



# Computational Aircraft Prototype Syntheses

## AIM Programming – AIMs and Units

### For ESP Rev 1.28

Bob Haimes

[bob@geocentrictech.com](mailto:bob@geocentrictech.com) or [haimes@mit.edu](mailto:haimes@mit.edu)

Geocentric Technologies LLC

Any CAPS Value Object can be assigned Units

- It is best to give any AnalysisIn, AnalysisOut or AnalysisDynO the appropriate units that the analysis expects. This makes dealing with the analysis much simpler.
- Any AnalysisIn Value will have already gone through a units conversion when exposed to the AIM.
- Examine the Body's *capsLength* attribute to determine the geometry's units.
- The AIM should either be unit-less as with all previous exercises **or** with all appropriate units defined.

pyCAPS uses the CAPS unit manipulation functions (from `udunits`) to be consistent with internal use of units in CAPS. Similar to the Pint<sup>1</sup> Python package, this defines the following two classes:

- `pyCAPS.Unit`
- `pyCAPS.Quantity`

where `pyCAPS.Unit` defines a unit which can be manipulated with standard operator, and `pyCAPS.Quantity` represents a value with units. This is designed to work with these classes as the C API uses the optional `units` string of the Value Structure. The best way to extract a value from a `pyCAPS.Quantity` is to divide it out by its units.

## Unit Manipulation

```
kg = pyCAPS.Unit("kg")
m  = pyCAPS.Unit("m")
s  = pyCAPS.Unit("s")

Newton = kg*m/s**2
```

## Value from Quantity

```
m  = pyCAPS.Unit("m")
ft = pyCAPS.Unit("ft")

q = 10 * m      # Make a Quantity

assert(10 == q/m)
assert(10 == q.value())
assert(q.convert(ft).value() == q/ft)
```

<sup>1</sup><https://pint.readthedocs.io>

## Path strings

If the Value type is *String* and units member of the Value Structure is set to “PATH” then any slashes are converted automatically based on the OS type currently in use.

## Pointer types and *linkages*

For Value types of either *Pointer* or *PointerMesh* the units member of the Value Structure can be used to insure that there is an appropriate match when linking Values. That is; the unit strings must also match.

Note: using pointer linkages between AIMs should only be done cautiously because the data pointed to always needs to be structurally identical across the AIMs.

# AIM Helper Functions

## Units Handling

- provides useful functions for the AIM programmer
- gives access to CAPS Object data
- note that all function names begin with `aim_`
- if any of these functions are used, then the library must be included (`libaimUtil.a/aimUtil.lib`) in the AIM so/DLL build

## Unit conversion

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

**aimInfo** the AIM context

**count** length of *inValue* and *outValue*

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

**inValue** array of values to be converted

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

**outValue** array of returned converted value (may be same pointer as *inValue*)

**icode** integer return code

## Unit inversion

```
icode = aim_unitInvert(void *aimInfo, const char *inUnits,  
                       char **outUnits)
```

**aimInfo** the AIM context

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

**outUnits** the returned string units = 1/*inUnits* (freeable)

**icode** integer return code

## Unit multiplication

```
icode = aim_unitMultiply(void *aimInfo, const char *inUnits1,  
                        const char *inUnits2, char **outUnits)
```

**aimInfo** the AIM context

**inUnits1** the pointer to the string declaring left units

**inUnits2** the pointer to the string declaring right units

**outUnits** the returned string units = inUnits1\*inUnits2 (freeable)

**icode** integer return code

## Unit division

```
icode = aim_unitDivision(void *aimInfo, const char *inUnits1,  
                        const char *inUnits2, char **outUnits)
```

**aimInfo** the AIM context

**inUnits1** the pointer to the string declaring numerator units

**inUnits2** the pointer to the string declaring denominator units

**outUnits** the returned string units = inUnits1/inUnits2 (freeable)

**icode** integer return code

## Unit raise to a power

```
icode = aim_unitRaise(void *aimInfo, const char *inUnits,  
                      const int power, char **outUnits)
```

**aimInfo** the AIM context

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

**power** power to raise **inUnits**

**outUnits** the returned string units = **inUnits** ^ **power** (freeable)

**icode** integer return code

## Unit raise to root

```
icode = aim_unitRoot(void *aimInfo, const char *inUnits,  
                     const int root, char **outUnits)
```

**aimInfo** the AIM context

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

**root** root to raise **inUnits**

**outUnits** the returned string units = **inUnits** ^ 1/**root** (freeable)

**icode** integer return code



## Unit Offset

```
icode = aim_unitOffset(void *aimInfo, const char *inUnits,  
                       const double offset, char **outUnits)
```

**aimInfo** the AIM context

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

**offset** offset to add to inUnits

**outUnits** the returned string units = inUnits @ offset (freeable)

**icode** integer return code

## Check if two unit strings are convertible

```
icode = aim_unitConvertible(void *aimInfo, const char *unit1,  
                           const char *unit2)
```

**aimInfo** the AIM context

**unit1** string pointer declaring units

**unit2** string pointer declaring units

**icode** integer return code

## Retrieve capsLength length unit attribute from bodies

```
icode = aim_capsLength(void *aimInfo, const char **lengthUnit)
```

**aimInfo** the AIM context

**lengthUnit** the returned string length unit of the bodies

**icode** integer return code

## Get Unit System

```
icode = aim_unitSys(void *aimInfo, char **unitSys)
```

**aimInfo** the AIM context

**unitSys** a returned pointer to a character string declaring the unit system – can be **NULL**

**icode** integer return code

Note: This is a string set at Problem initialization and was supposed to set the unit system in use (i.e., *SI*, *US* or the like) for the problem at-hand. This standardization never took hold, but this string got used for other unit information (so it could not be easily removed). Best to ignore!

In *exercises/session11* modify `theAIM.c` to work with `session11.py`:

- Mass should be in **kg** and length should be in **m**
- Add another input (`wallThickness` – in **m**), so that all densities can be in **kg/m<sup>3</sup>**
- Add unit definitions for the variables set by `aimInputs` and `aimOutputs`
- Allow for the scaling of the geometry (when the triangulation is written)
- Once functioning, explore changing units in the Python script