed Description	15
0.4.5.2 Member Function Documentation	16
0.4.6 BoundSequence Class Reference	17
0.4.6.1 Detailed Description	18
0.4.6.2 Member Function Documentation	18
0.4.7 DataSet Class Reference	18
0.4.7.1 Detailed Description	19
0.4.7.2 Member Function Documentation	19
0.4.8 DataSetSequence Class Reference	21
0.4.8.1 Detailed Description	22
0.4.8.2 Member Function Documentation	22
0.4.9 ParamSequence Class Reference	22
0.4.9.1 Detailed Description	23
0.4.9.2 Member Function Documentation	23
0.4.10 Problem Class Reference	24
0.4.10.1 Detailed Description	24
0.4.10.2 Constructor & Destructor Documentation	25
0.4.10.3 Member Function Documentation	25
0.4.11 ProblemGeometry Class Reference	26
0.4.11.1 Detailed Description	27

0.4.11.2 Member Function Documentation	27
0.4.12 Sequence Class Reference	32
0.4.12.1 Detailed Description	33
0.4.13 ValueIn Class Reference	34
0.4.13.1 Detailed Description	34
0.4.13.2 Member Function Documentation	34
0.4.14 ValueInSequence Class Reference	35
0.4.14.1 Detailed Description	35
0.4.15 ValueOut Class Reference	35
0.4.15.1 Detailed Description	36
0.4.15.2 Member Function Documentation	36
0.4.16 ValueOutSequence Class Reference	36
0.4.16.1 Detailed Description	37
0.4.17 VertexSet Class Reference	37
0.4.17.1 Detailed Description	38
0.4.17.2 Member Function Documentation	38
0.4.18 VertexSetSequence Class Reference	38
0.4.18.1 Detailed Description	39
0.4.18.2 Member Function Documentation	39
0.5 Example Documentation	40
0.5.1 problem5.py	40
0.5.2 problem6.py	40
Index	41

0.1 Introduction

0.1.1 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 `gettingStarted`.

0.1.2 Key differences between pyCAPS and CAPS

- Manipulating the "owner" information for CAPS objects isn't currently supported

0.1.3 Clearance Statement

0.2 Hierarchical Index

0.2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Analysis	2
AnalysisGeometry	6
Bound	15
DataSet	18
Problem	24
ProblemGeometry	26
Sequence	32
AnalysisSequence	10
AttrSequence	14
BoundSequence	17
DataSetSequence	21
ParamSequence	22
ValueInSequence	35
ValueOutSequence	36
VertexSetSequence	38
ValueIn	34
ValueOut	35
VertexSet	37

0.3 Class Index

0.3.1 Class List

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

Analysis	Defines a CAPS Analysis Object	2
AnalysisGeometry	Defines Analysis Geometry Object	6
AnalysisSequence	Defines a Sequence of CAPS Analysis Objects	10
AttrSequence	Defines a Sequence of CAPS Attribute Value Objects	14
Bound	Defines a CAPS Bound Object	15
BoundSequence	Defines a Sequence of CAPS Bound Objects	17
DataSet	Defines a CAPS DataSet Object	18
DataSetSequence	Defines a Sequence of CAPS DataSet Objects	21
ParamSequence	Defines a Sequence of CAPS Parameter Value Objects	22
Problem	Defines a CAPS Problem Object	24
ProblemGeometry	Defines Problem Geometry Object	26
Sequence	Base class for all CAPS Sequence classes	32
ValueIn	Defines a CAPS input Value Object	34
ValueInSequence	Defines a Sequence of CAPS input Value Objects	35
ValueOut	Defines a CAPS output Value Object Not a standalone class	35
ValueOutSequence	Defines a Sequence of CAPS output Value Objects	36
VertexSet	Defines a CAPS VertexSet Object	37
VertexSetSequence	Defines a Sequence of CAPS Bound Objects	38

0.4 Class Documentation

0.4.1 Analysis Class Reference

Defines a CAPS [Analysis](#) Object.

Inherits object.

Inherited by capsAnalysis.

Public Member Functions

- def [preAnalysis](#) (self)
Run the pre-analysis function for the AIM.
- def [runAnalysis](#) (self)
Run the analysis function for the AIM.
- def [postAnalysis](#) (self)
Run post-analysis function for the AIM.
- def [analysisDir](#) (self)
Property returns the path to the analysis directory.
- def [name](#) (self)
Property returns the name of the CAPS [Analysis](#) Object.
- def [dirty](#) (self)
Returns linked analyses that are dirty.
- def [info](#) (self, printInfo=False, **kwargs)
Gets analysis information for the analysis object.
- def [createTree](#) (self, filename="name", **kwargs)
Create a HTML dendrogram/tree of the current state of the analysis.
- def [createOpenMDAOComponent](#) (self, inputVariable, outputVariable, **kwargs)
Create an OpenMDAO Component[1.7.3]/ExplicitComponent[2.8+] object; an external code component (External↔ Code[1.7.2]/ExternalCodeComp[2.8+]) is created if the executeCommand keyword argument is provided.

0.4.1.1 Detailed Description

Defines a CAPS [Analysis](#) Object.

Created via Problem.analysis.create().

Parameters

<i>Analysis.geometry</i>	AnalysisGeometry instances representing the bodies associated with the analysis
<i>Analysis.input</i>	ValueInSequence of ValueIn inputs
<i>Analysis.output</i>	ValueOutSequence of ValueOut outputs
<i>Analysis.attr</i>	AttrSequence of ValueIn attributes

0.4.1.2 Member Function Documentation

0.4.1.2.1 [createOpenMDAOComponent\(\)](#) `def createOpenMDAOComponent (`
`self,`
`inputVariable,`
`outputVariable,`
`** kwargs)`

Create an OpenMDAO Component[1.7.3]/ExplicitComponent[2.8+] object; an external code component (External↔ Code[1.7.2]/ExternalCodeComp[2.8+]) is created if the executeCommand keyword argument is provided.

This functionality should work with either version 1.7.3 or >=2.8 of OpenMDAO.

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>	<p>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_#" were # represents the iteration number (default - False). By default (= False) input and output files will be continuously overwritten. Notes:</p> <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[1.7.2]/ExternalCodeComp[2.8+] object is created; if not provided or set to None a Component[1.7.3]/ExplicitComponent[2.8+] object is created (default - None).
<i>inputFile</i>	Optional list of input file names for OpenMDAO to check the existence of before OpenMDAO executes the "solve_nonlinear"[1.7.3]/"compute"[2.8+] (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"[1.7.3]/"compute"[2.8+] (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[1.7.2]/ExternalCodeComp[2.8+] 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[1.7.2]/ExternalCodeComp[2.8+] component. The use of this depends on the expected AIM execution.

Parameters

<i>setSensitivity</i>	Optional dictionary containing sensitivity/derivative settings/parameters. Currently only Finite difference is supported!. See OpenMDAO documentation for additional details of "deriv_options"(version 1.7) or "declare_partials"(version 2.8). Common values for a finite difference calculation would be <code>setSensitivity['type'] = "fd"</code> (Note in the version 2.8 documentation this variable has been changed to "method" both variations will work when using version 2.8+), <code>setSensitivity['form'] = "forward" or "backward" or "central"</code> , and <code>setSensitivity['step_size'] = 1.0E-6</code> (Note in the version 2.8 documentation this variable has been changed to "step" both variations will work when using version 2.8+).
-----------------------	---

Returns

Returns the reference to the OpenMDAO component object created.

0.4.1.2.2 createTree() `def createTree (`
 self,
 filename = "name",
 ** *kwargs*)

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

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</code> = False).
**kwargs	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 "_AttrName") 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 <code>attrMap</code> for details.

0.4.1.2.3 dirty()

```
def dirty (
    self )
```

Returns linked analyses that are dirty.

Returns

A list of dirty analyses that need to be exeuted before executing this analysis. An empty list is returned if no linked analyses are dirty.

0.4.1.2.4 info()

```
def info (
    self,
    printInfo = False,
    ** kwargs )
```

Gets analysis information for the analysis object.

Parameters

<i>printInfo</i>	Print information to sceen 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)
-----------------	---

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

- pyCAPS/problem.py

0.4.2 AnalysisGeometry Class Reference

Defines [Analysis](#) Geometry Object.

Inherits object.

Public Member Functions

- def `bodies` (self)
Get dict of geometric bodies.
- def `save` (self, filename, directory=os.getcwd(), extension=".egads", writeTess=True)
Save the current geometry used by the AIM to a file.
- def `view` (self, **kwargs)
View the geometry associated with the analysis.
- def `attrList` (self, attributeName, **kwargs)
Retrieve a list of geometric attribute values of a given name ("attributeName") for the bodies loaded into the analysis.
- def `attrMap` (self, getInternal=False, **kwargs)
Create geometric attribution map (embedded dictionaries) for the bodies loaded into the analysis.

0.4.2.1 Detailed Description

Defines `Analysis` Geometry Object.

Parameters

<code>AnalysisGeometry.despmtr</code>	<code>ValueInSequence</code> of <code>ValueIn</code> CSM design parameters
<code>AnalysisGeometry.cfcpmtr</code>	<code>ValueInSequence</code> of <code>ValueIn</code> CSM configuration parameters
<code>AnalysisGeometry.concpmtr</code>	<code>ValueInSequence</code> of <code>ValueIn</code> CSM constant parameters
<code>AnalysisGeometry.outpmtr</code>	<code>ValueOutSequence</code> of <code>ValueOut</code> CSM outputs

0.4.2.2 Member Function Documentation

0.4.2.2.1 attrList() `def attrList (`
`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

<code>attributeName</code>	Name of attribute to retrieve values for.
<code>**kwargs</code>	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

0.4.2.2.2 attrMap() `def attrMap (`
 `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 "_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

0.4.2.2.3 bodies() `def bodies (`
 `self)`

Get dict of geometric bodies.

Returns

Returns a dictionary of the bodies in the [Analysis](#) Object, as well a the capsLength unit. Keys use the body "_name" attribute or "Body_#".

0.4.2.2.4 save()

```
def save (
    self,
    filename,
    directory = os.getcwd(),
    extension = ".egads",
    writeTess = True )
```

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.
<i>writeTess</i>	Write tessellations to the EGADS file (only applies to .egads extension)

0.4.2.2.5 view()

```
def view (
    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).
-------------------	--

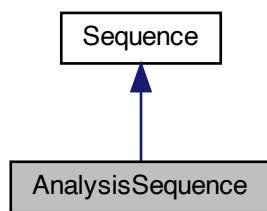
The documentation for this class was generated from the following file:

- pyCAPS/problem.py

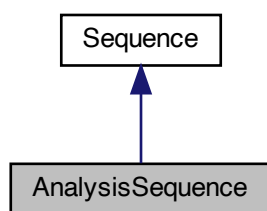
0.4.3 AnalysisSequence Class Reference

Defines a [Sequence](#) of CAPS [Analysis](#) Objects.

Inheritance diagram for AnalysisSequence:



Collaboration diagram for AnalysisSequence:



Public Member Functions

- def `create` (self, aim, name=None, capsIntent=None, unitSystem=None)
Create a CAPS [Analysis](#) Object.
- def `copy` (self, src, name=None)
Create a copy of an CAPS [Analysis](#) Object.
- def `dirty` (self)
Returns analyses that are dirty.

0.4.3.1 Detailed Description

Defines a [Sequence](#) of CAPS [Analysis](#) Objects.

0.4.3.2 Member Function Documentation

0.4.3.2.1 copy() `def copy (`
 `self,`
 `src,`
 `name = None)`

Create a copy of an CAPS [Analysis](#) Object.

Parameters

<i>src</i>	Name of the source Analysis Object or an Analysis Object
<i>name</i>	Name of the new Analysis Object copy

0.4.3.2.2 create() `def create (`
 self,
 aim,
 name = *None*,
 capsIntent = *None*,
 unitSystem = *None*)

Create a CAPS [Analysis](#) Object.

Parameters

<i>aim</i>	Name of the AIM module
<i>name</i>	Name (e.g. key) of the Analysis Object. Must be unique if specified. If <i>None</i> , the default is <code>aim+str(instanceCount)</code> where <code>instanceCount</code> is the count of the existing 'aim' instances.
<i>capsIntent</i>	Analysis intention in which to invoke the AIM.
<i>unitSystem</i>	See AIM documentation for usage.

Returns

The new [Analysis](#) Object is added to the sequence and returned

0.4.3.2.3 dirty() `def dirty (`
 self)

Returns analyses that are dirty.

Returns

A list of dirty analyses. An empty list is returned if no analyses are dirty.

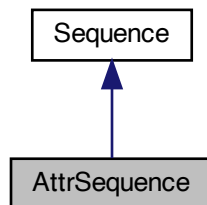
The documentation for this class was generated from the following file:

- `pyCAPS/problem.py`

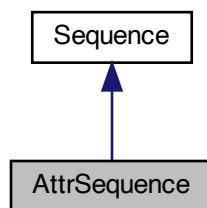
0.4.4 AttrSequence Class Reference

Defines a [Sequence](#) of CAPS Attribute Value Objects.

Inheritance diagram for AttrSequence:



Collaboration diagram for AttrSequence:



Public Member Functions

- def [create](#) (self, name, data, overwrite=False)
Create an attribute (that is meta-data) to the CAPS Object.

0.4.4.1 Detailed Description

Defines a [Sequence](#) of CAPS Attribute Value Objects.

0.4.4.2 Member Function Documentation

0.4.4.2.1 create() `def create (`
 `self,`
 `name,`
 `data,`
 `overwrite = False)`

Create an attribute (that is meta-data) to the CAPS Object.

See example

Parameters

<i>name</i>	Name used to define the attribute.
<i>data</i>	Initial data value(s) for the attribute. Note that type casting is done automatically based on the determined type of the Python object.
<i>overwrite</i>	Flag to overwrite any existing attribute with the same 'name'

Returns

The new Value Object is added to the sequence and returned

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

- pyCAPS/problem.py

0.4.5 Bound Class Reference

Defines a CAPS [Bound](#) Object.

Inherits object.

Inherited by capsBound.

Public Member Functions

- `def name (self)`
Property returns the name of the CAPS [Bound](#) Object.
- `def close (self)`
Closes the bound indicating it's complete.
- `def info (self, printInfo=False, **kwargs)`
Gets information for the bound object.
- `def createTree (self, filename="boundName", **kwargs)`
Create a HTML dendrogram/tree of the current state of the bound.

0.4.5.1 Detailed Description

Defines a CAPS [Bound](#) Object.

Created via Problem.bound.create().

Parameters

<i>Bound.vertexSet</i>	VertexSetSequence of VertexSet instances
<i>Bound.attr</i>	AttrSequence of ValueIn attributes

0.4.5.2 Member Function Documentation

0.4.5.2.1 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

<i>filename</i>	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> = <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.

0.4.5.2.2 info() `def info (`
 self,
 printInfo = `False`,
 ** *kwargs*)

Gets information for the bound object.

Parameters

<i>printInfo</i>	Print information to screen if <code>True</code> .
** <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)
-----------------	--

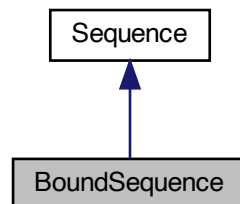
The documentation for this class was generated from the following file:

- pyCAPS/problem.py

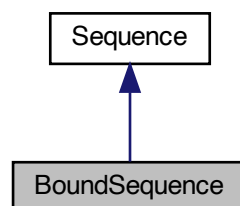
0.4.6 BoundSequence Class Reference

Defines a [Sequence](#) of CAPS [Bound](#) Objects.

Inheritance diagram for BoundSequence:



Collaboration diagram for BoundSequence:



Public Member Functions

- def `create` (self, capsBound, dim=2)
Create a CAPS `Bound` Object.

0.4.6.1 Detailed Description

Defines a `Sequence` of CAPS `Bound` Objects.

0.4.6.2 Member Function Documentation

0.4.6.2.1 `create()` `def create (`
 `self,`
 `capsBound,`
 `dim = 2)`

Create a CAPS `Bound` Object.

Parameters

<code>capsBound</code>	The string value of the capsBound geometry attributes
<code>dim</code>	The dimension of the bound

Returns

The new `Bound` Object is added to the sequence and returned

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

- pyCAPS/problem.py

0.4.7 DataSet Class Reference

Defines a CAPS `DataSet` Object.

Inherits object.

Public Member Functions

- def `name` (self)
Property returns the name of the CAPS `DataSet` Object.
- def `data` (self)
Executes `caps_getData` on data set object to retrieve data set variable.
- def `xyz` (self)

- *Executes caps_getData on data set object to retrieve XYZ coordinates of the data set.*
- `def connectivity (self)`
Executes caps_triangulate on data set's vertex set to retrieve the connectivity (triangles only) information for the data set.
- `def link (self, source, dmethod=caps.dMethod.Interpolate)`
Link this DataSet to an other CAPS DataSet Object.
- `def view (self, fig=None, numDataSet=1, dataSetIndex=0, **kwargs)`
Visualize data set.
- `def writeTecplot (self, filename=None, file=None)`
Write data set to a Tecplot compatible data file.

0.4.7.1 Detailed Description

Defines a CAPS DataSet Object.

Created via VertexSet.dataSet.create().

Parameters

DataSet.attr	AttrSequence of ValueIn attributes
--------------	------------------------------------

0.4.7.2 Member Function Documentation

0.4.7.2.1 connectivity() `def connectivity (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.]

0.4.7.2.2 data() `def data (self)`

Executes caps_getData on data set object to retrieve data set variable.

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.]

0.4.7.2.3 link()

```
def link (
    self,
    source,
    dmethod = caps.dMethod.Interpolate )
```

Link this [DataSet](#) to an other CAPS [DataSet](#) Object.

Parameters

<i>source</i>	The source DataSEt Object
<i>dmethod</i>	Transfer method: dMethod.Interpolate or "Interpolate", tMethod.Conserve or "Conserve"

0.4.7.2.4 view()

```
def view (
    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. '#')).

0.4.7.2.5 writeTecplot()

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

0.4.7.2.6 xyz() `def xyz (`
 `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.])

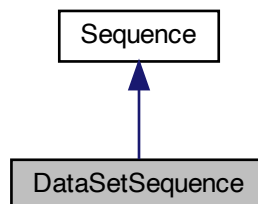
The documentation for this class was generated from the following file:

- `pyCAPS/problem.py`

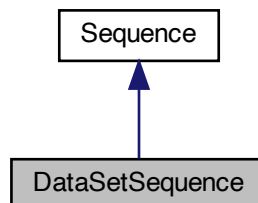
0.4.8 DataSetSequence Class Reference

Defines a [Sequence](#) of CAPS [DataSet](#) Objects.

Inheritance diagram for DataSetSequence:



Collaboration diagram for DataSetSequence:



Public Member Functions

- def `create` (self, dname, ftype, init=None, rank=None)
Create a CAPS [DataSet](#) Object.
- def `fields` (self)
Returns a list of the fields in the [Analysis](#) Object associated with this [DataSet](#).

0.4.8.1 Detailed Description

Defines a [Sequence](#) of CAPS [DataSet](#) Objects.

0.4.8.2 Member Function Documentation

0.4.8.2.1 `create()` `def create (`
 `self,`
 `dname,`
 `ftype,`
 `init = None,`
 `rank = None)`

Create a CAPS [DataSet](#) Object.

Parameters

<i>dname</i>	The name of the data set
<i>ftype</i>	The field type (FieldIn, FieldOut, GeomSens, TessSens, User)
<i>init</i>	Initial value assigned to the DataSet . Length must be consistent with the rank.
<i>rank</i>	The rank of the data set (only needed for un-connected data set)

Returns

The new [DataSet](#) Object is added to the sequence and returned

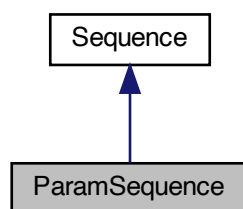
The documentation for this class was generated from the following file:

- pyCAPS/problem.py

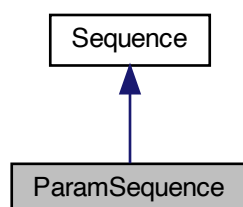
0.4.9 ParamSequence Class Reference

Defines a [Sequence](#) of CAPS Parameter Value Objects.

Inheritance diagram for ParamSequence:



Collaboration diagram for ParamSequence:



Public Member Functions

- def [create](#) (self, name, data, limits=None, fixedLength=True, fixedShape=True)
Create an parameter CAPS Value Object.

0.4.9.1 Detailed Description

Defines a [Sequence](#) of CAPS Parameter Value Objects.

0.4.9.2 Member Function Documentation

0.4.9.2.1 create() `def create (`
 `self,`
 `name,`
 `data,`
 `limits = None,`
 `fixedLength = True,`
 `fixedShape = True)`

Create an parameter CAPS Value Object.

Parameters

<i>name</i>	Name used to define the parameter.
<i>data</i>	Initial data value(s) for the parameter. Note that type casting is done automatically based on the determined type of the Python object.
<i>limits</i>	Limits on the parameter values
<i>fixedLength</i>	Boolean if the value is fixed length
<i>fixedShape</i>	Boolean if the value is fixed shape

Returns

The new Value Object is added to the sequence and returned

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

- pyCAPS/problem.py

0.4.10 Problem Class Reference

Defines a CAPS [Problem](#) Object.

Inherits object.

Inherited by capsProblem.

Public Member Functions

- def `__init__` (self, problemName, phaseName=None, capsFile=None, outLevel=1)
Initialize the problem.
- def `setOutLevel` (self, outLevel)
Set the verbosity level of the CAPS output.
- def `autoLinkParameter` (self, param=None)
*Create a link between a created CAPS parameter and analysis inputs of **all** loaded AIMS, automatically.*
- def `createTree` (self, filename="myProblem", **kwargs)
Create a HTML dendrogram/tree of the current state of the problem.

0.4.10.1 Detailed Description

Defines a CAPS [Problem](#) Object.

The [Problem](#) Object is the top-level object for a single mission/problem. It maintains a single set of interrelated geometric models (see [ProblemGeometry](#)), analyses to be executed (see [Analysis](#)), connectivity and data (see [Bound](#)) associated with the run(s), which can be both multi-fidelity and multi-disciplinary.

Parameters

<i>Problem.geometry</i>	ProblemGeometry instances representing the CSM geometry
<i>Problem.analysis</i>	AnalysisSequence of Analysis instances
<i>Problem.parameter</i>	ParamSequence of ValueIn parameters
<i>Problem.bound</i>	BoundSequence of Bound instances
<i>Problem.attr</i>	AttrSequence of ValueIn attributes

0.4.10.2 Constructor & Destructor Documentation

0.4.10.2.1 `__init__()` `def __init__ (`
 `self,`
 `problemName,`
 `phaseName = None,`
 `capsFile = None,`
 `outLevel = 1)`

Initialize the problem.

Parameters

<i>problemName</i>	CAPS problem name that serves as the root directory for all file I/O.
<i>phaseName</i>	the current phase name (None is equivalent to 'Initial')
<i>capsFile</i>	CAPS file to load. Options: *.csm or *.egads.
<i>outLevel</i>	Level of output verbosity. See setOutLevel .

0.4.10.3 Member Function Documentation

0.4.10.3.1 `autoLinkParameter()` `def autoLinkParameter (`
 `self,`
 `param = None)`

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

Valid CAPS value, parameter objects must be created with `Problem.parameter.create()`. Note, only links to ANALY-SISIN inputs are currently made at this time.

Parameters

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

0.4.10.3.2 `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 = 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>).

0.4.10.3.3 setOutLevel()

```
def setOutLevel (
    self,
    outLevel )
```

Set the verbosity level of the CAPS output.

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

Parameters

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

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

- `pyCAPS/problem.py`

0.4.11 ProblemGeometry Class Reference

Defines [Problem](#) Geometry Object.

Inherits object.

Inherited by `capsGeometry`.

Public Member Functions

- def `build` (self)
Explicitly build geometry.
- def `save` (self, filename="myGeometry", directory=os.getcwd(), extension=".egads")
Save the current geometry to a file.
- def `view` (self, **kwargs)
View or take a screen shot of the geometry configuration.
- def `attrList` (self, attributeName, **kwargs)
Retrieve a list of attribute values of a given name ("attributeName") for the bodies in the current geometry.
- def `attrMap` (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.
- def `bodies` (self)
Get dict of geometric bodies.
- def `lengthUnit` (self)
Get the lenght Unit of geometric bodies.
- def `writeParameters` (self, filename)
Write an OpenCSM Design Parameter file to disk.
- def `readParameters` (self, filename)
Read an OpenCSM Design Parameter file from disk and and overwrites (makes dirty) the current state of the geometry.

0.4.11.1 Detailed Description

Defines `Problem` Geometry Object.

Parameters

<code>ProblemGeometry.despmtr</code>	<code>ValueInSequence</code> of <code>ValueIn</code> CSM design parameters
<code>ProblemGeometry.cfcpmtr</code>	<code>ValueInSequence</code> of <code>ValueIn</code> CSM configuration parameters
<code>ProblemGeometry.concpmtr</code>	<code>ValueInSequence</code> of <code>ValueIn</code> CSM constant parameters
<code>ProblemGeometry.outpmtr</code>	<code>ValueOutSequence</code> of <code>ValueOut</code> CSM outputs

0.4.11.2 Member Function Documentation

0.4.11.2.1 attrList() `def attrList (`
`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.
<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

```
0.4.11.2.2 attrMap() def attrMap (  
    self,  
    getInternal = False,  
    ** kwargs )
```

Create attribution map (embedded 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

0.4.11.2.3 bodies()

```
def bodies (
    self )
```

Get dict of geometric bodies.

Returns

Returns a dictionary of the bodies and the capsLength unit. Keys use the body "_name" attribute or "Body_#".

0.4.11.2.4 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 attrMap for details.

0.4.11.2.5 lengthUnit()

```
def lengthUnit (
    self )
```

Get the length Unit of geometric bodies.

Returns

Returns the length unit defined by capsLength attribute.

0.4.11.2.6 readParameters() `def readParameters (`
 `self,`
 `filename)`

Read an OpenCSM Design Parameter file from disk and and overwrites (makes dirty) the current state of the geometry.

Parameters

<i>filename</i>	Filename of the OpenCSM Design Parameter file
-----------------	---

0.4.11.2.7 save() `def save (`
 `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.

0.4.11.2.8 view() `def view (`
 `self,`
 `** kwargs)`

View or take a screen shot of the geometry configuration.

The use of this function to save geometry 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

<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 \$filename is not None, the viewer is changed to matplotlib.
<i>portNumber</i>	Port number to start the server listening on (default - 7681).
<i>title</i>	Title to add to each figure (default - None).
<i>filename</i>	Save image(s) to file specified (default - None). 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 - False).
<i>ignoreBndBox</i>	Ignore the largest body (default - False).
<i>showImage</i>	Show image(s) (default - False).
<i>showAxes</i>	Show the xyz axes in the image(s) (default - False).
<i>showTess</i>	Show the edges of the tessellation (default - False).
<i>dpi</i>	Resolution in dots-per-inch for the figure (default - None).
<i>tessParam</i>	Custom tessellation paremeters, see EGADS documentation for makeTessBody function. values will be scaled by the norm of the bounding box for the body (default - [0.0250, 0.0010, 15.0]).

0.4.11.2.9 writeParameters() `def writeParameters (`
 self,
 filename)

Write an OpenCSM Design Parameter file to disk.

Parameters

<i>filename</i>	Filename of the OpenCSM Design Parameter file
-----------------	---

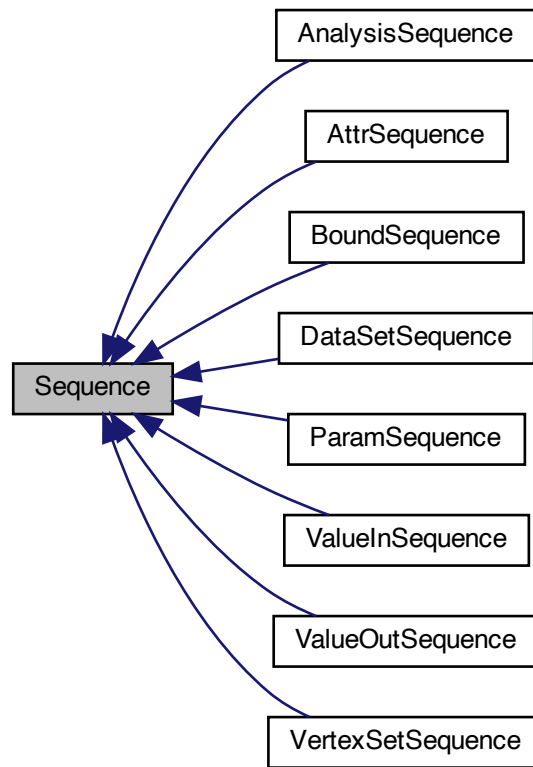
The documentation for this class was generated from the following file:

- pyCAPS/problem.py

0.4.12 Sequence Class Reference

Base class for all CAPS [Sequence](#) classes.

Inheritance diagram for Sequence:



Public Member Functions

- `def keys (self)`
Returns the keys of the [Sequence](#).
- `def values (self)`
Returns the values of the [Sequence](#).
- `def items (self)`
Returns the items of the [Sequence](#).

0.4.12.1 Detailed Description

Base class for all CAPS [Sequence](#) classes.

A CAPS [Sequence](#) only contains instances of a single type. Items are added to the Sequence via the 'create' method in derived classes. Items cannot be removed from the sequence (except for CAPS Attributes).

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

- `pyCAPS/problem.py`

0.4.13 ValueIn Class Reference

Defines a CAPS input Value Object.

Inherits object.

Public Member Functions

- `def value (self)`
Property getter returns a copy the values stored in the CAPS Value Object.
- `def value (self, val)`
Property setter sets the value in the CAPS Value Object.
- `def limits (self)`
Property getter returns a copy the limits of the CAPS Value Object.
- `def limits (self, limit)`
Property setter sets the limits in the CAPS Value Object (if changable)
- `def name (self)`
Property returns the name of the CAPS Value Object.
- `def link (self, source, tmethod=caps.tMethod.Copy)`
Link this input value to an other CAPS Value Object.
- `def unlink (self)`
Remove an existing link.

0.4.13.1 Detailed Description

Defines a CAPS input Value Object.

0.4.13.2 Member Function Documentation

0.4.13.2.1 link() `def link (`
`self,`
`source,`
`tmethod = caps.tMethod.Copy)`

Link this input value to an other CAPS Value Object.

Parameters

<i>source</i>	The source Value Object
<i>tmethod</i>	Transfer method: tMethod.Copy or "Copy", tMethod.Integrate or "Integrate", tMethod.Average or "Average"

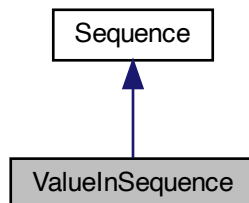
The documentation for this class was generated from the following file:

- pyCAPS/problem.py

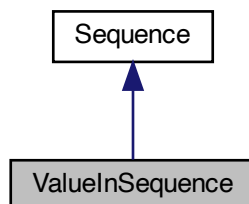
0.4.14 ValueInSequence Class Reference

Defines a [Sequence](#) of CAPS input Value Objects.

Inheritance diagram for ValueInSequence:



Collaboration diagram for ValueInSequence:



Additional Inherited Members

0.4.14.1 Detailed Description

Defines a [Sequence](#) of CAPS input Value Objects.

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

- pyCAPS/problem.py

0.4.15 ValueOut Class Reference

Defines a CAPS output Value Object Not a standalone class.

Inherits object.

Public Member Functions

- def [value](#) (self)
Property getter returns a copy the values stored in the CAPS Value Object.
- def [name](#) (self)
Property returns the name of the CAPS Value Object.
- def [props](#) (self)
Property getter returns a copy the values stored in the CAPS Value Object.
- def [hasDeriv](#) (self)
Returns a string list of of the input Value Object names that can be used in [deriv](#).
- def [deriv](#) (self, [name](#)=None)
Returns derivatives of the output Value Object.

0.4.15.1 Detailed Description

Defines a CAPS output Value Object Not a standalone class.

0.4.15.2 Member Function Documentation

0.4.15.2.1 [deriv\(\)](#)

```
def deriv (
    self,
    name = None )
```

Returns derivatives of the output Value Object.

Parameters

<i>name</i>	Name of the input Value Object to take derivative w.r.t. if name is None then a dictionary with all derivatives from hasDeriv are returned
-------------	--

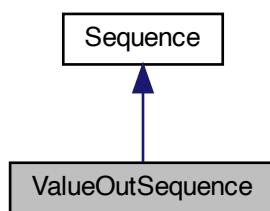
The documentation for this class was generated from the following file:

- pyCAPS/problem.py

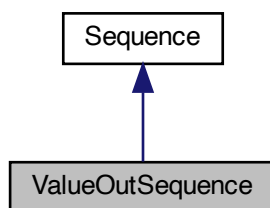
0.4.16 ValueOutSequence Class Reference

Defines a [Sequence](#) of CAPS output Value Objects.

Inheritance diagram for ValueOutSequence:



Collaboration diagram for ValueOutSequence:



Additional Inherited Members

0.4.16.1 Detailed Description

Defines a [Sequence](#) of CAPS output Value Objects.

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

- pyCAPS/problem.py

0.4.17 VertexSet Class Reference

Defines a CAPS [VertexSet](#) Object.

Inherits object.

Public Member Functions

- `def name (self)`
Property returns the name of the CAPS [VertexSet](#) Object.
- `def getDataConnect (self)`
Executes caps_triangulate on data set's vertex set to retrieve the connectivity (triangles only) information for the data set.

0.4.17.1 Detailed Description

Defines a CAPS [VertexSet](#) Object.

Created via `Bound.vertexSet.create()`.

Parameters

<code>VertexSet.dataSet</code>	DataSetSequence of DataSet instances
<code>VertexSet.attr</code>	AttrSequence of ValueIn attributes

0.4.17.2 Member Function Documentation

0.4.17.2.1 `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.]

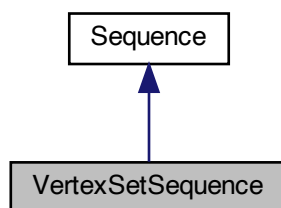
The documentation for this class was generated from the following file:

- pyCAPS/problem.py

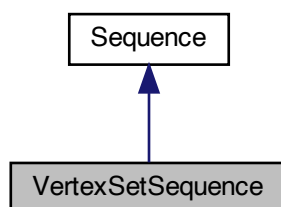
0.4.18 VertexSetSequence Class Reference

Defines a [Sequence](#) of CAPS [Bound](#) Objects.

Inheritance diagram for VertexSetSequence:



Collaboration diagram for VertexSetSequence:



Public Member Functions

- def `create` (self, analysis, vname=None)
Create a CAPS `VertexSet` Object.

0.4.18.1 Detailed Description

Defines a `Sequence` of CAPS `Bound` Objects.

0.4.18.2 Member Function Documentation

0.4.18.2.1 `create()`

```
def create (
    self,
    analysis,
    vname = None )
```

Create a CAPS `VertexSet` Object.

Parameters

<i>analysis</i>	A CAPS Analysis Object or the string name of an Analysis Object instance
<i>vname</i>	Name of the VertexSet (same as the Analysis Object if None)

Returns

The new [VertexSet](#) Object is added to the sequence and returned

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

- pyCAPS/problem.py

0.5 Example Documentation

0.5.1 problem5.py

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

```

1 #Use case set verbosity of the problem
2 import pyCAPS
3
4 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem. The
5 # project name "basicTest" may be optionally set here; if no argument is provided
6 # the CAPS file provided is used as the project name.
7 print("Loading file into our Problem")
8 myProblem = myProblem.Porblem(problemName = "outLevelExample",
9                               capsFile="csmData/cfdMultiBody.csm",
10                              outLevel="debug")
11
12
13 # Change verbosity to minimal - 0 (integer value)
14 myProblem.setOutLevel("minimal")
15
16 # Change verbosity to standard - 1 (integer value)
17 myProblem.setOutLevel("standard")
18
19 # Change verbosity to back to minimal using integer value - 0
20 myProblem.setOutLevel(0)
21
22 # Change verbosity to back to debug using integer value - 2
23 myProblem.setOutLevel(2)
24
25 # Give wrong value (raises and Error)
26 myProblem.setOutLevel(10)

```

0.5.2 problem6.py

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

```

1 # Use: Check creating a tree on the problem
2 import pyCAPS
3
4 # Load a *.csm file "./csmData/cfdMultiBody.csm" into our newly created problem.
5 myProblem = myProblem.Porblem(problemName = "outLevelExample",
6                               capsFile="csmData/cfdMultiBody.csm",
7                               outLevel="debug")
8
9 # Create problem tree
10 myProblem.createTree()

```

Index

- `__init__`
 - Problem, 25
- Analysis, 2
 - `createOpenMDAOCComponent`, 3
 - `createTree`, 5
 - dirty, 5
 - info, 6
- AnalysisGeometry, 6
 - `attrList`, 7
 - `attrMap`, 8
 - bodies, 9
 - save, 9
 - view, 10
- AnalysisSequence, 10
 - copy, 11
 - create, 13
 - dirty, 13
- `attrList`
 - AnalysisGeometry, 7
 - ProblemGeometry, 27
- `attrMap`
 - AnalysisGeometry, 8
 - ProblemGeometry, 28
- AttrSequence, 14
 - create, 14
- `autoLinkParameter`
 - Problem, 25
- bodies
 - AnalysisGeometry, 9
 - ProblemGeometry, 30
- Bound, 15
 - `createTree`, 16
 - info, 16
- BoundSequence, 17
 - create, 18
- connectivity
 - DataSet, 19
- copy
 - AnalysisSequence, 11
- create
 - AnalysisSequence, 13
 - AttrSequence, 14
 - BoundSequence, 18
 - DataSetSequence, 22
 - ParamSequence, 23
 - VertexSetSequence, 39
- `createOpenMDAOCComponent`
 - Analysis, 3
- `createTree`
 - Analysis, 5
 - Bound, 16
 - Problem, 25
 - ProblemGeometry, 30
- data
 - DataSet, 19
- DataSet, 18
 - connectivity, 19
 - data, 19
 - link, 19
 - view, 20
 - `writeTecplot`, 20
 - xyz, 21
- DataSetSequence, 21
 - create, 22
- `deriv`
 - ValueOut, 36
- dirty
 - Analysis, 5
 - AnalysisSequence, 13
- `getDataConnect`
 - VertexSet, 38
- info
 - Analysis, 6
 - Bound, 16
- lengthUnit
 - ProblemGeometry, 30
- link
 - DataSet, 19
 - ValueIn, 34
- ParamSequence, 22
 - create, 23
- Problem, 24
 - `__init__`, 25
 - `autoLinkParameter`, 25
 - `createTree`, 25
 - `setOutLevel`, 26
- ProblemGeometry, 26
 - `attrList`, 27
 - `attrMap`, 28
 - bodies, 30
 - `createTree`, 30
 - lengthUnit, 30
 - `readParameters`, 31

- save, [31](#)
- view, [31](#)
- writeParameters, [32](#)

readParameters

- ProblemGeometry, [31](#)

save

- AnalysisGeometry, [9](#)
- ProblemGeometry, [31](#)

Sequence, [32](#)

setOutLevel

- Problem, [26](#)

ValueIn, [34](#)

- link, [34](#)

ValueInSequence, [35](#)

ValueOut, [35](#)

- deriv, [36](#)

ValueOutSequence, [36](#)

VertexSet, [37](#)

- getDataConnect, [38](#)

VertexSetSequence, [38](#)

- create, [39](#)

view

- AnalysisGeometry, [10](#)
- DataSet, [20](#)
- ProblemGeometry, [31](#)

writeParameters

- ProblemGeometry, [32](#)

writeTecplot

- DataSet, [20](#)

xyz

- DataSet, [21](#)