AML 5.0B5 Reference Manual

AML 5.0B5 Reference Manual

Table of Contents

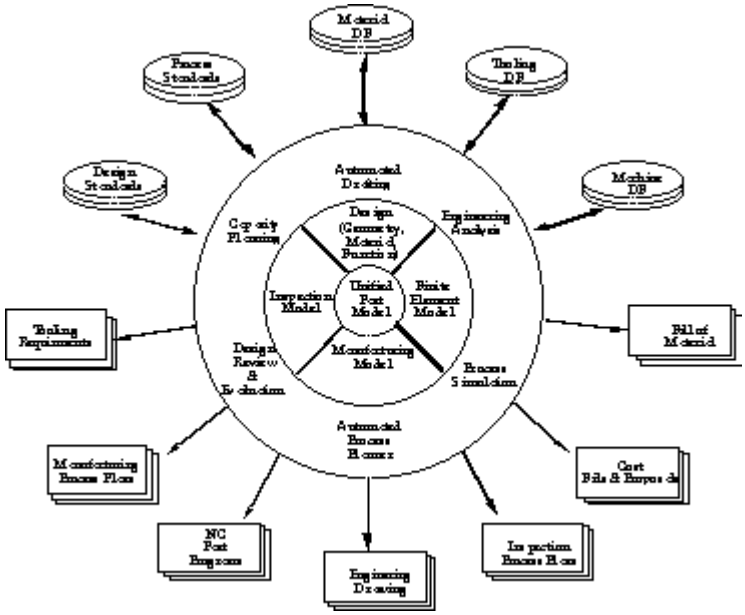
Chapter 1. The Adaptive Modeling Language

AML is an Adaptive Modeling Language for knowledge-based concurrent engineering. It is a comprehensive modeling paradigm to integrate design specifications, part geometry/features, manufacturing, inspection, and analysis processes in a unified part model. AML provides a Knowledge Based Engineering (KBE) framework that captures knowledge from the modeled domain and creates parametric models with that knowledge. Classes inheriting from AML primitives may be defined and methods may be written against these classes providing user-defined behavior. After defining the classes, a hierarchical part model is instantiated where the attributes of objects can be related using unidirectional non-cyclic constraints. This part model may be utilized as a parametric design in a "what-if" scenario by changing design parameters and re-computing the model as the constraints are propagated through the model on demand. AML is "adaptive" in that it can be used to model a wide range of domains that have interacting components and constrained behavior between them. Hence it can be adapted to diverse engineering applications. In addition, AML is dimensionless and may be adapted to utilize any dimension units.

Unified Model

Various aspects of a problem can be detailed through a single unified model in AML. An example of such an application is structural design. A geometric design is created followed by the association of various physical attributes with the geometry. Then the attributes for a finite element mesh, and the knowledge required for generating input files for analysis, are maintained. AML allows all this information to be stored in a single model in a structured fashion. Furthermore, knowledge for manufacturing, inspection, and tooling can be incorporated in the same model for the automation of the manufacturing and inspection process plans. Feedback could be provided at various stages to different entities in the model. A complete user interface for the problem including input and output forms, menus etc. can also be associated with the same part model that encompasses the various aspects of the application.

Figure 1-1. Unified Part Model

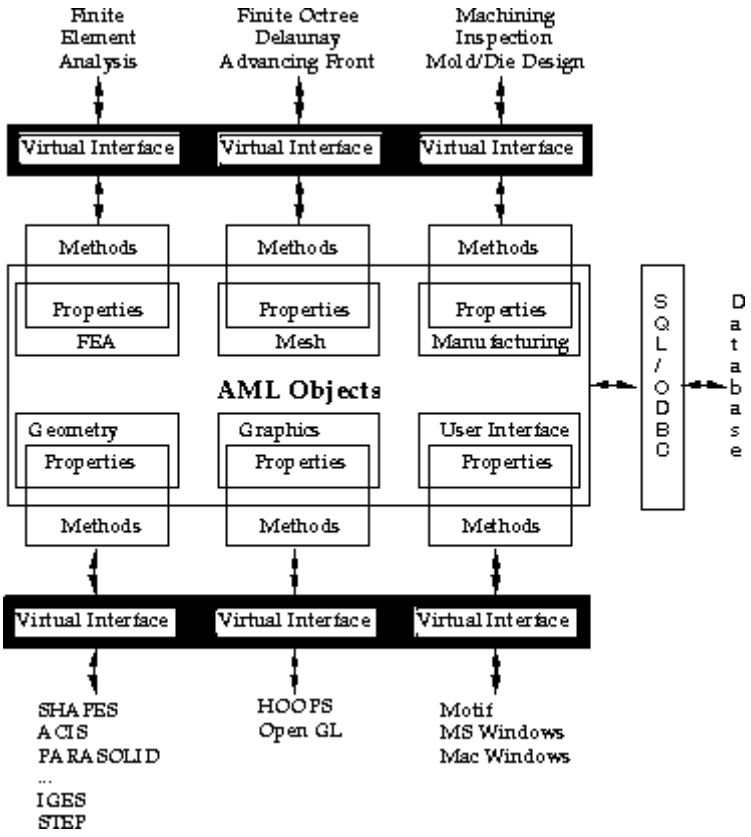


Virtual layers

The virtual layer paradigm built on AML enables the engineer to emphasize on solving the problem, rather than the various means of solving it. Consider a structural design and analysis problem where a solid modeler is needed to create a design, then a mesh generator is required to create a finite element mesh, and finally an analysis program will have to generate a solution. At each stage there are various programming techniques available which have their own interface and hence require an engineer to be cognizant of the various syntaxes. The AML paradigm provides a common interface to a number of solid modelers in addition to different mesh generators and FEA solvers. This is achieved by the virtual layers implemented thus providing a common consistent interface. Another advantage of providing virtual layers is that the problem can be approached using different solution strategies. For example, the same geometric design could be created using either the AML SHAPES-based or the AML Parasolid-based solid modeler, to compare accuracy and performance, or imported via a STEP file from ProEngineer or other CAD systems. Similar Virtual Layers could be created over the other components of AML ensuring compatibility with existing systems, as well as a common interface for accessing them. A Virtual Layer for Meshing has been implemented with Patran.

System Architecture

Figure 1-2. AML Architecture



The system consists of several AML modules (sets of classes and methods) relating to the different knowledge domains each performing a different function as illustrated in the figure above. All the modules are written within the AML object-oriented architecture although they do communicate with external programs through the Virtual Layers. Additional modules can be defined and loaded into AML to 'adapt' the language for a specific purpose. Since AML is modular, only the necessary systems need to loaded into AML. Hence, if a problem requires a modeling framework but no graphics or geometry, only the kernel can be loaded for an application. Applications invoking different aspects of the system built in AML utilize a common user interface to the system. Hence, user familiarization is required only with one interface irrespective of the applications - Design, Analysis, Manufacturing, and Inspection, all applications utilize a common AML interface.

The AML Kernel

The lowest level of the AML objects as illustrated in the example text below provide the language constructs for defining classes objects, methods, and the constraint mechanism. All subsequent objects simply augment the language. The AML core system provides the ability to define classes and methods for objects and properties. The constraint mechanism, the part hierarchy and other basic language

constructs are also provided by AML. It also defines the *OBJECT* class that all AML classes are inherit from.

Object Oriented Design

AML's object-oriented design paradigm provides two different types of relations. A *Class-Subclass* relation allows data abstraction, encapsulation, sharing of data structures and behavior, and polymorphism. Subclasses can be derived from any of the AML classes, or from user-defined classes. Multiple inheritance is available for classes. A class can be derived from an existing class, and new properties can be added, or formulae and values redefined for existing properties, as shown below.

```
(define-class class-name
:inherit-from (class-list)
:properties (property-list)
:sub-objects (subobject-list)
)
```

The class-list is a list of classes to inherit from. The property-list is a list of object properties that are defined similar to the objects. The subobject-list is a list of objects directly located under the object being defined in the part-subpart hierarchy. A *Whole-Part* relation enables the creation of a tree structure unified model where the children of any node of the tree represent sub-objects. Various aspects of the problem can be structured hierarchically according to the domain being modeled (Design, Analysis, Manufacturing and Inspection).

Constraint Mechanism

AML's underlying constraint mechanism supports *demand driven* and *dependency backtracking* behaviors. Demand-driven refers to the fact, that the value of a property is not calculated until it is demanded. Until a value is demanded, an internal flag refers to the property value as being unbound or the property being smashed. Hence several properties that affect a certain property can be modified, but the effected property does not need to be recalculated every time, but only when it is finally needed. Dependency backtracking is the mechanism that actually propagates constraint changes throughout the part model. When a property is modified, all the properties or objects that it affects are smashed. When a property is smashed, it further smashes all properties or objects that it affects, hence propagating the change by notifying entities that they need to be recalculated when demanded next.

Tree search

The *the* macro provides the means for querying the model for properties and objects, as well as establishing constraint relationships. The *select-object* function provides a means of selecting objects over the entire model, or for a particular branch of the tree.

Parametric Design

The constraint mechanism coupled with the tree-search provides the parametric modeling environment. Several properties of the model can be changed and then the results can be computed, which may result in new geometry or different outputs. Hence a "what-if" scenario can be achieved without the user

having to manually notify entities of change. The change-value and change-formula methods assist in modification of properties.

Event Triggers

Although a demand-driven paradigm suffices for several applications, situations arise that require notification of entities that may be outside the domain of dependencies, and actions need to occur at the moment an event takes place. The AML system defines classes that can be inherited into objects to define actions on creation, deletion, change and smashing of properties or objects. An example of the use of an event trigger is in the way AML handles the freeing of external pointers on notification from AML events. When a property that holds a pointer external to AML, is smashed, a trigger is activated, that invokes the system call to destroy the pointer, and unbound the property. This would not be achievable using dependencies alone, since the pointer is external to the constraint network. This mechanism is unique to the AML functionality of the Virtual Interface that independently manages automatic allocation of memory to external applications.

Dynamic modification of a model

Since design is inherently iterative and dynamic, AML also attempts to be dynamic in nature. Values and formulae of properties can be changed after the model is instantiated. AML also permits the addition and deletion of objects and properties after the model is created. The model definition can be created using the define-class macro, instanced, and worked on. At a later stage, new objects and properties can be added to the model. Functions like add-property, add-object, delete-property, delete-object can assist in dynamic modification of the model.

Property Objects

So far, AML has been described as having a schema where properties are type free. AML also permits the definition of new property classes inheriting from existing ones such as *property-object*. Properties can be added to and methods written against a property class like any other class. In fact, property-object is also derived from object, with the additional slots described earlier (value, formula, etc.). In the framework, the property-objects will be the basis for accessing the virtual interfaces to communicate with external databases and foreign applications.

Methods

As mentioned earlier, methods can be defined against any AML class or one derived from an AML class. This lets the user modify behavior of classes according to the needs of the application. Here, polymorphism as well as 'virtual functions' can be utilized.

Geometry

The various applications of AML including CAD, layout and configuration, CAM, and FEM/FEA, have different geometric requirements. These individual requirements are satisfied through the capability of augmenting the part model for different representations to satisfy various demands of the various applications. These different representations are manipulated through a unified part model. AML presents a number of objects/classes for modeling simple primitives in addition to complex geometrical operations. Complex operations for mixed-dimensional solid/surface/wire-frame booleans incorporating

non-manifold topology are also supported. Additional objects for advanced modeling of free-form surfaces (NURBS, Beziers, etc.) also exist. The system supports an IGES, STEP, and DXF interface to external CAD systems.

Geometric Reasoning

AML supports geometric reasoning for process planning automation to integrate the part design with the manufacturing/inspection/analysis plan for simultaneous engineering. Various queries and methods to enhance and modify the part geometry as required by the manufacturing, analysis and inspection processes are supported by AML. For example, queries about a distributed set of points on a free form surface, along with their normals and connected path could be dynamically queried and presented in a separate object. This object could be used for an inspection plan.

Graphics

Through its virtual layer capabilities, AML supports advanced visualization techniques for manipulation and display of the geometrical objects. These include color contour and vector plots.

Parametric Feature Based Design Environment

The system provides a unique interactive design environment; It is parametric, constraint driven, free-form feature based, has solid and surface modeling capabilities. Geometric as well as non-geometric features can be modeled. The system will enable easy referencing and parametric association for feature properties that could be linked to external processes as a part of the Virtual Layer Interface.

User Interface

AML provides a complete set of interface classes including forms, buttons, radio boxes, check boxes, input forms and pop up menus supported on Unix/Motif and Windows/NT. Since the user interface model is represented using the same knowledge representation system as the applications, it too is dynamic in nature, and the attributes of user interface entities such as color and size of buttons can be altered dynamically.

Attribute Tagging and Propagation

Attribute Tagging and Propagation is being utilized for facilitating association of information with entities in a geometric model. This information is typically such that needs to be conveyed to downstream processes such as manufacturing, inspection, meshing or analysis. Typically, when a geometric model is constructed, several stages of construction geometry are required. These construction entities are then "booleaned", transformed, swept, revolved, etc. to finally create the model. In a parametric modeling environment, reconfiguring a model involves the modification of parameters at the construction level and regeneration of the geometric model. Hence, all supplementary information would need to be re-conveyed to the final model as well as downstream processes every time the model is reconfigured. First, using Attribute Tagging, the supplementary information is associated with construction configurable geometry. Next, every downstream operation, including final design, analysis, manufacturing, have the information passed on through Attribute Propagation. Attribute Propagation

ensures that every operation on the geometry including reads/writes, booleans, sweeps, etc. propagate the attributes through the operation. As a result, when the model is reconfigured i.e. upstream design entities are modified in geometry or other properties, the Attribute Propagation mechanism ensures that supplementary information is passed downstream automatically. The Attribute Tagging and Propagation mechanism is integrated with the demand driven and backward propagation mechanism using Event Property objects.

Mesh Generation

The *Automatic Mesh Generation* system (Figure 2) is an AML module that allows tight integration of various mesh generation and analysis applications. The system provides a Virtual Interface to support various third party mesh generators. The system permits the selection of geometry to mesh, tagging the vertices, edges and faces of geometry for selective refinement of the mesh, and meshing the geometry by calling the external mesh generator. It provides objects for meshing as well as a user interface for the same. It also provides methods and a user interface for visualizing the mesh by querying the mesh database created by the mesh generator.

Finite Element Analysis (FEA)

The *Finite Element Analysis (FEA)* system (Figure 2) enables the definition of an analysis problem by defining regions of interest, material models, solution strategies and other requirements for analyzing various problems utilizing a Mesh generator and a Finite Element Solver. The various entities of interest are modeled as AML classes that can be utilized to instance a complete Finite Element Analysis problem model. The problem can be associated with the geometric objects as well as the mesh. The system generates several files that the solver can read and execute to generate results. This system too can be extended to provide a Virtual Solver Layer that can talk to various FEA solvers.

Summary

The framework provides a versatile parametric modeling environment supporting a unified, adaptive part model. AML offers a flexible modeling environment that can be utilized for a spectrum of engineering problems. AML enables the abstraction of the modeled domain into a set of interacting entities which can then model various scenarios. These problems may have a large number of attributes that are linked through a non-cyclic unidirectional constraint graph. AML can also be applied to problems requiring a high degree of visualization of entities. Applications requiring complex geometric operations are also well suited to AML. The interpretive environment is suited to simulating "what-if" scenarios and iterative modeling environments.

Chapter 2. AML Basics

Defining Classes

The define-class Construct

define-class [Language Construct]

Define is used to define a new AML class. Defining a new class allows instances of that class to be created. When a class is defined, its parent class(es) (i.e. the class(es) it directly inherits from) must be defined. A class that inherits from another class inherits all the properties and methods (see define-method later in the chapter) defined on the parent class. In other words, all properties of the parent class will also be properties of the new class and all methods defined on the parent class can also be called on instances of the new class. When defining a class, the following can also be specified:

1. New properties and formulas that the new class defines.
2. New formulas for the properties that the class inherits.
3. Subobjects of the new class.

Creating an instance of a class is done using the create-model and add-object functions. Create-model will create an instance and place the instance at the top of the model system. Add-object will create an instance and insert that instance into the model at a specified location. There is another function which can be used to create object instances. Create-object will create an instance and return the instance without adding that instance to the model. The instance will be outside of the model tree. The use of create-object is highly discouraged. Format: (define-class class-name :inherit-from(class1 class2 ... classN) :properties(Properties specification) :subobjects(Subobjects specification))

Arguments:

- **class-name**

  Any symbol to be used as the name of the class being defined.

- **:inherit-from**

  A list of predefined classes this class should inherit from.

Keyword Arguments:

- **:properties**

  A list of property specifications. A specification may be a simple property/formula pair or an object specification list containing the name, class, and property/formula pairs for the property object. When a class is specified, the class should inherit from property-object if the user wants the property to behave like a standard AML property with a formula and a value. The class can be the filename of a saved model.

- **:subobjects**

  A list of subobject specification lists containing the name, class, and property/formula pairs for the subobject. The class of the subobject may not inherit from property-object. The class can be the

filename of a saved model. The name of a subobject must be a symbol and cannot be a string or a number.

1. The class used in the specification of a subobject or a property object should be the name of a defined class.
2. Since the items in the reference path of the can find an object based on its class, using a class name as a subobject name or a property name can be confusing and typically should be avoided.

See Also: the

Specifying Inheritance

Inheritance is a mechanism for class reuse. Through inheritance, a class will have all of the properties, subobjects, and methods of the classes that it inherits from (its direct super-class(es)). The inherit-from keyword accepts a list of classes to inherit from. If a property, subobject, or method is present in more than one of the super-classes, the order of precedence is from left to right. That means that if an :inherit-from contains the list (box-object cylinder-object), the box-object geometry will be created. If the order of the list is reversed to (cylinder-object box-object), then the cylinder-object geometry will be created. The highest level class that a user can inherit from is the class object.

Specifying Properties

The optional :properties keyword is used to define properties and their associated formulas. Properties may be specified as a simple property/formula as shown in the define-class example or as a property object specification as shown in the property objects section. The properties may be the names of new properties to be added to the class or names of properties that are defined in the super-classes. If the names are the same as those in super-classes they are considered overriding properties; the formula of the overriden property is now that of the new class. The formula of a property can be a function call, method call, any valid AML statement, or a simple value/number/string/symbol. To access property values, the referencing is used (see also !, and ^).

Note: properties can be an instance of multiple classes. For reference look at the second example below.

Specifying Subobjects

The optional :subobjects keyword for define-class must be a list of subobject specifications. The subobjects will be created as child instances of the instance of the class being defined. Defining subobjects requires the following format: (subobject-instance-name :class class-to-be-instantiated property-specifications). The subobject-instance-name is a symbol that will become the name of the subobject instance. The class-to-be-instantiated may be any expression that evaluates to a valid class name. The expression may also use the references to establish dependencies between the properties and the class of subobject created. The property-specifications at the subobject level are exactly the same as the :properties level for define-class.

Note: Within the class-to-be-instantiated expression, if the referencing is used, "the" starts its search at the instance of the parent object and not at the subobject.

Note: subobjects can be an instance of multiple classes. For reference look at the second example below.

Remark:

There are a number of internal system properties that are defined for all classes. Specifying these as properties, methods or subobjects will be futile because the system will use the internal properties and ignore others of the same name. The reserved property names are: deletable?,property-table,subobject-table,object-class,object-name,depend-on-instance-list,effect-instance-list,superior,self,level-number.

Examples and functions

Example 2-1. Defining Classes

```
(in-package :aml)

;; For AML classes, methods and functions used in this example, please refer
to the index of the AML reference manual

(define-class MATERIAL-OBJECT
  :inherit-from (object)
  :properties
  (
   material nil
   density 1.0
   volume 1.0
   weight (* ^density ^volume)
   )
  )

(define-class c1
  :inherit-from (object)
  :properties (
               (p1 :class '(saved-property private-property)
                   formula 2
                   )
              )
  :subobjects (
               (s1 :class '(tagging-object box-object))
               )
  )

(define-class WOODEN-OBJECT
  :inherit-from (material-object)
  :properties
  (
   color 'brown
   material 'wood
   density 0.03  ; oak ~0.85 kg/dm^3
   )
  )

(define-class STEEL-OBJECT
  :inherit-from (material-object)
```

```
  :properties
  (
   color 'cyan
   material 'steel
   density '0.1  ; chrome-nickel steel 7.9 kg/dm^3
   )
  )

(define-class TABLE-TOP
  :inherit-from (wooden-object box-object)
  :properties
  (
   volume (* ^height ^width ^depth)
   )
  )

(define-class TABLE-LEG
  :inherit-from (steel-object cylinder-object)
  :properties
  (
   volume (* pi (half !diameter) (half !diameter) !height)
   )
  )

(define-class TABLE
  :inherit-from (assembly-object)
  :properties
  (
   table-height 36.0
   table-width 72.0
   table-depth 36.0
   thickness 1.0
   display? nil
   total-weight (loop for i in (select-object
                                :from (the superior)
                                :class 'material-object
                                :eval '(the weight))
                  sum i)

   (object-list :class 'private-property
                formula (remove
                         (the superior)
                         (select-object
                          :from (the superior)
                          :class 'graphic-object
                          ))
                )
   (origin :class 'point-object
           coordinates '(0.0 0.0 0.0)
           display? t
           )
   )
  :subobjects
```

```
(
 (top :class 'table-top
      height ^^thickness
      width ^^table-width
      depth ^^table-depth
      display? (not ^^display?)
      reference-object (the superior superior)
      orientation (list
                   (translate (list
                               0.0
                               (- ^^table-height
                                  (half ^^thickness))
                               0.0)))
      )
 (legs :class 'series-object
       series-prefix 'leg
       class-expression 'table-leg
       init-form
       '(display? (not ^^^display?)
         height (- !table-height !thickness)
         diameter !thickness
         reference-object (the superior superior superior)
         orientation
         (list
          (rotate 90.0 :x-axis)
          (translate
           (list (if (evenp ^index)
                     (- (half (the table-width))
                        (the thickness))
                   (- (the thickness)
                      (half (the table-width)))
                   )
                 (half (the height))
                 (if (> (the index) 1)
                     (- (half (the table-depth))
                        (the thickness))
                   (- (the thickness)
                      (half (the table-depth)))
                   ))))
         )
       quantity 4
       )
 )
)
```

class-direct-defined-properties [Function]

Returns the list of names of the properties directly defined by the class specified.

Arguments:

- **class-name**

  Name of the class in question.

See Also: class-direct-defined-subobjects

class-direct-defined-subclasses [Function]

Returns the list of names of classes that directly inherit from the class specified.

Arguments:

- **class-name**

  Name of the class in question.

See Also: class-direct-defined-superclasses

class-direct-defined-subobjects [Function]

Returns the list of names of the subobjects directly defined by the class specified.

Arguments:

- **class-name**

  Name of the class in question.

See Also: class-direct-defined-properties

class-direct-defined-superclasses [Function]

Returns a list of the class names that a class directly inherits from, that is one level up the inheritance tree.

Arguments:

- **class-name**

  Name of the class in question.

See Also: class-direct-defined-subclasses

find-class [Function]

This function is used to check for the existence of a class. It returns a non-nil value if the symbol passed to it is the name of a defined class.

Arguments:

- **symbol**

  Symbol (name) of class in question.

Optional Arguments:

- **error?**

  Optional argument. When nil, the function returns nil if the symbol passed is not the name of a defined class; otherwise the function signals an error.

  Default Value: t

Example 2-2. Using find-class

```
(find-class 'dummy-class nil) ;; returns nil
```

```
(find-class 'box-object nil)  ;; returns a non-nil value (reference to the
class definition)
```

The Referencing

Functions

the [Function]

the is used to reference an object or a property in an AML object tree. The default behavior is as follows: A reference to an instance that inherits from property-class will return the value of that property. A reference to any other object will return the instance of the object.

Optional Arguments:

- **reference-path**

  The reference-path is a series of object and property names identifying a tree path to a desired object or property. The symbol superior can be used in the path. The symbol superior specifies the desire to move one level up from the current level. For example: (the superior) is the parent instance of the instance we are tracing from, (the superior superior) is the parent instance of the parent instance of the instance we are tracing from, (the table leg1 height) is a reference to the height property of an instance named leg1 that is a subobject of an instance named table. Other example references: (the superior superior box width), (the superior cost), (the superior superior superior table leg1 bottom-surface)

Keyword Arguments:

- **:from**

  Optional keyword to specify the instance to start tracing from. If not specified, the referencing starts at the current instance: Within the formula of a property, the current instance is the property object itself. Inside the source code of a method, the current instance is the instance of the class that the method was called on. Within the source code of a function, the current instance is the current instance at the time the function was called. At the AML command prompt, the current instance is the root instance of the current model.

  Default Value: (the)

- **:eval?**

  Optional keyword. This keyword is effective when referencing properties only (instances of property-class). When t, the referencing returns the value of the property being reference, when nil, it returns the property object instance instead.

  Default Value: t

- **:dependencies?**

  Optional keyword. This keyword is useful when a reference is made from within a property formula: When this keyword is t, the formula of the property, when evaluated, sets a dependency on the property/object being referenced. When nil, no dependency is set.

  Default Value: t

- **:relation**

Optional keyword. This keyword specifies where to restart the tracing in case an element of the reference path is not found at the current level. Default is (the superior), which will cause referencing to retrace from the superior instance (recursively) if a property/object name is not found at the current level. When nil, no search is performed and a reference-path is expected to be an exact path to the desired instance.

Default Value: (the superior)

- **:error**

  Optional keyword that specifies what to return if the property or object is not found. The default behavior will cause an error if the property or object is not found.

  1. If the first symbol in the reference-path is NOT the name of a property or subobject of the current instance (with the exception of the symbol superior), and the relation keyword uses its default value (i.e. is not set by the user): the referencing will search up the object tree until it finds the first object whose name matches the beginning of the reference-path and evaluates the rest of the reference-path from there. For example, given the following object tree:

     ```
     Table
     --Top
     ----width
     ----height
     --Legs
     ----leg1
     ----leg2
     ----leg3
     ------height
     ------diameter
     ```

     From within the formula of the diameter property, (the table top width) is a valid the reference and is equivalent to (the superior superior superior top width).

  2. When referring to a property, the symbol self can be appended to a reference-path in order to return the property object instance instead of the property value. This is equivalent to setting the keyword eval? to nil. Example: (the table top width self)

  3. If the not reference-path is given, (the), the current instance is returned. If the current instance is a property, its value is returned. Note that a (the) reference from within the formula of a property will cause a circular dependency.

Example 2-3. The Example

```
(the mode1 object1 subobject1)

(the something (:from (model-manager) :error nil))
```

See Also: the-list

the-list [Function]

The-list has the same purpose of the referencing described above. It is used instead when the reference path is in the form of a list, which gives more flexibility when the reference path is variable.

Arguments:

- **reference**

  A list of object or property names used as a reference-path (as described with the reference above) to the desired object or property.

Keyword Arguments:

- **:from**

  The instance to start tracing from. The default is the current instance: Within the formula of a property, the current instance is the property object itself. Inside the source code of a method, the current instance is the instance of the class that the method was called on. Within the source code of a function, the current instance is the current instance at the time the function was called. At the AML command prompt, the current instance is the root instance of the current model.

- **:eval?**

  Optional keyword. This keyword is effective when referencing properties only (instances of property-class). When t, the referencing returns the value of the property being reference, when nil, it returns the property object instance instead.

  Default Value: t

- **:dependencies?**

  Optional keyword. This keyword is useful when a reference is made from within a property formula: When this keyword is t, the formula of the property, when evaluated, sets a dependency on the property/object being referenced. When nil, no dependency is set.

  Default Value: t

- **:relation**

  Optional keyword. This keyword specified where to restart the tracing in case an element of the reference path is not found at the current level. Default is (the superior), which will cause referencing to retrace from the superior instance (recursively) if a property/object name is not found at the current level. When nil, no search is performed and a reference-path is expected to be an exact path to the desired instance.

- **:error**

  Optional keyword that specifies what to return if the property or object is not found. The default behavior will cause an error if the property or object is not found.

Remarks The-list is typically used, instead of the regular the referencing, when the reference path contains variable elements.

See Also: the reference

Example 2-4. Using the-list

```
;; Assuming the class table from example2.0 is loaded, try the following at
the AML command prompt:

(create-model 'table)
; This returns the table instance that was created

(the-list '(table))
; This returns the table instance

(the-list '(table top height))
; This returns the top height value

(the-list '(diameter) :from (the-list '(legs leg-0000)))
; The legs leg-0000 diameter value

(the-list '(superior) :from (the-list '(table legs)))
; The table instance

(the-list '(table top height) :eval? nil)
; The table top height instance

(let ((a 'table)
      (b 'height))
  (the-list (list a 'top b))
  )
;; This returns the table height
```

! and ^ (the shortcut characters) [Language Construct]

The shorthand methods for referencing properties and objects are ! and ^. Here are examples to illustrate: !leg1 is shorthand form of (the leg1), ^height is a shorthand form of (the superior height), ^^leg1 is a shorthand form of (the superior superior leg1)...etc. The use of keywords is not allowed with these shorthand forms.

Defining Methods

Methods

define-method [Function]

Define-method defines a named method and functional behavior for objects of a given class. A method is a function that can only be called in conjunction with an instance of the given class or any of its subclasses. When called, a method always returns the value returned by the last statement of the method body. When a method is defined on a class A, it can be called on instances of class A and instances of all subclasses of A. If class B is a subclass of class A, the behavior of a method M defined on A can be overridden by redefining M (using the same method name) on class B.

Arguments:

- **method-name**

The method name, it must be a symbol.

- **class**

  The class name of the class that the method is written for.

Optional Arguments:

- **args**

  Arguments for the method. There are 3 types of arguments: required arguments, optional arguments, and keyword arguments. When a required argument is defined with a method, the user must specify its value when he/she calls the method. An optional argument/keyword is assigned its default value in case the user does not specify it when the method is called. Keywords are a special form of optional arguments that allow the user to call the method with a value for a specific keyword without having to worry about the order in which keywords are defined and without necessarily specifying any other keyword. For that reason it is recommended to use keywords instead of optional arguments, and the combination of keywords and optional arguments within the same method definition is discouraged. Definition and usage of optional and keyword arguments for a method are the same as those of a function's optional arguments and keyword. See examples listed with defun.

- **body**

  The action(s) (AML statements) the method will perform when called.

  1. When a method is called, the method's object instance is always passed as the first argument to the method call.
  2. Within a method, The-referencing starts tracing at the instance that the method was called on.
  3. Within a method defined on an object that does not inherit from property-class, the statement (the) returns the object instance itself. Within a method defined on a property-object (or any class that inherits from property-object), the statement (the) returns the value of the property.
  4. A define-method may not be defined on a class that has not been previously defined and loaded into memory. It is suggested that source code is organized in a way that methods definition immediately follow the class they are defined for.
  5. When overriding a method on a sub-class, you should use the same name, required arguments, optional arguments and keywords as in the original method defined on the super-class.
  6. When a method M is called on an instance of class C, and method M is not defined on class C, the method named M defined on a super-class of C is used. If M is defined on more than one super-class of C, the one belonging to the closest super-class (in the class inheritance hierarchy) is used. For example, let's assume the following: Class cube-class inherits from box-class and box-class inherits from geometry-class. Method volume-of-object is defined on box-class and is also defined on geometry-class. If volume-of-object is called on an instance of cube-class, the method code defined with box-class will execute and not the code defined with geometry-class.
  7. Do not use self as a variable within a method or as an argument to a method because self is a ystem variable that is always equal to the object instance that the method was called on.

See Also: defun, call-next-method, the-referencing

Example 2-5. Defining Methods

```
(in-package :aml)
```

```
(define-method weight box-object (density)
  (let ((volume (* (the height) (the width) (the depth))))
    (* density volume))
  )

(define-method weight cylinder-object (density)
  (let ((volume (/ (* pi (the diameter) (the diameter) (the height))
                   4.0)))
    (* density volume))
  )

;; Below is a chunk of code showing how the method can be called

(let ((table-object (add-object (root-object) 'table 'table))
      )
  (weight (the top (:from table-object)) 0.1) ; The weight using the method
for box-object
  (weight (the legs leg-0000 (:from table-object)) 0.05)) ; The weight using
the method for cylinder-object
```

call-next-method [Function]

This function finds the next same method to call according to the instance's inheritance list. The method must exist and has exactly same arguments definition.

Defining Functions

Functions

defun [Function]

defun defines an AML function. It always returns the value returned by the last statement of the function body.

Arguments:

- **function-name**

  The function name, it must be a symbol.

Optional Arguments:

- **args**

  Arguments for the function. There are 3 types of arguments: required arguments, optional arguments, and keyword arguments. When a required argument is defined with a function, the user must specify its value when he/she calls the function. An optional argument/keyword is assigned its default value in case the user does not specify it when the function is called. Keywords are a special form of optional arguments that allow the user to call the method with a value for a specific keyword without having to worry about the order in which keywords are defined and without necessarily specifying any other keyword. For that reason it is recommended to use keywords instead of optional arguments, and the combination of keywords and optional arguments within the same method definition is discouraged.

- **body**

  The action(s) (AML statements) the function is going to perform.

When calling a function that has keywords, both the keyword name and value must be specified and the keyword name must be preceded by a semi-colon. Refer to example below.

See Also: define-method

Example 2-6. Defining Functions

```
(in-package :aml)

;; Defining the function quadratic
(defun quadratic (a b c)
  (let* (
         (radical (- (expt b 2) (* 4 a c)))
         (demoninator (twice a))
         (numberator+ (+ (- b)
                         (sqrt radical)))
         (numberator- (- (- b) (sqrt radical)))
         )
    (list (/ numberator+ demoninator)
          (/ numberator- demoninator))
    )
  )

;; Calling the function quadratic
(quadratic 1 3 2) returns (-1.0f0 -2.0f0)

;; Defining the function draw-objects-from-list using a keyword
(defun draw-objects-from-list (object-list &key (echo? nil))
  (loop
    for obj in object-list
    do (when echo? (format t "Drawing: ~a~%" (object-name obj)))
    (draw obj)
    )
  )

;; Calling the function draw-objects-from-list

(create-model 'name-generator)
(add-object (the name-generator) (generate-name (the) 'obj) 'box-object)
(add-object (the name-generator) (generate-name (the) 'obj) 'box-object)
(draw-objects-from-list (children (the name-generator)))
(draw-objects-from-list (children (the)) :echo? t)
; This will print object names on the command line buffer while drawing
```

Example 2-7. Defining Functions with Optional and Key Arguments

```
(in-package :aml)

;; Defining the function quadratic using optional arguments:
(defun quadratic1 (&optional (a 1.0) (b 1.0) (c 1.0))
  (let* (
```

```
          (radical (- (expt b 2) (* 4 a c)))
          (demoninator (twice a))
          (numberator+ (+ (- b)
                          (sqrt radical)))
          (numberator- (- (- b) (sqrt radical)))
         )
    (list (/ numberator+ demoninator)
          (/ numberator- demoninator))
    )
  )

;; Calling the function quadratic
;; This call will assign 2.0 to the argument and
;; will use the default values for b and c:
(quadratic1 2.0)
;; If the user needs to set the value of c, a and b
;; must also be specified because c is defined at the end:
(quadratic1 2.0 1.0 -3.0)

;; Defining the function quadratic using keyword arguments:
(defun quadratic2 (&key (a 1.0) (b 1.0) (c 1.0))
  (let* (
          (radical (- (expt b 2) (* 4 a c)))
          (demoninator (twice a))
          (numberator+ (+ (- b)
                          (sqrt radical)))
          (numberator- (- (- b) (sqrt radical)))
         )
    (list (/ numberator+ demoninator)
          (/ numberator- demoninator))
    )
  )

;; Calling the function quadratic
(quadratic2 :b 4.5 :c 1.0) ;; will use the default value for a
(quadratic2 :a 1.2 :c 4.3) ;; will use the default value for b
(quadratic2) ;; will use default values for all arguments
```

Basic AML Classes and Associated Methods and Functions

Functions, Methods and Classes

address-from-object [Function]

Takes an AML object as input, returns the memory address of the object. The address is valid in the AML session until the object becomes garbage and collected by GC. The object can be restored by calling object-from-address function.

Arguments:

- **object**

  object instance

Example 2-8. Address-from-object example.

```
(the)
;;;returns #lt;interface @0x02CF44A4gt;
(address-from-object (the))
;;;returns 47137956
(object-from-address 47137956)
;;; returns #lt;interface @0x02CF44A4gt;
```

aml-class [class]

This is the super-class of all AML classes. Every class in the Adaptive Modeling Language inherits from this class. This class must not be instantiated as is; it is intended for internal use to provide functionality that all classes require.

adaptive-class [class]

This class is an internal system class and should not be instantiated by the user. This class mainly allows the dynamic addition and deletion of objects and properties to any existing AML object that inherits from it.

Inheritance: aml-class

null-object [class]

This class is used to instantiate an object of an "unkown" class with no properties. It is useful when defining a subobject within a class definition whose class is a conditional statement. This class should not be used in the inherit-from section of a define-class statement, rather it can be used when defining the class type for a subobject or property object.

Inheritance: adaptive-class

Example 2-9. Using null-object

```
(in-package :aml)

(define-class adaptive-boolean-class
  :inherit-from(object)
  :properties(
              boolean-type 0 ;; 0 for union, 1 for difference
              )
  :subobjects(
              (box :class 'box-object
                   depth  1.0
                   height 1.0
                   width  1.0
                   )
              (cylinder :class 'cylinder-object
                        diameter 0.5
```

```
                    height   2.0
                    )
            (boolean-object :class (case !boolean-type
                                     (0 'union-object)
                                     (1 'difference-object)
                                     (t 'null-object) ;; Unknown state
                                     )
                            object-list (list ^^box ^^cylinder)
                            )
            )
)
```

object-class [class]

This class provides properties that are needed for internal use and the ability to track dependencies. These properties may be referenced but not initialized by any subclasses. This is a system super-class and should not be instantiated by the user.

Inheritance: aml-class

Properties:

- **added-objects**

  List of object instances that were dynamically added to this instance, i.e. subjects of this instance that are not defined in the class definition

  Default Formula: nil

object-root-class [class]

This class is the root class of all objects that are not property-objects. This class is a system super-class and should not be instantiated by the user.

Inheritance: adaptive-class

object [class]

Object is the highest level class an AML user can inherit-from. In other words, while developing AML source code, a user defined class cannot inherit from any class that is a super-class of object. This class can also be directly instantiated.

Inheritance: object-root-class

add-object [Function]

Adds a named subobject to the object instance specified and returns the instance added.

Arguments:

- **parent**

  The object instance that the new object will be added to.

- **name**

  The name of the object instance to be added. This must be a symbol and cannot be a string or a number.

- **class**

  The class name of the object instance to be added, can be a symbol/name or a list of class names. The class can be the filename of a saved model.

Keyword Arguments:

- **:init-form**

  A list of property/formula pairs which overrides the formula of the properties specified in the class definition. A property mentioned in init-form that is not a property of the class specified will not be added to the object created.

  Default Value: nil

- **:eval?**

  This causes the added object to be instantiated immediately. A nil value means that the instance will not be created until the object is referenced.

  Default Value: t

Example 2-10. Using add-object

```
(add-object (the table) 'leg 'table-leg
   :init-form '(diameter !thickness
                height (- !table-height !thickness)))
```

added-objects [Method]

Defined on Classes:

aml-class

This method returns the list of added-objects of the given object instance. An added-object is an object that was added at runtime as a sub-object of another object, i.e. it is not a sub-object defined in a class definition.

Arguments:

- **Instance**

  An object instance.

add-property [Function]

Adds a property of class property-object to the object instance provided. This will add a property and an associated formula. This function returns the value of the new property if the eval? keyword is true, it returns nil otherwise.

Arguments:

- **instance**

  The object instance where the new property will be added.

- **property-name**

  The name of the property to be added, must be a symbol

- **formula**

  The formula of the new property.

Keyword Arguments:

- **:eval?**

  When t, causes the added property to be instantiated and its value to be demanded immediately.The function returns the value. When nil, the property is added but not referenced keeping its value unbound. The function returns nil.

  Default Value: t

See Also: add-property-object

Example 2-11. Using add-property

```
(add-property (the table) 'surface-area '(* (the table-width) (the table-
depth)))
```

add-property-object [Function]

This function allows for an instance of any class to be added as a property of a given instance. This function is useful when the intent is to a property instance of class that is more specific than a standard property-object. This function also allows the developer to add an instance that is not a property-object to the properties section of an object. This function returns the new instance added if the keyword "eval?" is t, otherwise it returns the name of the new instance.

Arguments:

- **instance**

  The object instance where the new property will be added.

- **property-name**

  The name of the property to be added, must be a symbol

- **class**

  The name of the class of the property to be added, must be a symbol

Keyword Arguments:

- **:eval?**

  When t, causes the added property to be instantiated immediately (and its value demanded when applicable).

  Default Value: t

- **:init-form**

  This keyword is used to initialize the property formulas of the instance being added. It is a list of property names and formulas, see example below.

See Also: add-property add-object

Example 2-12. Using add-property-object to add a box

```
(add-property-object (the) 'box 'box-object :init-form '(width 5.0 height 4.0
depth (* (the superior width) 2.0)))
```

Example 2-13. Using add-property-object to add a simple-property-class instance

```
(add-property-object (the) 'counter 'simple-property-class)
```

aml-save [Method]

Defined on Classes:

aml-class

This method allows the user to save the object instance provided into an AML object/model file. (The entire object tree under the given instance is saved). This method also saves all dependencies within that object. Using the function aml-load, the object can be retrieved later on as a model or as an object within another model. In order to successfully retrieve an object, the user has to make sure that the saved object does not contain any "the" reference outside of its tree branch; otherwise, the exact same tree structure is expected. This method returns nil if an error happens during saving. It returns the path string to the saved file on success. The class formula of added objects is saved when aml-save is called.

Arguments:

- **instance**

  An object instance.

- **file-name**

  File path string where the model/object is to be saved. The convention is to have a ".mdl" extension for the file name.

Keyword Arguments:

- **:append?**

  When non-nil, appends the saved model/object to the file if the file exists.

  Default Value: nil

- **:as-model?**

  This keyword is ignored. All instances are saved as regular objects.

  Default Value: t

- **:new-root-name**

  When non-nil, can specify a new name for the object/model to save. This can be dangerous because some of the references within the object tree may depend on the original name of the object.

  Default Value: nil

- **:force-overwrite?**

  When non-nil, the given file is overwritten without prompting the user.

  Default Value: nil

- **:user-name**

Allows the user to specify a user name. Has to be a single-line string.

- **:current-version**

  The version number/symbol/text of the current model, cannot be a multiple-line string.

- **:previous-version**

  The version number/symbol/text of the previous model, cannot be a multiple-line string

- **:description**

  Allows the user to enter text that describes the model, cannot be a multiple-line string

- **:compress**

  It allows the user to compress model files. The aml-compression system needs to be loaded for compression to work. If the compression system is not loaded, aml-save ignores the new keyword and does not compress the model file.

  Default Value: t

- **:save-loaded-objects?**

  If set to t, subobjects and properties that were created as a result of a reference to a model file are treated as regular added objects. Therefore, the model file being generated will contain these objects/properties and will not refer to the original model file(s) that created it. Otherwise, when this keyword is nil, these subobjects/properties are not saved in the model file being generated, only a reference to the original model file(s) is saved.

  Default Value: nil

- **:dir**

  It allows the user to specify the directory when the supplied file name is nil (interactive mode).

  Default Value: (logical-path :saved-models)

See Also: aml-load

children [Method]

Defined on Classes:

aml-class

Returns a list of the direct subobjects of an instance. Returns nil when no subobjects exist or no subobjects pass the :class and :test criteria.

Arguments:

- **Instance**

  The object instance whose subobjects you wish to return.

Keyword Arguments:

- **:class**

  If a class name (symbol) is specified, only children inheriting from that class will be returned. Default value is t which accepts all classes. This keyword can also be a list of class names; objects that inherit from any of the classes in the list will be returned.

Default Value: t

- **:test**

  If a test is specified, only children who pass the test will be returned. A test is a quoted AML expression. To refer to the object being tested within a test expression "(the)" is used (see example below).

  Default Value: t

- **:dependencies?**

  If the method is called from a property formula, this specifies whether or not to set up dependencies on the added-objects. A value of t means that the property calling this method will be smashed if an object is added.

  Default Value: t

- **:expand?**

  When this keyword is nil, only previously demanded subobjects are returned, i.e. unbound subobjects that have not been referenced/demanded will not be considered.

  Default Value: t

Example 2-14. Using children

```
(children (the object))
;; returns a list of object instances that are direct subobjects of "(the
object)"
(children (the object) :expand? nil)
;; returns a list of object instances that are direct subobjects of "(the
object)" but does not consider subobjects that have never been
demanded/referenced.
(children (the object) :class 'box-object)
;; returns a list of object instances of class box-object that are direct
subobjects of "(the object)"
(children (the object) :test '(or (equal (object-name (the)) 'box1) (equal
(object-name (the)) 'box1)))
;; returns the list of object instances that are direct subobjects of "(the
object)" and whose name is either box1 or box2.
```

delete-object [Function]

This function permanentely deletes the given object instance. It works on all object instances.

Arguments:

- **instance**

  The object-class instance to be deleted.

See Also: add-property-object add-object

delete-property [Function]

This function deletes a property of an object instance and returns the property deleted. If the property does not exist, it will generate an error.

Arguments:

- **instance**

  The object-class instance that contains the property

- **property-name**

  The name (symbol) of the property to be deleted

See Also: add-property-object add-property

Example 2-15. Example call to delete-property

```
(delete-property (the test-object) 'my-property)
```

depend-on-instance-list [Function]

This function returns a list of all the object/property instances that the given instance depends on

Arguments:

- **instance**

  The object/property instance to query

See Also: effect-instance-list

Example 2-16. Function usage example

```
(depend-on-instance-list (the test-model object1 property1 self)) ;; Returns
a list of object and property instances that property1 depends on
```

describe [Function]

Prints information about the given object instance. It lists properties, property values, and direct subobjects.

Arguments:

- **Instance**

  An object instance.

displayed? [Method]

Defined on Classes:

object

This method determines if the object instance specified is drawn on the current/active display window. It returns t if it is, nil if it's not.

Arguments:

- **instance**

  Object instance in question

draw [Method]

Defined on Classes:

object-class

General method for drawing objects on the currently active graphics display

Arguments:

- **instance**

  The object instance to be drawn (or root instance of the object tree to draw)

Keyword Arguments:

- **:draw-subobjects?**

  When non nil, draws the entire object tree whose root is the specified instance

  Default Value: t

- **:clear-display?**

  When non nil, clears the active graphics display before drawing.

  Default Value: nil

- **:test**

  This is a quoted AML expression that will be evaluated before drawing each of the targeted object instances. The expression must return a non nil value for the object to be drawn. Inside this expression the statement (the) evaluates to the current object being tested. The default value is t which causes all instance to pass the test.

  Default Value: t

- **:update-window?**

  When non nil, the window is refreshed at the end of the draw method.

  Default Value: t

See Also: undraw

Example 2-17. Using the Draw Method

```
(in-package :aml)

(define-class boxes-and-cylinders-class
  :inherit-from(object)
  :properties(
              )
  :subobjects(
              (box1 :class 'box-object
                    width 1.0
                    height 1.0
                    depth 1.0
                   )
              (cylinder1 :class 'cylinder-object
                         diameter 0.5
                        )
```

```
              (box2 :class 'box-object
                    width 2.0
                    height 2.0
                    depth 2.0
                    )
              (cylinder2 :class 'cylinder-object
                         diameter 2.5
                         )
              )
  )

;; This function creates a model and draws all boxes in the model tree

(defun create-and-draw-boxes ()
  (let ((model (create-model 'boxes-and-cylinders-class))
        )
    (draw model :test '(equal (type-of (the)) 'box-object))
    )
  )
```

effect-instance-list [Function]

This function returns a list of all the object/property instances that depend on the given object/property

Arguments:

- **instance**

  The object/property instance to query

See Also: depend-on-instance-list

Example 2-18. Function usage example

```
(effect-instance-list (the test-model object1 property1 self)) ;; Returns a
list of object and property instances that depend on property1
```

expand [Function]

Forces the expansion of the specified object, which means demands subobjects that have not been referenced yet.

Arguments:

- **Instance**

  The object to expand.

Keyword Arguments:

- **:all**

  By default, the function will only expand the immediate subobjects. If :all is t, the function will expand the entire subobjects' tree.

  Default Value: nil

object-branch-index [Function]

This function returns the depth (number of levels down from the root-object)of the specified object in the tree.

Arguments:

- **object**

  An aml object.

model-manager [class]

This is the top object instance in the AML object tree structure. It holds all the current models in the current AML session. It also holds global settings and options. Every AML session has only one model-manager instance that is automatically instantiated when AML is started.

Inheritance: model-manager-class, adaptive-class

Remarks: The user should not instantiate any instance of model-manager. The user can add a property to the model-manager to be used as global settings. The user can call the function model-manager in order to grab the instance of the model-manager class.

See Also: model-manager

current-model [Function]

This function returns the current active model.

Arguments:

- **instance**

  An instance of a model manager.

```
(create-model 'name-generator)
;; creates a name-generator model
(current-model (the superior))
;; returns the parent of the model
```

model-manager [Function]

Returns the model-manager object instance.

Example 2-19. Model-manager Example

```
(the interface forms (:from (model-manager)))
```

object-has-been-deleted? [Function]

This function takes an object instance and returns t if that object has been deleted or is no longer valid. It returns nil otherwise.

object-name [Function]

Given an object/property-object instance, this function returns the name of the instance. The name of an instance is always a symbol, not a string.

Arguments:

- **instance**

  An AML object.

Example 2-20. Using object-name

```
(let ((object-list (select-object :from (root-object) :class 'box-object))
      )
  (loop for object in object-list collect (object-name object)))
;; The statement above returns a list of names of all objects of class box-
object that exist under the current model
```

post-save-action [Method]

Defined on Classes:

aml-class

This method allows the user to run a command or a list of commands after an object of class 'aml-class' or that inherits from "aml-class".

Example 2-21. Using post-save-action

```
(in-package :AML)

(define-class S-CLASS
 :inherit-from (box-object)
 :properties (
          )
 )

(define-method POST-SAVE-ACTION S-CLASS ()
 (print "5 6 7 8 9 10")
 )
```

pre-save-action [Method]

Defined on Classes:

aml-class

This method allows the user to run a command or a list of commands before an object of class 'aml-class' or that inherits from "aml-class".

Example 2-22. Using pre-save-action

```
(in-package :AML)

(define-class S-CLASS
 :inherit-from (box-object)
 :properties (
          )
 )

(define-method PRE-SAVE-ACTION S-CLASS ()
 (print "0 1 2 3 4")
 )
```

print-tree [Function]

This function will print (to the specified output stream) the nodes of the model tree starting at the supplied instance. If a node has not been previously demanded it will not be expanded. This method may be used to determine the state of expansion of a model.

Arguments:

- **instance**

  An aml object instance specifying the root of the instance tree to be shown.

Keyword Arguments:

- **:indent**

  The number of spaces to indent at every level

  Default Value: 3

- **:level**

  The indentation level of the root branch

  Default Value: 1

- **:expand?**

  Specifies whether to expand branches that have not been demanded yet. When t, all branches are expanded.

  Default Value: nil

- **:stream**

  AML stream to send the output to (can be a file stream), default value is t which sends the output to the AML command line.

  Default Value: t

- **:show-class?**

  When non-nil, the class name of an instance is shown next to the instance name.

Default Value: nil

properties [Function]

Returns a list of the names of the properties of the specified object/property instance.

Arguments:

- **instance**

  An object-instance.

Keyword Arguments:

- **:dependencies?**

  When this keyword is not nil, a call to this function creates a dependency on the results of the function. I will illustrate with an example: If the formula of a certain property prop1 contained a call to "properties" on object1, then a new property is added to object1 or a property is deleted from object1 (at runtime), the value of prop1 is smashed.

  Default Value: nil

- **:ordered?**

  When this keyword is not nil, the property names are returned in the order they were listed in the class definition (source code). If properties were added to the object at runtime, they are returned at the end of the list in the order they were created.

  Default Value: nil

See Also: property-objects

Example 2-23. Using properties

```
(create-model 'table)
(properties (root-object)) ;; root-object is a function that returns an
instance to the current model
;; Returns '(table-height layer color position table-width thickness geom
objectlist orientation canonical-geom display? total-weight reference-object
line-width render table-depth coordinate-object simple-geom-info simple-geom
added-objects ...)
```

property? [Method]

Defined on Classes:

aml-class

Returns a non nil value when the property name given is a property of the instance given

Arguments:

- **instance**

  Instance of an AML object.

- **name**

Property name to check for.

Example 2-24. Using property?

```
(property? (the interface) 'font)
```

property-bound? [Function]

The function takes an object instance and a property name and returns t if the specified property is bound to a value, i.e. its value has been demanded. This function returns nil is the property is in an unbound state.

Arguments:

- **Instance**

  The object containing the property.

- **property-name**

  The name of the property to check, must be a symbol

See Also: smash-value

Example 2-25. Using property-bound?

```
(property-bound? (the object1) 'property-name)
;; Returns t if the property is bound to a value, nil if the property is
value unbound/smashed
```

property-objects [Method]

Defined on Classes:

aml-class

This method takes an object/property instance and returns a list of property instances that are properties of that object/property.

Arguments:

- **Instance**

  The class to find the property-objects of.

Keyword Arguments:

- **:class**

  The class every property returned must inherit from. Default value causes all properties to return.

  Default Value: t

See Also: properties

Example 2-26. Using property-objects

```
(property-objects (the interface)) ;; Returns the list of property instances
that are properties of "(the interface)"
```

```
(property-objects (the interface) :class 'simple-property-class) ;; Returns
the list of simple-property-class instances that are properties of "(the
interface)"
```

reparent-object [Function]

Changes the superior of an object instance by making it a subobject of the new parent instance specified.

Arguments:

- **instance**

  Object instance to reparent

- **parent-object**

  Object instance specifying new parent

This function should only be used by advanced AML developers. Reparenting an object may render some of its property formulas invalid when those formulas contain "the" references that can lead to an error when traced from the new object position.

series-next [Function]

This function can be called only on a member subobject of a series-object instance. It returns the value of a property from the next subobject in the series or it returns the next subobject instance itself. It causes an error if the instance's superior is not a series-object instance.

Arguments:

- **instance**

  An object instance whose superior inherits from series-object.

Optional Arguments:

- **property**

  The name of the desired property from the next object in the series. The default is 'self which will return the instance of the next object in the series.

- **final-value**

  This is the value that will be returned if the instance is the last subobject in the series. Default is the superior of the instance.

See Also: series-object, series-previous

Example 2-27. Using series-next

```
(series-next (the series-member-0002) 'volume) ;; Returns the volume of the
next object in the series
(series-next (the series-member-0002) 'self)   ;; Returns the instance of the
next object in the series
```

series-previous [Function]

This function can be called only on a member subobject of a series-object. It returns the value of a property from the previous subobject in the series or the previous subobject instance. It will cause an error if the instance's superior is not a series-object instance.

Arguments:

- **instance**

  An object instance whose superior inherits from series-object.

Optional Arguments:

- **property**

  The name of the desired property to return from the previous object in the series. The default is 'self which will return the instance of the previous object in the series.

- **initial-value**

  This is the value that will be returned if the instance is the first subobject in the series.

  Default Value: nil

See Also: series-next, series-object,

Example 2-28. Using series-previous

```
(series-previous (the series-member-0002) 'volume) ;; Returns the volume of
the previous object in the series
(series-previous (the series-member-0002) 'self)   ;; Returns the instance of
the previous object in the series
```

subobject? [Method]

Defined on Classes:

aml-class

This method queries an instance for the existence of a subobject given the subobject name. It returns a non nil value if the subobject exists.

Arguments:

- **instance**

  Instance of an AML object.

- **name**

  Subobject name to check for, must be a symbol

Example 2-29. Using subobject?

```
(subobject? (the interface) 'forms)
```

superior [Function]

The function returns the instance of the superior/parent of the given instance.

Arguments:

- **instance**

  The object instance to find the superior of.

tree-display [Method]

Defined on Classes:

aml-class

This method returns the name (symbol) that is displayed in the tree for a given class. This method must return a symbol. When a model tree is displayed, this method will be called on each object in the tree. This method can be overridden on user application classes to modify the tree display of instances of these classes. The default action is to return the object name as shown in the example below.

Arguments:

- **instance**

  An instance of an AML object.

Example 2-30. Using tree-display

```
(define-method tree-display aml-class ()
   (object-name self)
 )
```

undraw [Method]

Defined on Classes:

This method removes an object instance (or instance tree) from the active display window.

Arguments:

- **instance**

  The object instance to be undrawn

Keyword Arguments:

- **:subobjects?**

  When this keyword in t, all object instances in the tree under the instance specified are also undrawn.

  Default Value: t

- **:update-window?**

  When this keyword is nil, the active window is not automatically refreshed.

  Default Value: t

See Also: draw

unsaved-object [class]

This class allows the user to dynamically add objects to a model/object without saving these objects when the model/object is saved. When an AML model or object is saved, all instances of unsaved-object, as well as their properties and subobjects are not saved. In other words, if an instance of unsaved-object is added to the model, and the model is then saved, the added instance will not be part of the saved model.

Inheritance: object

See Also: aml-save

Example 2-31. Unsaved-object Example

```
(define-class unsaved-box-class
  :inherit-from(box-object unsaved-object)
    )
;; All instances of unsaved-box-class that are dynamically added to an AML
model will not be saved when the model is saved.
```

unsaved-subobjects [class]

This class allows the user to changes the values or formulas of the properties of the subobjects of a class without saving these changes when the AML model is saved.

Inheritance: object

See Also: unsaved-object, aml-save

Example 2-32. Unsaved-subobjects Example

```
(define-class my-series-class
  :inherit-from(series-object unsaved-subobjects)
  )

;; All instances of my-series-class will behave as follows: Even if the user
changes the values or formulas of the subobjects of this class, these changes
are not saved when the AML model/object is saved.
```

Language

Functions

aml-load [Function]

This is a function that allows users to retrieve a saved object from a file that was generated by a call to aml-save and return the instance of the loaded object on success. It returns nil if an error occured while loading the object/model.

Arguments:

- **:file-name**

  File path string where the model/object is saved. It must be a file that was generated by a call to aml-save. It can accept files with paths that have spaces.

Keyword Arguments:

- **:addto**

  Object instance to add the retrieved object to. This argument is ignored if the keyword as-model? is non-nil. Default value adds to the current model.

  Default Value: root-object

- **:as-model?**

  When non-nil, retrieves the saved object as a root model under the model-manager.

  Default Value: nil

- **:display-error-messages?**

  If set to t, and the keyword :enable-error-handling? key is set to t, errors generated during the load will result in error messaged displayed on the message pane.

  Default Value: nil

- **:log-file-name**

  Path string of the file that will conatin a log of errors that are generated when enable-error-handling? is set to t

  Default Value: nil

- **enable-error-handling?**

  Setting this key to t allows the user to retrieve models that may contain problems. A log of the errors encountered during the retrieval are stored in the log-file-name.

  Default Value: nil

- **:object-name**

  Name of the root node of the retrieved object/model. When nil, the name saved in the file is used.

  Default Value: nil

- **:print-messages-to-screen?**

  Prints the message to screen

  Default Value: t

- **:print-messages-to-message-pane?**

  When t, send messages to current message pane

  Default Value: t

- **:model-information?**

  If set to t then aml-load would return: 1. Branch Root object 2. User Name 3. Date model was saved 4. Time model was saved 5. Previous model version 6. Current model version 7. List of loaded modules 8. Description else aml-load would return branch root object.

- **:temp-directory**

  Path string of the temp directory used by the function. The function does not use anymore the "temp" environment variable.

  Default Value: (logical-path :temp)

See Also: aml-save

apropos [Function]

Apropos is a function that is used from the AML command line to check for the existence of a class, function or method when the user only knows a part of the name. apropos take on argument (a symbol) specifying the name or part of the name of the class, method or function in question.

Arguments:

- **Symbol**

  Any method,class or function name. Or any part of the name.

Example 2-33. Apropos example

```
(apropos 'button)
;; This call returns a descriptive listing all classes, methods and functions
  that have "button" in their name.
```

auto-naming? [Function]

Returns the value of the auto-naming? property from the root object. If the auto-naming? property is not present, it will be added and defaulted to t.

Example 2-34. Example

```
(auto-naming?)
t
(the auto-naming?)
t
(change-value (the auto-naming?) nil)
nil
(auto-naming?)
nil
(the auto-naming?)
nil
```

change-formula [Function]

Change-formula changes the evaluation formula for the specified property to the specified expression.

Arguments:

- **property-reference**

  The 'the' reference to the specified property.

- **new-formula**

The new formula of the property. The formula should be quoted so that it is evaluated in the context of the property and used as a formula. If the formula is not quoted, it will be evaluated prior to being place in the formula slot.

Example 2-35. Change Formula

```
(create-model 'table)
;; creates a table model
(the legs leg-0000 diameter)
;; returns the initial value of diameter : 1.0
(the legs leg-0000 height)
;; returns the initial value of height 35.0
(change-formula (the legs leg-0000 height) '(* 30.0 (the diameter)))
;; changes the formula for the height to : (* 30.0 (THE DIAMETER))
(the legs leg-0000 height)
;; returns the new height value : 30.0
(change-value (the legs leg-0000 diameter) 1.2)
;; changes the value of the diameter 1.2
(the legs leg-0000 height)
;;returns the new height value 36.0
```

See Also: change-value

change-value [Function]

Change-value changes the current value for the specified property to the specified new value.

Arguments:

- **property-reference**

  The 'the' reference to the specified property.

- **new-value**

  The new value of the property.

Example 2-36. Using Change Value

```
(create-model 'table)
;; creates a table model
(the thickness)
;; returns the initial value of thickness : 1.0
(the legs leg-0000 diameter)
;; returns the initial value of diameter : 1.0
(the legs leg-0000 height)
;; returns the initial value of height 35.0
(change-value (the thickness) 1.5)
;; changed the thickness value to 1.5
(the thickness)
;; returns 1.5
(the legs leg-0000 diameter)
;; returns the new value of diameter based on formula 1.5
```

```
(the legs leg-0000 height)
;; returns the new height value 34.5
```

See Also: change-formula

classes-for-method [Function]

Classes-for-method will return the class names that have a method of the given name defined.

Arguments:

- **method-name**

  The name of a method. The method-name must be a symbol.

Example 2-37. Identifying class for method

```
(classes-for-method 'delete-object)
;; returns the list with classes : (DELETE-EVENT OBJECT-CLASS)
(classes-for-method 'center-of-edge)
;; returns the list with classes : (GRAPHIC-OBJECT PLOT-FRAME PLOT-INFO-BOX
SIMPLE-GEOMETRY-CLASS)
```

create-model [Function]

Create-model allows the user to create a new model based on the specified class.

Arguments:

- **name**

  The name for the model being created. If the class argument is not supplied, the name must be a valid class name.

Keyword Arguments:

- **:class**

  A class defined using define-class.

  Default Value: name of model

- **:deletable?**

  This argument specifies whether the model being created will be deletable using the delete-model and delete-all-models commands. The default value will allow the model to be deleted.

  Default Value: t

- **:init-form**

  A list of property/formula pairs which overrides the values of the properties specified in the class definition.

Example 2-38. Using create-model

```
(create-model 'table)
```

```
;; creates a new table model
(print-tree (the))
;; returns the initial tree of the table:
   TABLE
        (origin unexpanded)
        (top unexpanded)
        (legs unexpanded)
(the table)
;; returns the instance of the table
(create-model 'new-table :class 'table)
;; creates a new table based on the table class
(print-tree (the))
;; returns the initial tree of the new tanle:
   NEW-TABLE
        (origin unexpanded)
        (top unexpanded)
        (legs unexpanded)
(the new-table)
;; returns the instance of a new table based on the same class
(create-model 'box-object :init-form '(height (* 2.0 ^width) width 3.2 depth
(/ ^height 1.23) color 'blue))
```

create-object [Function]

Create-object allows the user to make an instance of a specified object type. This instance will not be in the model system. In order to reference the instance, it will have to be bound to a variable or property. The use of this function is highly discouraged. The functions create-model and add-object will create objects within the AML tree structure.

Arguments:

- **class**

  A class defined by using define-class.

Example 2-39. Using create-object

```
(create-object 'table)
;; creates a table object
```

See Also: create model, add-object

current-object [Function]

Returns the current object for the calling location. This would be where the tracing would begin.

Example 2-40. Using current-object

```
(create-model 'table)
;; creates a new table model
(current-object)
;; returns reference to the table
```

```
(delete-all-models t)
;; deletes all active models
(current-object)
;; referes to the model manager because there are no models underneath
```

default [Function]

When specified as the formula for a property, default will look up the tree for a property with the same name. If one is found, the value of that property is used. Otherwise, the specified default-formula is used.

Optional Arguments:

- **default-formula**

  If the search up the tree for a property with the same name is unsuccessful, the formula specified here will be used. If no formula is specified, the default will be a pop-up-typein.

Example 2-41. Using default

```
(define-class DEFAULT-TEST
     :inherit-from (box-object)
     :properties (
                height (* 2.0 ^depth)
                width (* 3.0 ^depth)
                depth 1.0
                 )
     :subobjects (
                (cylinder :class 'cylinder-object
                height (default)
                diameter (default 0.5)
                )
        )
   )

(create-model 'default-test)
;; creates a new model called default-tree
(the height)
;; returns the initianl 2.0
(the cylinder height)
;; returns the initial cylinder height(defaulted to ) - 2.0
(the cylinder diameter)
;; returns : 0.5
(change-value (the depth) 2.0)
;; changed the value to : 2.0
(the cylinder height)
;; returns the new value of height : 4.0
```

delete-all-models [Function]

Delete all models from the model system. The action requires confirmation before deleting.

Optional Arguments:

- **force?**

  When a value of t is supplied for this argument, all models will be deleted without a confirmation. When a value of nil is supplied, the user will be asked to confirm the deletion.

  Default Value: nil

Example 2-42. Using delete-all-models

```
(create-model `table)
;; creates a new table model
(current-object)
;; returns an instance of the table
(delete-all-models)
;; deletes all models
(current-object)
;; returns reference to the model-manager because there are no current models
```

See Also: delete-model

delete-model [Function]

Delete-model removes the specified model from the model system.Deletes a method from the current session.

Arguments:

- **method-name**

  The name of top (root) object for the model to be deleted.

Example 2-43. Deleting model

```
(create-model 'box-object)
;; creates a box-object model
(the box-object)
;; returns the instance of the object
(delete-model 'box-object)
;; deletes the box-object
(the box-object (:error nil))
;; returns nil for instead of an error
```

See Also: delete-all-models

find-the-trace [Function]

This function returns the exact the reference path to the supplied instance starting at the object provided by the from keyword. This function guaranties that the path returned is the exact the reference to the supplied instance starting at the instance provided by the keyword from; i.e., "the" will not perform any search when evaluated.

Arguments:

- **instance**

  An object instance. When instance is nil, the function always returns '(the)'.

Keyword Arguments:

- **:from**

  An object instance to start "the" tracing from. If from is not a superior to instance, the tree trace returned will include the exact number of superior statements needed to reach the closest common superior. When from is nil, the tree path returned will always start by "(the model-manager ...)".

  Default Value: nil

Example 2-44. Using find-the-trace

```
(find-the-trace (the interface) :from (the interface forms))
;; returns : (THE SUPERIOR)
(create-model 'box-object)
;; creates box-object model
(find-the-trace (the box-object width self) :from (the box-object height
self))
;; returns (THE SUPERIOR WIDTH)
```

find-tree [Function]

The find-tree method returns a tree trace to the supplied instance in the form of a the reference statement.

Arguments:

- **instance**

  An object instance.

Keyword Arguments:

- **:from**

  An object instance to start "the" tracing from. If from is not a superior to instance, the tree trace returned will start at the closest common superior. When from is nil, the tree trace returned will always start by "(the model-manager ...)".

  Default Value: nil

Example 2-45. Using find-tree

```
(create-model 'table)
```

```
;;creates a teble model
(find-tree (the))
;; returns (THE TABLE)
(find-tree (the table top))
;; returns the new top - (THE TABLE TOP)
(find-tree (the top))
;; returns : (THE TABLE TOP)
(find-tree (the top) :from (the table))
;; returns (THE TOP)
```

See Also: find-the-trace

get-object-class-formula [Function]

This function returns the given class formula of self when it is added.

Arguments:

- **self**

  It is the instance of an added object (property).

ignore-current-dependency [Function]

This is a wrapper function to allow the dependency mechanism to be shut off.

Arguments:

- **body**

  The body of code to perform without dependencies being set to the current object.

Example 2-46. Using ignore-current-dependency

```
>(define-class MY-BLOCK
   :inherit-from (box-object)
   :properties (
                height 10.0
                width 5.0
                depth (+ ^height (ignore-current-dependency ^width))
        )
  )
(create-model 'my-block)
;;creates ne my-block model
(the height)
;; returns the initial height : 10.0
(the width)
;; returns the initial width : 5.0
(the depth)
;; returns the initial depth :15.0
(i-depend-on (the depth (:eval? nil)))
;; returns what depth depends on :
(THE MY-BLOCK DEPTH)
(THE MY-BLOCK HEIGHT)
(change-value (the height) 5.0)
```

```
;; changed the value of height to 5.0
(the height)
;; returns 5.0
(the depth)
;; returns 10.0
(change-value (the width) 10.0)
;; changes the width to 10.0
(the width)
;; returns 10.0
(the depth)
;; returns 10.0 - not changed because of the definition
(change-value (the height) 10.0)
;; changes to 10.0
(the height)
;; returns 10.0
(the depth)
;; returns the new value 20.0
```

inheritance-list [Function]

The inheritance-list method will retrieve a list of all the inherited classes of a class or object instance in the order of precedence (i.e., the order of the list is the order the system uses to decide which method to call).

Arguments:

- **class-or-instance**

  An object instance of a class or the name of a class.

Example 2-47. Using inheritance-list

```
;; using loop for printing the list of classes inherited from table
(loop for class in (inheritance-list 'table) do (print class))
TABLE
ASSEMBLY-OBJECT
SIMPLE-GEOMETRY-CLASS
GRAPHIC-OBJECT
POSITION-OBJECT
OBJECT
ADAPTIVE-CLASS
OBJECT-CLASS
AML-CLASS
STANDARD-OBJECT
T
NIL
```

list-models [Function]

Returns a list of the models that are attached to the model manager.

Example 2-48. Using list-models

```
(list-models)
;; there no created models so returns just : (INTERFACE)
(create-model 'table)
;; creates a table model
(list-models)
;; returns the list : (INTERFACE TABLE)
(create-model 'box-object)
;; creates a box-object
(list-models)
;; returns the list : (INTERFACE BOX-OBJECT TABLE)
```

See Also: create-model, delete-all-models, delete-model

methods-for-class [Function]

The methods-for-class method will return all of the methods that have been defined for the given class name or instance.

Arguments:

- **class-or-instance**

  An object instance of a class or the name of a class.

See Also: classes-for-method

object-instance [Function]

Used to find the instance of a subobject or property in the specified object. If the object does not have a subobject/property of the given name, nil will be returned.

Arguments:

- **instance**

  An object

- **name**

  The name of the subobject/property to find

Example 2-49. Using object-instance

```
(create-model 'table)
;; creates a table model
(object-instance (the) 'leg)
;; returns : NIL
(object-instance (the) 'legs)
;; returns an instance of object legs
(object-instance (the) 'total-weight)
;; returns an instance of property object weigth
```

property-class [class]

The property-class is the class that all objects (referred to as properties or property objects) that feature will a formula and a value inherit from. Note: Subclasses of property-class may not be used as subobjects.

Inheritance: aml-class

root-object [Function]

Root-object returns the top (root) object instance of the current (active) model.

Example 2-50. Using root-object

```
(create-model 'table)
;; creates a table model
(root-object)
;; returns an instance of the table object
(the object-name (:from (root-object)))
;; returns the acutal name : TABLE
```

saved-model-name-class [Function]

Takes a saved model file path string and returns a list whose first element is the saved model name and the second element is the saved model class.

Example 2-51. Using save-model-name-class

```
(saved-model-name-class "/usr/local/boxes.mdl")
;; returns (BOX BOX-OBJECT)
```

saved-model-and-object-names [Function]

Takes a saved model file path string and returns a list containing the following information: The first element of the returned list is a list containing the model name and the model class respectively. The other elements of the returned list are a list each containing the object name and object class (respectively) of every object in the saved hierarchy tree.

Example 2-52. Using saved-model-and-object-names

```
(create-model 'name-generator)
;; creates a name-generator model
(add-object (the) 'box 'box-object)
;; adds box to the name-generator
(add-object (the) 'cylinder 'cylinder-object)
;; adds cylinder to the name-generator
(aml-save (the) "/usr/local/model.mdl" :as-model? t)
;; returns : "/usr/local/model.mdl"
(saved-model-and-object-names "/usr/local/model.mdl")
```

```
;; returns ((NAME-GENERATOR NAME-GENERATOR) ((BOX BOX-OBJECT) (CYLINDER
CYLINDEROBJECT)))
```

select-model [Function]

Select-model selects a model from the model system to be the current active model.

Arguments:

- **model-name**

  The name of the model to be selected as the current active model.

Example 2-53. Using select-model

```
(delete-all-models t)
;; returns - NIL
(select-model 'table)
;; returns - NIL
(the)
;; references to the model-manager
(create-model 'table)
;; creates a table model
(select-model 'table)
;; selects the table
(the)
;; refers the table instance
```

select-object [Function]

Select-object allows the user to query the object instance tree for instances that satisfy certain criteria. Select-object may be used to return the object instances or other instance information. This function returns the list of selected objects or a list based on the eval keyword described below.

Note: Select-object only traverses the objects stored as subobjects. If an object is stored as a property, or as a sub-object of a property, it will not be encountered on the select-object search path. For properties, refer to select-property below.

Keyword Arguments:

- **:from**

  The object where the search starts, this object is also returned in the list of it passes the user criteria

- **:class**

  The class name that an object must inherit from in order to be selected. This keyword can also take a list of classes: Objects that inherit from any of these classes will be selected.

- **:test**

  The test an object must pass in order to be selected. The test is a quoted AML expression that evaluates to t or nil. In order to refer to the current object being tested within the test expression, "(the)" can be used (see example below). The default test selects all instances in the instance tree.

  Default Value: t

- **:expand?**

  A quoted AML expression that will be evaluated at every object of the tree to determine if the tree branch under that object should be expanded, if not already expanded. The default will force the entire tree to expand.

  Default Value: t

- **:eval**

  A quoted expression that determines the return values of the function. This expression is called on every selected object, after the test, class and branch critieria have passed on that object. The values returned will form the elements of the list returned by select-object. The default will return the object instance. Example: (select-object :from (root-object) :eval '(object-name (the))) will return a list of the object names of the selected objects instead of their corresponding object instances.

  Default Value: '(the)

- **:branch?**

  A quoted AML expression that will be evaluated at each node of the tree to determine whether to look down that branch of the tree. It should be an expresstion that evaluates to t or nil. The default will look down all branches.

  Default Value: t

- **:dependencies?**

  If the select-object is called from a property, this determines whether or not to set dependencies on the added objects of the objects that were encountered by the selection. This keyword should be t or nil.

Example 2-54. Using select-object

```
(create-model 'table)
;; creates table model
(select-object :expand? nil)
;; returns all objects under the root-object, except for objects that have
not been referenced yet.
(select-object :expand? t)
;; returns all objects under the root-object. It also expands and returns
objects that have not been referenced yet.
(select-object :class 'table-leg)
;; returns all objects under the root-object that are of class table-leg or
of any other class that inherits from table-leg.
(select-object :from (the legs))
;; returns all objects that are located under "(the legs)". It does not start
the search at the root-object like in the above calls.
(select-object :test '(object-instance (the) 'height))
;; returns all objects under the root-object that contain a property named
height.
(select-object :test '(object-instance (the) 'height) :eval '(the height))
;; evaluates the height of all objects under the root-object that contain it
as a property.
;; It returns a list of the height value of each. : (1.0 35.0 35.0 35.0 35.0)
(select-object :branch? '(not (typep (the) 'series-object)))
```

```
;; does not search branches of the model-tree unless the object at that
branch inherits from 'series-object.
(select-object :from (root-object) :eval '(object-name (the)))
;; returns the names (symbols) of all objects under root-object
```

select-property [Function]

Select-property allows the user to query the model tree for property instances or property related information. It returns a list whose elements have passed the user criteria. The values returned in the list depend on the eval keyword described below.

Remarks: The search traverses all objects and properties of the model tree; however, when a property is encountered, it is considered a leaf node and properties/subobjects under it are not searched.

Keyword Arguments:

- **:from**

  The top object where the search process starts.

- **:object-class**

  The class an object must inherit from in order for its properties to be selectable.

- **:object-test**

  Quoted expression to be evaluated at every object of the tree. The expression must return t in order for the properties of the object in question to be considered. Default value is to allow all objects.

  Default Value: t

- **:class**

  The class a property must inherit from in order to be selected.

- **:test**

  The test a property object must pass in order to be selected. It should be a quoted expression that evaluates to t or nil. The default selects all.

  Default Value: t

- **:eval**

  A quoted expression that will be evaluated at each selected property in the tree. This expression is evaluated on properties that have already passed all other tests. The values returned are the elements of the list returned by select-property. Example: (select-property :from (root-object) :eval '(the)) will return a list of values of all selected properties, while (select-property :from (root-object) :eval '(the self)) will return a list of instances of all selected properties.

  Default Value: '(the)

- **:branch?**

  A quoted expression that will be evaluated at each node of the tree to determine whether to look down that branch of the tree. It should be an expresstion that evaluates to t or nil. The default will look down all branches.

  Default Value: t

- **:ordered?**

  When t, properties are returned in the order they were defined in the class definition.

Example : The usage of keywords is similar to their usage in the function select-object. Please refer to the select-object examples.

smash-value [Function]

Smash-value unbounds (smashes) the specified property value causing it to be recalculated the next time it is demanded. When smash-value is called on a object instance, and the instance's creation from the object being defined as a subobject within a class, smash-value causes this instance to "smash", which means it is deleted until the next time it is demanded by a "the" reference.

Arguments:

- **property-reference**

  The the reference to the specified property whose value is to be smashed.

Example 2-55. Using smash-value

```
(create-model 'table)
;; creates a table model
(the top height)
;; returns initial value of height : 1.0
(change-value (the top height) 1.5)
;; changes the value of height to 1.5
(the top height)
;; returns the value of height : 1.5
(smash-value (the top height))
;; smashes the given value of height
(the top height)
;; returns the initial value of height : 1.0
```

subobjects [Function]

Returns a list of the names of the subobjects for the given object.

Arguments:

- **instance**

  An object instance.

Keyword Arguments:

- **:dependencies?**

  If the method is called from a property, this specifies whether or not to smash the value when the number of children changes. A value of t will cause dependencies to be set to the children.

Example 2-56. Using subobjects

```
(create-model 'table)
;; creates a table model
```

```
(subobjects (the))
;; returns the list of subobjects of table : (ORIGIN TOP LEGS)
(subobjects (the legs))
;; returns the list of subobjects of (the legs) : (LEG-0000 LEG-0001 LEG-0002
LEG-0003)
(subobjects (the legs leg-0000))
;; returns the subobjects of one of the legs - NIL
```

trace-from [Function]

Trace-from temporarily overrides the standard the referencing behavior. Within the body of trace-from, the referencing is set to start from the instance supplied.

Arguments:

- **object**

  An instance or expression that returns an instance to start the referencing from within the body.

Optional Arguments:

- **body**

  The code to be executed while (the) is the instance passed. The body can be more then one statement.

Example 2-57. Using trace-from

```
(create-model 'table)
;; creates a new table model
(trace-from (the table top) (the color))
;; returns current color of the top - RED
;; The trace-from statement above is equivalent to (the color (:from
(the table-top)))
```

Chapter 3. Non Geometric Classes

Property Classes

Property objects are entities that provide properties with the ability to contain more information than just a value. This is accomplished by creating properties as instances of the class property-object. When a property is created using a property/formula pair specification, an instance of property-object is created. The formula of the property gets set to the supplied formula and the value of the property is assigned by evaluating the formula when the property is demanded. Because properties are class instances, methods may be written to modify or enhance their behavior

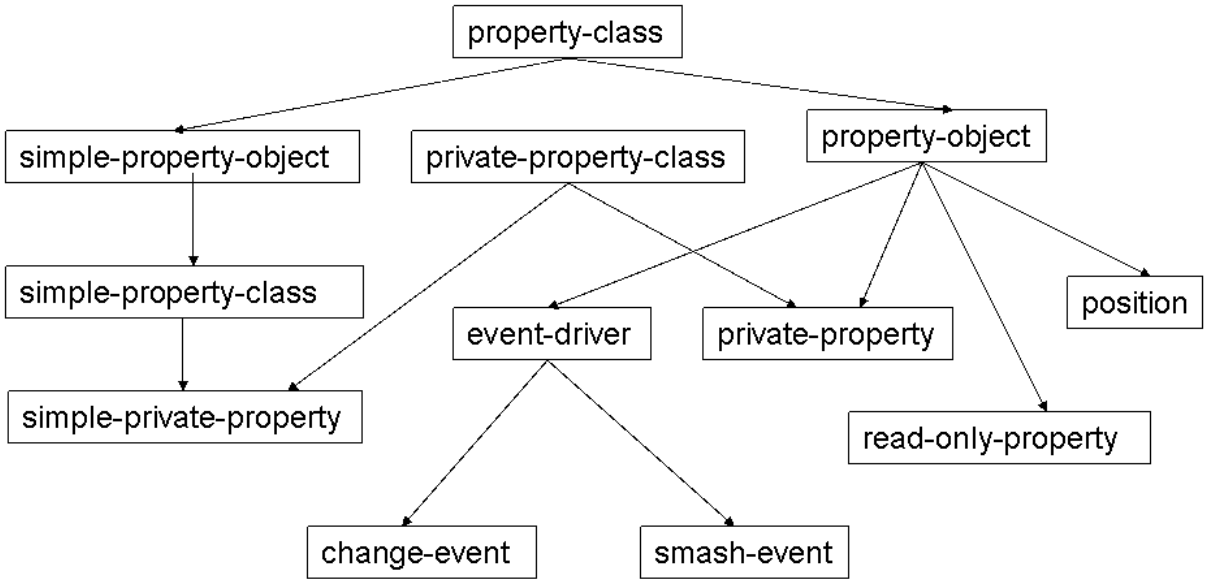
Creating Property Classes

Creating new property classes is the same as creating any other classes, using define-class.Classes that will be used as properties should inherit from property-object. Adding properties with default formulas to properties is done in the class definition. The formula and value may be specified as properties, but if the value is supplied the formula will be ignored until the value gets smashed and demanded again. When a value is supplied and the formula is ignored this also means that no dependencies are established and the only way the property will be smashed is through a call to smash-property-value. If only the formula is supplied the value will be calculated using the formula.

Note: Classes that inherit from property-object may not be used as subobjects. However, any defined class may be used as a property. If a class does not inherit from property-object and it is used as the class in a property specification there will not be a formula or a value property for that property. In this case, references directly to that property will return the instance of the property object. select-object will not select those instances since they are not subobjects.

For an example on defining properties, see the example "Defining and Using Properties" with the description of property-object

Figure 3-1. Property Classes Inheritance Tree.



Functions, Methods and Classes

property-class? [Function]

Takes a class name or an object and returns t if it inherits from property-object, nil otherwise.

Arguments:

- **thing**

  An class name or an instance

Example 3-1. Using property-class?

```
(create-model 'table)
;; creates new table model
(property-class? (the legs (:eval? nil)))
;; returns NIL
(property-class? (the table-height (:eval? nil)))
;; returns T
(property-class? 'property-object)
```

```
;; returns T
(property-class? 'box-object)
;; returns NIL
```

get-formula [Method]

Defined on Classes:

property-class

The get-formula method returns the formula of a given property.

Arguments:

- **instance**

  The property instance to get the formula from.

```
;;Assume class table is instantiated above
> (create-model 'table)
;; creates the model
> (get-formula (the top height (:eval? nil)))
;; returns the formula for the height : (THE THICKNESS)
> (the top height)
;; returns the value of the height 1.0
> (change-formula (the top height (:eval? nil)) '(* 2.0 (the thickness)))
;; returns the new formula for the height : (* 2.0 (THE THICKNESS))
> (the top height)
;; evaluates and returns the new height value 2.0
> (get-formula (the top height (:eval? nil)))
;; returns the new formula (* 2.0 (THE THICKNESS))
```

simple-property-class [class]

Simple-property-classes are properties that do not set up a dependency network. It is the same as simple-property-objec that was defined in AML versions preceding AML version 4.0.

Inheritance: property-class

change-property-formula [Method]

Defined on Classes:

simple-property-class

Change-property-formula is the method invoked by change-formula to cause a property's formula to be changed to the specified formula.

Arguments:

- **instance**

  An instance of a simple-property-object whose formula will be changed. This is most commonly an instance of property-object which inherits from simple-property-object. (see below)

- **new-formula**

The new formula for the property. The formula should be quoted so that it is evaluated in the context of the property and used as a formula. If the formula is not quoted, it will be evaluated prior to being placed in the formula slot.

Optional Arguments:

- **user-set?**

  If this value is true it will notify the model that the new formula needs to be saved when the model is saved.

  Default Value: nil

Remark: Change-property-formula should only be used by advanced users that completely understand the need for this functionality. Most of the time change-formula should be used instead.

```
;;assume table is Instantiated above
(create-model 'table)
 (the legs leg-0000 diameter)
;; returns the original leg diameter1.0
(the legs leg-0000 height)
;; returns the original leg height35.0
(change-property-formula (the legs leg-0000 height (:eval? nil)) '(* 30.0
(the diameter)))
;; changes the formula for legs height to : (* 30.0 (THE DIAMETER))
(the legs leg-0000 height)
;; returns the new height 30.0
(change-value (the legs leg-0000 diameter) 1.2)
;;changes the diameter and returns 1.2
(the legs leg-0000 height)
;; returns the new height based on new diameter 36.0
```

See Also: change-formula

change-property-value [Function]

Change-property-value is the function invoked by change-value to cause a property's value to be changed to the specified value. If the current value of the property is the same as the new value, nil will be returned. Otherwise the value that the property is changed to will be returned.

Arguments:

- **instance**

  An instance of a simple-property-object whose value will be changed.This is most commonly an instance of property-object which inherits from simple-property-object. (see below)

- **new-value**

  The new value for the property.

Optional Arguments:

- **user-set?**

  When t, causes the new value to be saved when the object it belongs to is saved.

Default Value: t

Keyword Arguments:

- **:force?**

  Optional keyword. When t, the existing property value is forced to smash then change to the new value specified, even if the new value is equal to the existing value. This will force triggering the dependencies.

  Default Value: nil

This function accepts both optional and keyword argument, here is the format required: (change-property-value instance new-value [user-set? t] [:force? nil]). Remark: Change-property-value should only be used by advanced users that completely understand the need for this functionality. Most of the time change-value should be used instead.

```
;;Assume class table is instantiated above
(create-model 'table)
;; creates the mode of the table
(the thickness)
;; returns the original value of thickness : 1.0
(the legs leg-0000 diameter)
;; returns the original diameter : 1.0
(the legs leg-0000 height)
;; returns the original height : 35.0
(change-property-value (the thickness (:eval? nil)) 1.5)
;; changes the thickness value to : 1.5
(the thickness)
;; returns the new valie of thickness : 1.5
(the legs leg-0000 diameter)
;; returns the new value of diameter : 1.5
(the legs leg-0000 height)
;; returns the new value of the height : 34.5
```

See Also: change-value

formula-reference-property-class [class]

This is a special property class that is used for the following purpose: If the user wants the formula of a property A to simply be equal to the formula of another property B at all time, property A needs to be of class (or inherit from) formula-reference-property-class.

Inheritance: property-object

The "local" formula of a formula-reference-property-class instance is ignored.

unsaved-simple-property [class]

An unsaved-simple-property behaves like an unsaved-property, and it also inherits the behavior of simple-property-object. This class allows the user to change the value or the formula of a simple-property-object (at runtime) without saving these changes when the AML model/object is saved. When

an AML model or object is saved, all changes to the value or formula of instances of unsaved-simple-property are not saved.

Inheritance: simple-property-class

See Also: aml-save, unsaved-property

property-object [class]

Default properties in the Adaptive Modelling Language are created as instances of property-object. Property-objects set up a dependency network through their formulas.

Inheritance: property-class, adaptive-class

Remark: Even though this class has two properties "formula" and "value", they are actually special entities. The referencing from the formula and value of a property-object starts at the property-object and not at the formula/value. The user should never access the properties directly. To get the value of the property, call the the following function (the (:from <property-instance>)) or (the ... <property name>). To get the formula of the property, call the function (get-formula <property-instance>). Any other way of getting these properties may result in dependencies not being set properly. Notice the difference between the formula and the dummy-prop referencing in the following example.

Example 3-2. Defining and Using Properties

```
(in-package :aml)

;;---------------------------------
;; 1. Defining properties in a class

(define-class PROPERTY-EXAMPLE
  :inherit-from (box-object)
  :properties (
               height 10.0
               ;; This is the shortcut form, it is equivalent to setting the
formula to 10.0

               (width :class 'property-object
                      formula 10.0)

               (depth :class 'property-object
                      formula (+ (the height) (the width))
                      value 10.0)

               (block :class 'box-object)

               )
  )

;; Try the following at the AML prompt:
(create-model 'property-example)
;; This will create a property-example model
the height)
;; Returns 10.0
(the width)
```

```
;; Returns 10.0
(the depth)
;; Returns 10.0
(the block)
;; Returns a box-object instance
(smash-value (the depth))
(the depth)
;; Returns 20.0

;;-------------------------
;; 2. Output Units example:

;; A simple class to hold units and a print string that could be used in a
report.
(define-class PRETTY-UNITS
  :inherit-from (property-object)
  :properties (
               units (default 'inches)
               pretty-print (format nil "~a ~a"
                                    (get-value (the superior))
                                    !units)
               )
  )

;; A class that will be used to test the pretty-units.
(define-class WIDGET
  :inherit-from (box-object)
  :properties (
               units 'inches
               (size :class 'property-object
                     formula 1.0
                     value 10.0)
               (height :class 'pretty-units
                       formula !size)
               )
  )


;; Try the following at the AML prompt:
(create-model 'widget)
(the size)
;; Returns 10.0
(the units)
;; Returns 'INCHES
(the height pretty-print)
;; Returns "10.0 INCHES"
(change-value (the units) 'cm)
(the height pretty-print)
;; Returns "10.0 CM"
(smash-value (the size))
;; Causes the value to smash, which means the next time the property is
referenced, the formula will be calculated.
(the height pretty-print)
```

```
;; Returns "1.0 CM"



;;-------------------------
;; 3. Range Checking Property Example:


(in-package :AML)
(define-class RANGE-PROPERTY
  :inherit-from (property-object)
  :properties (
               formula 1.0
               min 0.0
               max 10.0
               )
  )

(define-method IN-RANGE? RANGE-PROPERTY (NEW-VALUE)
  ;; This method will return a t for a new-value within the min-max
  ;; range and nil for a value that is "out of bounds".
  (let ((inrange? t))
    (when !min; If min set, is new-value above min?
      (setq inrange? (<= !min new-value)))
    (when (and inrange? !max); If max set, is new-value below max?
      (setq inrange? (<= new-value !max)))
    inrange?); return answer
  )
(define-method OUT-OF-RANGE-ACTION RANGE-PROPERTY (new-value)
  ;; This method can be used to change the behaviour of a property
  ;; when a value that is out of range is encountered.
  (cond ((and !min (>= !min new-value))
         ;; if the value is too small use the min
         (format t "Warning: ~a is below ~a minimum. Setting to ~a"
                 new-value (find-tree self :from (root-object)) !min)
         !min)
        ((and !max (>= new-value !max))
         ;; if the value is too large use the max
         (format t "Warning: ~a is above ~a maximum. Setting to ~a"
                 new-value (find-tree self :from (root-object)) !max)
         !max)
        )
  )
(define-method CHANGE-PROPERTY-VALUE RANGE-PROPERTY (new-value
                                                     &optional user-set? &key
force?)
  (if (in-range? self NEW-VALUE)
      ;; if new-value is within range, call change-property-value
      ;; method for property-object with same arguments.
      (call-next-method)
    ;; if new-value is out of range call change-property-value for
    ;; property-object with new arguments.
    (call-next-method self (out-of-range-action self NEW-VALUE)
```

```
                    user-set?))
  )
(define-class BRICK
  :inherit-from (box-object)
  :properties (
               (height :class 'RANGE-PROPERTY
                       Property Objects
                       formula 5.0
                       min 2.0
                       )
               (width :class 'RANGE-PROPERTY
                      formula 15.0
                      min 10.0
                      max 20.0
                      )
               (depth :class 'RANGE-PROPERTY
                      formula 10.0
                      min 1.0
                      max 15.0
                      )
               )
  )

;;Try the following at the AML [rompt:
(create-model 'brick)
;; creates a brick model
(the height)
;;Returns 5.0
(the width)
;; Returns 15.0
(the depth)
;; Returns 10.0
(change-value (the height) -1.0)
Warning: -1.0 is below (THE HEIGHT) minimum. Setting to 2.0
;; Returns 2.0
(the height)
;; Returns 2.0
(change-value (the height min) 7.0)
;; Returns ;; 7.0
(the height)
Warning: 5.0 is below (THE HEIGHT) minimum. Setting to 7.0
;; Returns 7.0
(smash-value (the height min))
;; Returns NIL
(the height)
;; Returns 5.0
```

Example 3-3. The-referencing from a property formula

```
(in-package :aml)

(define-class cube
```

```
:inherit-from(box-object)
:properties(b
            depth 5
            (width :class 'property-object
                   formula ^depth
                   dummy-prop ^^depth)
            (height :Class 'property-object
                    formula ^depth)
            )
)
```

unsaved-property [class]

This class allows the user to change the value or the formula of a property (at runtime) without saving these changes when the AML model/object is saved. When an AML model or object is saved, all changes to the value or formula of instances of unsaved-property are not saved.

Inheritance: property-object

See Also: aml-save

private-property [class]

The private-property is used to create properties that will typically be hidden from view (i.e., will not be displayed by functions such as describe).

Inheritance: property-object private-property-class

saved-file-data-property-mixin [class]

This class allows the user to save file data in the model file (i.e. geoms, meshes, ...).

Inheritance: saved-property

Properties:

- **save-method-name**

  The method called to save the data. The first argument after the instance is file-name which is generated automatically.

  Default Formula: nil

- **retrieve-method-name**

  The method called to retrieve the data. The first argument after the instance is file-name which is generated automatically.

  Default Formula: nil

- **save-arguments-list**

  Default Formula: nil

- **retrive-arguments-list**

  Default Formula: nil

- **save-retrieve-object**

  The object the method used to call the object.

Default Formula: (the superior self)

Example 3-4. Example for saved-file-data-property-mixin

```
                        (define-class DATA-PROPERTY-CLASS
 :inherit-from (saved-file-data-property-mixin)
 :properties (
          save-method-name 'save-data
          retrieve-method-name 'retrieve-data
          save-arguments-list (list (the superior value))
          )
 )

(define-method SAVE-DATA DATA-PROPERTY-CLASS (file-name val)
 (when (and file-name val)
   (with-open-file (stream file-name
               :direction :output
               :if-exists :supersede
               )
     (format stream "~a" val)
     )
   )
 )

(define-method RETRIEVE-DATA DATA-PROPERTY-CLASS (file-name)
 (when (probe-file file-name)
   (let (
     (v (with-open-file (stream file-name
                   :direction-input)
          (read-from-string (read-line stream))
          )
        )
     )
     (delete-file file-name)
     v
   )
   )
 )
```

simple-private-property [class]

The simple-private-property is a property that is typically hidden from view and does not set up a dependency network.

Inheritance: private-property-class simple-property-class

simple-private-unsaved-property [class]

A simple-private-unsaved-property behaves like an unsaved-simple-property, and it also inherits the behavior of simple-private-property. This class allows the user to change the value or the formula of a simple-private-property (at runtime) without saving these changes when the AML model/object is saved.

When an AML model or object is saved, all changes to the value or formula of instances of simple-private-unsaved-property are not saved.

Inheritance: simple-private-property, unsaved-simple-property

See Also: unsaved-property, aml-save

read-only-property [class]

The read-only-property is a property whose formula and value are set in the object definition and can not be modified using change-value or change-formula. The methods change-property-formula and change-property-value are written on this class to override the behavior inherited from property-object.

Inheritance: property-object

simple-property-object [class]

Simple-property-objects are properties that do not set up a dependency network.

Inheritance: property-class

Event Classes

Functions, Methods and Classes

event-driver [class]

The event-driver is the super-class of property objects that have associated events. This class should not be instantiated by the user.

Inheritance: property-object

create-event [class]

This causes an event to happen immediately after the object is created.

Inheritance: event-object

Properties:

- **after-create**

  The event that should occur immediately after the object is created.

  Default Formula: nil

change-event [class]

The change-event property supplies the ability to cause something to happen when the property value is changed.

Inheritance: event-driver

Properties:

- **after-change**

The formula of this property is used to cause an event to occur immediately after the parent property's value has been changed.

Default Formula: nil

before-change

The formula of this property is used to cause an event to occur just before the parent property's value is changed.

Default Formula: nil

smash-event [class]

The smash-event property supplies the ability to cause something to happen when the property value is smashed.

Inheritance: event-driver

Properties:

- **after-smash**

  The formula of this property is used to cause an event to occur immediately after the parent property's value has been smashed. The expression specified in the "after-smash" property is triggered only after all properties that the current property depends on are smashed.

  Default Formula: nil

  before-smash

  The formula of this property is used to cause an event to occur just before the parent property's value is smashed.

  Default Formula: nil

geom-property [class]

This property is used to store the pointer to the true geometry of the object. Its associated events cause the pointer to the true geometry to be freed when necessary in order to free memory. This property and its corresponding method are listed here for reference only and must not to be instantiated or managed by the user.

Inheritance: smash-event change-event

free-geom [Method]

Defined on Classes:

geom-property

Free-geom frees the C structure that contains the geometry of the object.

Arguments:

- **instance**

  A graphic object.

position [class]

The position property is used to store the position of a geometric object. It is used as a property of the position-object class. Position is used internally to contain the position of an object resulting from the orientation operations applied on that object. Instances of this property should not be overwritten by the user.

Inheritance: property-object

Properties:

- **i-vector**

  The local x-axis of the geometric object.

  Default Formula: nil

  j-vector

  The local y-axis of the geometric object.

  Default Formula: nil

  origin

  The local origin of the geometric object.

  Default Formula: nil

property-modifier-object [class]

A property-modifier-object is used to hold the value of a source property. Its value is equal to the value of the source property and depends on it. A property-modifier-object also adds the capability of changing the source property value to its own value property whenever value is modified.It is used mainly for display purposes of the source property value and to give the user control over the value of that source property via an intermediate displayed property.

Inheritance: object

Properties:

- **value**

  Value of the source property.The formula of this property should not be redefined by the user.

  Default Formula: nil

  source-object

  Object instance holding the source property.

  Default Formula: nil

  source-property-name

  Name of source property. It has to be a property of source-object.

  Default Formula: nil

Example 3-5. Property-modifier-object Example

```
(in-package :aml)
```

```
(define-class property-modifier-example
  :inherit-from(object)
  :properties(
              )
  :subobjects(
              (box :class 'box-object
                   )
              (modifier :class 'property-modifier-object
                        source-object ^^box
                        source-property-name 'width
                        )
              )
  )

;; This class will establish the following:
;; The modifier value will be equal to the box width.
;; Furthermore, everytime the modifier value is changed by the user,
;;   the box width is changed also.
```

Additional Classes

Methods and Classes

visibility-object [class]

This class can be used to make an object in the model tree "invisible" to the user.

Inheritance: object

Properties:

- **tree-display?**

  If set to nil the object will not be appear in the object instance tree.

  Default Formula: t

name-generator [class]

This object provides the ability to generate unique names by appending numbers to the end of the names.

Inheritance: object

Properties:

- **previous-name-list**

  This property is used to store the names and current numbers of names that have been generated.

  Default Formula: nil

- **auto-naming?**

A value of t will cause the generate-name method to automatically generate the next name for the prefix specified. A value of nil will cause the generate-name method to prompt the user to enter a name.

Default Formula: (default t)

add-a-point [Method]

Defined on Classes:

name-generator

This method has been deprecated and is available for backward compatibility only. This method adds a point-object subobject to the specified name-generator object.

Arguments:

- **instance**

  An object of type name-generator to add the point to.

- **point**

  The coordinates of the point to add.

```
> (create-model 'name-generator)
;; Creates a name-generator model
> (add-a-point (the) '(1.0 0.0 0.0))
;; adds the point (1.0 0.0 0.0)
> (draw (the))
;;draws the name-generator which is a point right now
```

add-vertex-points [Method]

Defined on Classes:

name-generator

This method has been deprecated and is available for backward compatibility only. This method is used to add a series of point-objects to a name-generator object. The points will be named using the generate-name mechanism.

Arguments:

- **instance**

  A name generator to add the points to.

- **locations**

  A list of the coordinates for the location of each point to be added.

```
> (create-model 'name-generator)
;; creates a name-generator model
> (add-vertex-points (the) '((0.0 0.0 0.0) (1.0 1.0 1.0) (2.0 1.0 0.0)))
;; adds three points with those coordinates
> (draw (the)
```

```
;; will draw the three points
```

connect-points-with-lines [Method]

Defined on Classes:

name-generator

This method has been deprecated and is available for backward compatibility only. This method is used to connect a series of point-objects with lines.

Arguments:

- **instance**

  An object of type name-generator to add the lines to.

- **points**

  A list of point objects to connect with lines.

current-number [Method]

Defined on Classes:

name-generator

Used in conjunction with the name-generator, this method returns the number given to the last object added using the specified prefix name.

Arguments:

- **instance**

  An instance of type name-generator.

- **name**

  The name to get the current number of.

```
> (create-model 'name-generator)
;; creates a name-generator model
> (current-number (the) 'box)
;; returns nil, because it's not yet generated
> (generate-name (the) 'box)
;; now it generates name for the box
> (current-number (the) 'box)
;; returns the new number for it - 1
```

generate-name [Method]

Defined on Classes:

name-generator

Used to append an incremental suffix to the name supplied.

Arguments:

- **instance**

  A name-generator object.

- **name**

  The base name to append the numbered suffix onto.

```
> (create-model 'name-generator)
;; creates a name-generator model
> (generate-name (the) 'box)
;; returns the new name for 1st box : BOX-0001
> (generate-name (the) 'box)
;; returns another name BOX-0002
> (generate-name (the) 'cylinder)
;; returns the name of first cylinder : CYLINDER-0001
> (generate-name (the) 'cylinder)
;; returns CYLINDER-0002
> (add-object (the) (generate-name (the) 'box) 'box-object)
;; returns a box instance which name is BOX-0003
```

increment-number [Method]

Defined on Classes:

name-generator

Increments the suffix number for the given name. This method does not establish dependency with the property previous-name-list when called from a property formula.

Arguments:

- **instance**

  A name-generator object.

- **name**

  The base name to increment the suffix number of.

```
> (create-model 'name-generator)
;; creates a new name-generator model
> (current-number (the) 'box)
;; returns : NIL - default
AML> (increment-number (the) 'box)
:: increments
AML> (current-number (the) 'box)
;; returns 1
AML> (increment-number (the) 'box)
:: increments
AML> (current-number (the) 'box)
;; returns 2
> (generate-name (the) 'box)
;; returns : BOX-0003 - next box
```

reset-number [Method]

Defined on Classes:

name-generator

Resets the suffix number for the given prefix name to 0. This method does not establishdependency with the property previous-name-list when called from a property formula.

Arguments:

- **instance**

  A name-generator object.

- **name**

  The base name to reset the suffix number of.

```
> (create-model 'name-generator)
;; creates a name-generator model
> (current-number (the) 'box)
;; returns : NIL - default
> (increment-number (the) 'box)
;; increments to 1
> (current-number (the) 'box)
;; returns 1
> (increment-number (the) 'box)
;; increments to 2
> (reset-number (the) 'box)
;; resets to 0
> (current-number (the) 'box)
;; returns 0
```

set-number [Method]

Defined on Classes:

name-generator

Sets the suffix number for the given prefix name to the number supplied.

Arguments:

- **instance**

  A name-generator object.

- **name**

  The base name to change the suffix number of.

- **number**

  The new suffix number.

```
> (create-model 'name-generator)
;; creates a name-generator model
```

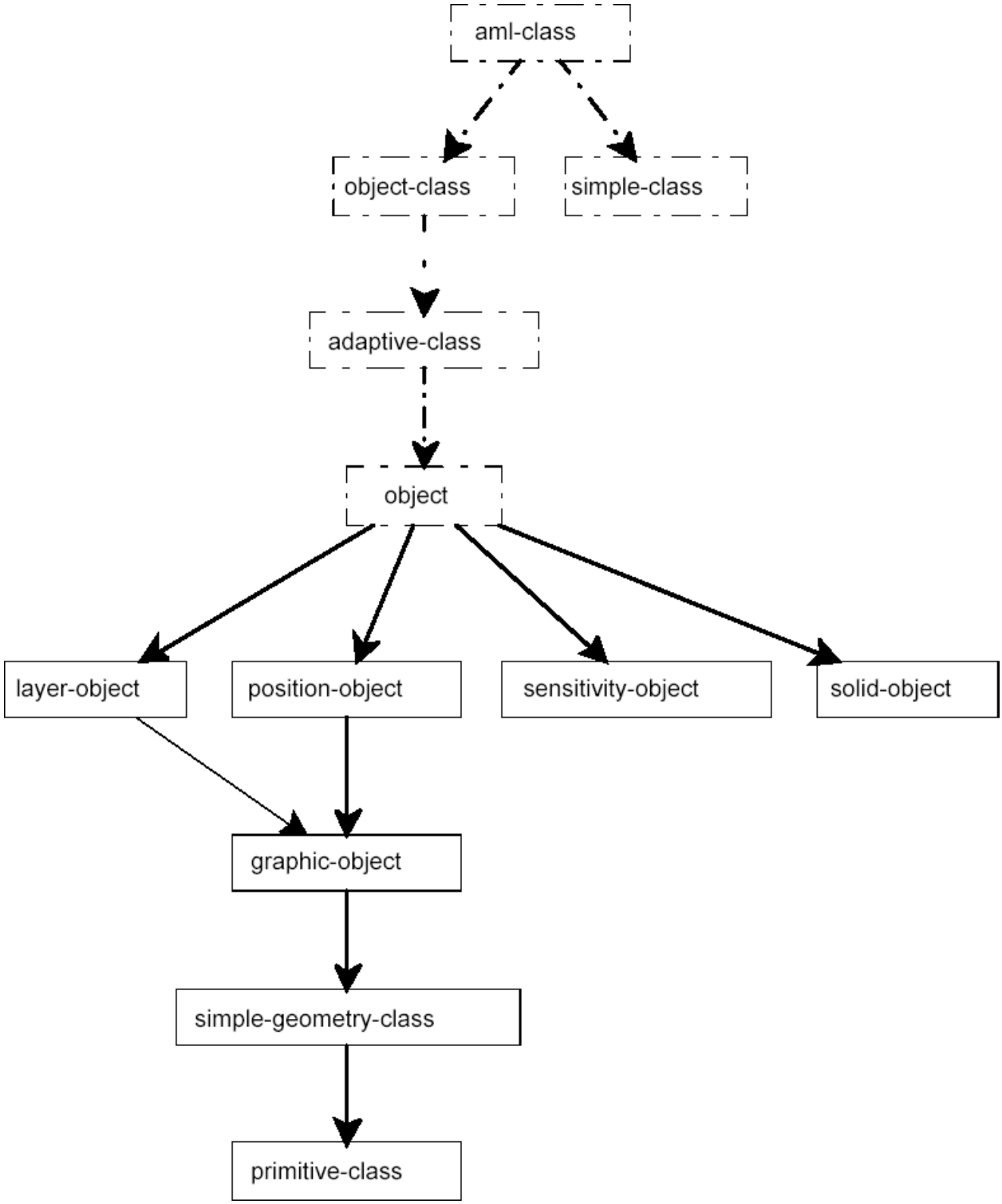
```
> (set-number (the) 'box 4)
;; sets the number to 4
> (current-number (the) 'box)
;; returns 4
```

Chapter 4. Geometric Objects

General

Position-Object

Figure 4-1. General Classes Defined in Chapter 4


aml-class
object-class
simple-class
adaptive-class
object
layer-object
position-object
sensitivity-object
solid-object
graphic-object
simple-geometry-class
primitive-class

position-object [class]

The position-object provides the ability for objects to be oriented in space. Most of the orientation methods are written on this object and are described in the chapter Orientation. All graphic objects inherit from the class graphic-object which inherits from position-object.

Inheritance: object

Properties:

- **reference-object**

  The position-object instance whose local frame will be the reference frame of this object. The default is nil which will cause positioning to be relative to global coordinates. Please refer to the chapter on orientation for more details. Please see Note below.

  Default Formula: (default nil)

- **orientation**

  The list of orientation commands used to position the object. Please refer to the orientation chapter.

- **position**

  The mathematical description for the current position of the object instance. This property should not be overwritten by the user. Please see the definition of the position property class. This property is an internal property; therefore, it is recommended that users do not make "the" references to this property or any of its subproperties for it may lead to undesirable results. It is suggested to use the method convert-coords to query the object for its position. (example: (convert-coords position-object-instance (0 0 0) :from :local :to :global)). The method convert-coords is described in the chapter Orientation.

- **reference-coordinate-system**

  Specifies the coordinate-system-class object which will be the reference frame for the position-object. The reference-coordinate-system must refer to an object of class coordinate-system-class. A value of nil indicates that the global reference frame should be used. If reference-coordinate-system refers to any object which does not inherit from coordinate-system-class, the global frame will be used. By default, the reference frame is global. (Defaults to nil). Refer to the orientation chapter for more details.

See Also: coordinate-system-class, chapter on orientation (chapter 6)

Note: The reference-coordinate-system property has been introduced in AML 3.1.2 in order to replace reference-object. For backward compatibility purposes, reference-object takes precedence when both properties are specified, and the reference-coordinate-system will be ignored. It is recommended to use a reference-coordinate-system.

The following orientation methods written on position-object are described in the Orientation chapter: apply-matrix-to-object, move-object, rotate-object, translate-object

center-of-object [Method]

Defined on Classes:

position-object,

geom-copy-object,

assembly-object

This method returns a list of the 3-dimensional coordinates for the center of an object instance.

Arguments:

- **Instance**

  The object instance for which to return the center coordinates.

Example 4-1. Center-of-object Example

```
(create-model 'box-object)
;; returns the instance of the box-object
(center-of-object (the))
;; returns (0.0 0.0 0.0)
(translate-object (the) '(1.0 2.0 3.0))
;; returns (LIST (TRANSLATE '(1.0 2.0 3.0) :DISTANCE NIL :GLOBAL? NIL))
(center-of-object (the))
;; returns (1.0 2.0 3.0)
```

See Also: center-of-edge, center-of-face, normal-to-face, vertex-of-object

Graphic-Object

graphic-object [class]

All AML classes associated with graphics and geometry inherit from graphic-object. This class should not be instantiated by the AML user nor should it be directly inherited into a user class. Note: All geometry is created with the center at the global origin, (0.0 0.0 0.0), by default.

Inheritance: position-object, layer-object

Properties:

- **color**

  The value may be a symbol or a string.

  Default Formula: white

- **display?**

  When t, an instance of this class will be capable of being drawn. When nil, an instance cannot be displayed.

  Default Formula: t

- **geom**

  This holds the geometric entity (geom id) of the instance.When an instance is drawn it uses this property value. This property should not be overwritten by the user.

  Default Formula: (progn (tsi::orient-geom (superior (current-object))))

- **render**

  This property defined the graphics rendering method. Values allowed are: boundary (wireframe graphics), shaded (shaded representation), facet (surface facets representation), isoline (grid of lines following the parametric lines of the surfaces), boundaryshaded, facet-shaded, isoline-shaded and

hidden-line(draws the hidden lines as dashed). Isoline modes do not work on the Parasolid-based solid modeler. Another value is '(shaded pattern) where pattern is a list of 128 integers to describe a 32x32 bitmap (4 bytes each row with 32 rows), each integer (0 ~ 255) represents one byte in the bitmap. Please note, this rendering pattern can only be printed using bitmap format.

Default Formula: boundary

- **line-width**

  Integer specifying the the width (in pixels) of the lines used to draw the object.

  Default Formula: 0 (Thinnest).

- **line-type**

  Integer specifying the style of the lines used to draw the object. The predefined line types range from 0 to 6: solid, dashed, dotted, dashed-dotted, dashed-double-dotted, dashed-tri-dotted, long dash. Any value above 6 (32bit integer) defines the pattern, bit 1 means the pixel is on, 0 means the pixel is off.

  Default Formula: 0

The following orientation methods defined on graphic-object are described in the Orientation chapter: interactive-rotate, interactive-rotate-bounding-box, interactive-translate, interactive-translate-bounding-box.

blink [Method]

Defined on Classes:

graphic-object

Method to make the outline of a graphic-object instance flash on the screen.

Arguments:

- **Instance**

  An instance of a graphic-object..

Optional Arguments:

- **times-to-blink**

  Must be a number indicating the number of blinks.

  Default Value: 3

- **delay**

  Time (in milliseconds) between blinks.

  Default Value: 0

Example 4-2. Blink Example

```
> (create-model 'box-object)
;; returns the instance of a box-object
> (draw (the))
;; draws the box-object
> (blink (the) 5)
;; blink the top part 5 times on the display
```

bring-object-to-front [Method]

Defined on Classes:

graphic-object

This method draws the given object in front of other objects. This functionality can be used in highlighting an object in wireframe mode or when the depth of the faces of the object is not important (since this may draw the back face of the object in front of its front face as a result).

Optional Arguments:

- **:front?**

  This keyword when set will draw the object in front of other objects.

  Default Value: t

center-of-edge [Method]

Defined on Classes:

graphic-object

This method returns the 3-dimensional coordinates for the center of a given edge of an object instance in a list.

Arguments:

- **Instance**

  The object instance containing the edge to query.

Optional Arguments:

- **edge-id**

  The id of the edge of which to return the coordinates. The id is defined by the system as a number but may be aliased by the user by defining the get-edge-id method. If nil or not given, a list of all edge centers will be returned.

Example 4-3. Center-of-edge Example

```
> (create-model 'box-object)
;; returns the instance of box-object
> (center-of-edge (the) 0)
;; returns (0.0 -0.5 0.5)
> (rotate-object (the) 90.0 '(1.0 0.0 0.0))
;; returns (LIST (ROTATE 90.0 '(1.0 0.0 0.0) :AXIS-POINT '(0.0 0.0 0.0)
:GLOBAL? NIL))
> (center-of-edge (the) 0)
;; returns (0.0f0 -0.5f0 -0.5f0)
```

See Also: center-of-face, center-of-object, normal-to-face, vertex-of-object

center-of-face [Method]

Defined on Classes:

graphic-object

This function returns the 3-dimensional coordinates for the center of a given face of an object instance in a list.

Arguments:

- **Instance**

  The object instance containing the face to query.

Optional Arguments:

- **face-id**

  The id of the face of which to return the coordinates. The id is defined by the system as a number but may be aliased by the user by defining the get-face-id method. If nil or not given, a list of all face centers is returned.

Example 4-4. Center-of-face Example

```
> (create-model 'box-object)
;; returns the instance of the box-object
> (center-of-face (the) 0)
;; returns (0.0 0.0 0.5)
> (center-of-face (the) :front)
;; returns (0.0 0.0 0.5)
> (rotate-object (the) 90.0 '(1.0 0.0 0.0))
;; returns (LIST (ROTATE 90.0 '(1.0 0.0 0.0) :AXIS-POINT '(0.0 0.0 0.0)
:GLOBAL? NIL))
> (center-of-face (the) :front)
;; returns (0.0 -0.5 0.0)
```

See Also: center-of-edge, center-of-object, normal-to-face, vertex-of-object

change-color [Method]

Defined on Classes:

graphic-object

Method to change the color of a graphic-object instance. If the instance being changed is currently displayed in the display window it will be updated.

Arguments:

- **Instance**

  The object instance containing the face to query.

- **new-color-name**

  A name of a color. Any color available in X-windows is valid.

Example 4-5. Change-color Example

```
> (create-model 'box-object)
;; returns the instance of the box-object
> (draw (the))
;; returns 0
> (change-color (the) 'dodgerblue)
;; returns 64
> (change-color (the) 'gold)
;; returns 8
> (change-color (the) 'aquamarine)
;; returns 64
```

dimension [Method]

Defined on Classes:

graphic-object

Returns the dimension of the specified object.

Arguments:

- **Instance**

  An instance of a graphic-object.

Example 4-6. Dimension Example

```
> (create-model 'box-object)
;; returns the instance of the box-object
> (dimension (the))
;; returns 3
> (change-value (the solid?) nil)
;; returns nil
> (dimension (the))
;; returns 2
> (create-model `sheet-object)
;; returns the instance of the sheet-object
> (dimension (the))
;; returns 2
> (create-model `line-object)
;; returns the instance of the line-object
> (dimension (the))
;; returns 1
```

facet [Method]

Defined on Classes:

graphic-object

Method to change the rendering method for a graphic-object to facet. If the instance is currently in the display window the display will be updated. Note: If the instance being changed is not currently displayed, (facet (the ...)) is equivalent to (change-value (the ... render) facet). If the instance is displayed this is an efficient means of updating the instance and the graphics.

Arguments:

- **Instance**

  An instance of a graphic-object.

Example 4-7. Facet Example

```
(create-model 'cylinder-object)
;; returns the instance of the cylinder-object
(draw (the))
;; returns 0
(facet (the))
;; returns 8
```

See Also: shade, unfacet

find-closest-point-on-object [Method]

Defined on Classes:

graphic-object

Given a graphic-object instance and a point in its neighborhood, this method returns the coordinate list of the point that lies exactly on the object and is the closest to the point provided. The method returns nil if there is not point on the object provided that lies within the tolerance with respect to the point provided.

Arguments:

- **Instance**

  An instance of a graphic-object.

- **point**

  Coordinate list (X Y Z) of the point located in the neighborhood of the graphic-object instance.

- **tolerance**

  Number specifying the tolerance distance to the graphic-object instance within which the point argument is located.

generate-points-on-curve-object [Method]

Defined on Classes:

graphic-object

This method evaluates points on a curve. Note that the graphic-object must have only one edge. If it has more than one edge, the points will be evaluated on the first edge. The method returns a list of the points coordinates.

Arguments:

- **Instance**

  An instance of a graphic-object.

Keyword Arguments:

- **:n**

  This keyword specifies the number of points to be generated on the curve.

  Default Value: 10

- **:tolerance**

  This keyword specifies how accurately the points lie on the curve.

  Default Value: 0.00001

get-a-point [Method]

Defined on Classes:

graphic-object

General method for graphic-object (and grid-object) used to select a point from the display. A right click will pull up a form for more selection options. The coordinates of the point and the formula used to get the point are returned in a list.

Arguments:

- **object-instance**

  An instance of a grid-object or graphic-object.

Keyword Arguments:

- **:message**

  Optional keyword. When a string is specified, it overwrites the default message in the current message pane.

  Default Value: 10

get-face-id [Method]

Defined on Classes:

graphic-object

The get-face-id method is a hook that can be defined for classes in order to customize the face names for objects. Internally when an action requires a face id this method is called with the instance for which to get the id. Therefore, defining a get-face-id method for a class will allow the faces to be referred to by any names found in the method.

Arguments:

- **instance**

  The object instance of which to return the face number.

- **id-name**

Any keyword (recommended), symbol, or number that names a face for the instance.

Example 4-8. get-face-id Example

```
;; This method already exists in the system and does not need defined unless
the names are to be changed.
(define-method get-face-id box-object (name)
  (case name
    ((0 :front) 0)                      ; alias face 0 as :front
    ((1 :back) 1)                       ; alias face 1 as :back
    ((2 :bottom) 2)                     ; alias face 2 as :bottom
    ((3 :right) 3)                      ; alias face 3 as :right
    ((4 :top) 4)                        ; alias face 4 as :top
    ((5 :left) 5)                       ; alias face 5 as :left
    (otherwise (cerror "Enter new face-id: "
                       (format nil "Face ~a does not exist on object ~a" name
                               (find-tree self)))
               (setq name (read-from-string (read-line) nil))
               (get-face-id self name))
    )
  )



(create-model 'box-object)
;; creates a new box-object
(get-face-id (the box-object) :right)
;; returns : 3
(get-face-id (the box-object) 3)
;; returns : 3
(get-face-id (the box-object) :top)
;; returns : 4
```

See Also: get-edge-id, get-vertex-id

get-moments-of-inertia [Method]

Defined on Classes:

graphic-object

This method computes the moments of inertia of an object. These can be computed in different reference frames or about specific axes.

Arguments:

- **instance**

  The object for which moments of inertia are to be calculated.

Keyword Arguments:

- **:coord-system**

  This specifies the coordinate system which provides the reference frame for the moments of inertia calculation. The valid arguments are :global, :ref, :local, :centroid, or a list specifying the coordinates

of the origin of the frame. If no coordinate system is specified, the calculations will be performed in the global frame. The coordinate system will be used to calculate the origin of the frame and its x, y, and z axes. If the :axes argument is also specified, the value given to that argument will override any value coming from :coord-system. In this way, the origin and axes can be specified separately. An argument of :global will perform all calculations in the global frame. An argument of :local will perform those calculations in the local frame of the object. An argument of :ref will perform the calculations in the frame of the reference coordinate system of the object. An argument of :centroid will perform the calcuations centered at the center of mass of the object and will use the global axes (unless overridden by the value specified in :axes). If a list specifying the origin of the frame (e.g., '(5.0 3.4 1.3)) is given, this will be used as the origin and the axes will default to the global axes. Note that the coordinates specified must be in the global system. Options: :global, :local, :ref, :centroid, list of coordinates.

Default Value: :global

- **:axes**

  The axis system of the coordinate frame about which the moments of inertia are calculated. The options are :global, :local, :ref, and :principal. If no :axes are specified (:axes is nil), then the following defaults are used for each coordinate system (:global - :global), (:local - :local), (:ref - :ref), and (:centroid - :global). The option for :principal axes is currently not supported. If it is specified, then an error message will be displayed and the :global axes will be used. Alternately, a list of two or three vectors can be given which specify the axis system. If only two vectors are given, the third will be determined by taking their cross-product. Options: :global, :local, :ref, :principal, list.

- **:about**

  If a vector is specified, then the moment of inertia about this axis will be computed and returned.

- **:density**

  If specified, then this value is used for the material density of the object when computing moments of inertia.

- **:values?**

  If t, it will return all possible return values in a list, by default returns only the moment-of-inertia-matrix.

  Default Value: nil

Return Values:

1. moment-of-inertia-matrix : The nine values in the moment of inertia matrix are returned in a single list.
2. position-of-centroid : The location of the objects centroid in global coordinates.
3. volume : The objects volume.
4. mass : The objects mass.
5. moment-of-inertia : If a vector is specified in :about, then the moment of inertia about that axis is returned.

get-sub-geoms [Method]

Defined on Classes:

graphic-object

Returns all of the geometric sub elements of the specified object.

Arguments:

- **instance**

  An instance of a graphic-object.

- **dimension**

  By default this will return all the sub-geoms. If 1, 2, or 3 is specified, only subgeoms of that dimension will be returned.

  Default Value: nil

Example 4-9. get-sub-geoms Example

```
(create-model 'box-object)
;; returns the instance of a box-object
(get-sub-geoms (the))
;; returns a list of geoms of dimensions 1, 2, and 3
(get-sub-geoms (the) 1)
;; returns a list of geoms with dimension 1
(get-sub-geoms (the) 2)
;; returns a list of geoms with dimension 2
(get-sub-geoms (the) 3)
;; returns a list of geoms with dimension 3
```

mesh-data-from-files-mixin [class]

This is a superclass for the class mesh-data-type-1-from-files-class.

Inheritance: primitive-class

mesh-data-type-1-from-files [class]

The following methods are

Inheritance: mesh-data-from-files-mixin

Properties:

- **coords-file-path**

  Coordinates file where each line in the file is of the format: node-id x y z.

- **connectivity-file-path**

  Connectivity file where each line in the file is of the format: element-id node-1-id node-2-id ... node-n-id, (2 <= n <= inf).

minmax-box [Method]

Defined on Classes:

graphic-object

Returns minimum and maximum values along the global X,Y and Z axis that bound the geometry of a given object. The values are returned as a list with three sublists, each representing the minimum and maximum value for the x, y, and z coordinates, respectively. Note: Since this method operates in the global frame, it does not necessarily return the dimensions of the smallest box that could contain the geometry of the object, nor can it return the bounding values in any other coordinate frame.

Arguments:

- **instance**

  The object instance of which to return the vertex number.

Example 4-10. minmax-box Example

```
(create-model 'box-object)
; returns an instance of a box-object
(minmax-box (the))
;; returns ((-0.5 0.5) (-0.5 0.5) (-0.5 0.5))
(rotate-object (the) 45.0 '(1.0 1.0 1.0))
;; returns : (LIST (ROTATE 45.0 '(1.0 1.0 1.0) :AXIS-POINT '(0.0 0.0 0.0)
:GLOBAL? NIL))

(minmax-box (the)) ; returns ((-0.810617208480835 0.810617208480835) (-
0.810617208480835 0.810617208480835) (-0.810617208480835 0.810617208480835))
```

normal-to-face [Method]

Defined on Classes:

graphic-object

Normal-to-face returns the 3-dimensional vector defining the vector normal to the given face of a given object instance. By definition the normal to a face is inward.

Arguments:

- **instance**

  The object instance containing the face to query.

- **face-id**

  The id of the face of which to return the normal vector. The id is defined by the system as a number but may be aliased by the user by defining the get-face-id method.

Example 4-11. normal-to-face Example

```
> (create-model 'box-object)
;; returns an instance of a box-object
> (normal-to-face (the) 0)
;; returns (0.0 0.0 -1.0)
> (rotate-object (the) 45.0 '(1.0 1.0 1.0))
;; returns (LIST (ROTATE 45.0 '(1.0 1.0 1.0) :AXIS-POINT '(0.0 0.0 0.0)
:GLOBAL? NIL))
> (normal-to-face (the) 0)
```

```
:; returns (-0.50587934f0 0.3106172f0 -0.80473787f0)
```

See Also: center-of-edge, center-of-face, center-of-object, vertex-of-object

null-geom? [Method]

Defined on Classes:

graphic-object

Used to determine whether or not the geom of the object is valid. If null-geom? returns t, then the geom is not valid.

Arguments:

- **instance**

  An instance of a graphic object.

Example 4-12. null-geom? Example

```
> (create-model 'box-object)
; returns an instance of the box-object
> (null-geom? (the))
;; returns NIL
```

object-in-display? [Method]

Defined on Classes:

graphic-object

Returns non nil when object is in current display window.

Arguments:

- **instance**

  An instance of a graphic object.

object-sensitive? [Method]

Defined on Classes:

graphic-object

Used to check if the object is selectable by mouse from the display. This returns t, except when the object also inherits from sensitivity-object, in which case it checks the property mouse sensitive.

Arguments:

- **instance**

  An instance of a graphic object.

See Also: sensitivity-object

remove-object-from-display [Method]

Defined on Classes:

graphic-object

This method is used to remove an instance from the list of drawn objects. The object will not be removed from the screen until an action, such as regen, that would cause it to be updated is performed.

Arguments:

- **instance**

  An instance of an object to be removed from the drawn objects.

Example 4-13. remove-object-from-display Example

```
(create-model 'name-generator)
;; returns an instance of a name-generator
(add-object (the) 'box-object 'box-object)
;; returns an instance of a box-object
(add-object (the) 'cylinder-object 'cylinder-object)
;; returns an instance of a cylinder-object
(draw (the))
;; returns 0
(remove-object-from-display (the box-object))
;; returns the geom of the box-object
(regen)
;; returns a geom
```

See Also: undraw

save-object-geom [Method]

Defined on Classes:

graphic-object

For an object with a valid geom, this method will save the geom to the specified file.

Note : When using the AML Parasolid-based solid modeler, the system adds a platform dependent extension to the file name. On WINDOWS FAT systems, ".X_T" is added. On UNIX and WINDOWS NTFS systems, ".xmt_txt" is added.

Arguments:

- **instance**

  An instance of a graphic-object.

- **filename**

  A string specifying the name [and path] of the file to which the geom is saved.

  Default Value: nil

- **:scale-factor**

  This is the factor that the saved object is scaled by after retrieval.

Default Value: 1.0

See Also: saved-geom-object

select-closest-edge [Method]

Defined on Classes:

graphic-object

It returns the closest edge subgeom of the graphic object to the given point.

Arguments:

- **point**

  Reference point.

Keyword Arguments:

- **:values?**

  When t, it returns the edge and the distance as a list.

select-closest-face [Method]

Defined on Classes:

graphic-object

It returns the closest face subgeom of the graphic object to the given point.

Arguments:

- **point**

  Reference point.

Keyword Arguments:

- **:values?**

  When t, it returns the face and the distance as a list.

select-edge [Method]

Defined on Classes:

graphic-object

Method for interactively selecting an edge from an instance that is displayed in the display window. It returns two values: The vgl geom id of the edge selected and the index of that edge with the graphic-object.

Arguments:

- **instance**

  An instance of a graphic-object.

Example 4-14. Select-edge Example

```
(create-model 'box-object)
```

```
;; returns an instance of a box-object
(draw (the))
;; returns 0
(select-edge (the))
;; returns 2 values. The geom of the edge selected and the selected edges' id
; waits for a mouse click and returns the values
```

See Also: select-face, select-point

select-face [Method]

Defined on Classes:

graphic-object

Method for interactively selecting a face from an instance that is displayed in the display window. It returns two values: The vgl geom id of the face selected and the index of that face within the graphic-object.

Arguments:

- **instance**

  An instance of a graphic-object.

Example 4-15. Select-face Example

```
(create-model 'box-object)
;; returns an instance of a box-object
(draw (the))
;; returns 0
(select-face (the))
;; returns 2 values. The geom of the face selected and the id of the selected
face
9199644
3
;; waits for a mouse click and returns the values
```

See Also: select-edge, select-point

select-point [Method]

Defined on Classes:

graphic-object

Method for interactively selecting a vertex from an instance that is displayed in the display window. It returns two values: The vgl geom id of the point selected and the index of that point within the graphic-object.

Arguments:

- **instance**

  An instance of a graphic-object.

Example 4-16. Select-point Example

```
(create-model 'box-object)
;; returns an instance of a box-object
(draw (the))
;; returns 0
(select-point (the))
;; returns 2 values. The geom of the point selected and the id of the
selected vertex
10155416
4

;; waits for a mouse click and returns the values
```

See Also: select-edge, select-point

shade [Method]

Defined on Classes:

graphic-object

Method to change the rendering type of a graphic-object to shaded. If the instance is currently in the display window the display will be updated.

If the instance being changed is not currently displayed, (shade (the ...) t) is equivalent to (change-value (the ... render) 'shaded). If the instance is displayed this is an efficient means of updating the instance and the graphics.

Arguments:

- **instance**

  An instance of a graphic-object.

Optional Arguments:

- **:shade?**

  Possible values are: t for shaded representation; 'facet for faceted representation; 'boundary-shaded for boundary representation.

  Default Value: t

- **:shade-subobjects?**

  When t, shades all the graphic-objects in the tree starting at instance.

  Default Value: nil

- **:update-window?**

  When t, the display (canvas) is refreshed at the end of executing shade. When nil, the display (canvas) is not refreshed.

  Default Value: t

Example 4-17. Shade Example

```
> (create-model 'cylinder-object)
;; returns an instance of a cylinder-object
> (draw (the))
;; draws the cylinder and returns nil
> (shade (the) 'facet)
;; facets the geometry and returns nil
```

See Also: select-facet, toggle-shade, unshade

show-face [Method]

Defined on Classes:

graphic-object

This method will display a point at the center of the face and the normal vector of the face. The thickness controls the point thickness in pixels; the color controls the color of the vector.

Arguments:

- **instance**

  An instance of a graphic-object.

- **face-id**

  The name or id of the face to show.

Keyword Arguments:

- **:thickness**

  The number of pixels thick to make the point at the center of the face.

  Default Value: 3

- **:color**

  The color of the vector to draw.

  Default Value: 'green

Example 4-18. Show-face Example

```
> (create-model 'box-object)
;; returns an instance of the box-object
> (draw (the))
;; draws the box and returns nil
> (change-view :iso)
;; changes the view and returns t
> (show-face (the) :front)
;; returns the geom of the face that appears in front
```

snap-on-grid [Method]

Defined on Classes:

graphic-object

This method is used to graphically drag and select a point on a grid using the grid properties to control the snap and extent of the grid. It can also be called on a graphic-object. It returns two values: The first value is 1 when the point is selected with a left mouse click, and 0 when the point is selected with a right mouse click. The second value is a list of the x,y,z coordinates of the point selected.

Arguments:

- **instance**

  An instance of grid-object or graphic-object to drag the point on.

- **point**

  The initial point.

  Default Value: nil

Example 4-19. Snap-on-grid Example

```
(create-model 'name-generator)
;; returns an instance of name-generator
(add-object (the) 'grid-object 'grid-object)
;; returns an instance of a grid-object
(draw (the))
;; draws the grid and returns nil
(zoom :all)
;; returns a list of x and y coordinates (More info needed)
(snap-on-grid (the grid-object))
;; Drag the point on the grid returns the vertex of the point
(change-value (the snap (:from (the grid-object))) t)
;; returns t
(snap-on-grid (the grid-object))
;; Then drag the point on the grid. Notice it now snaps to the points
specified by the
;; snap spacing of the grid. Returns the vertex of the new point.
1
(2.0f0 6.0f0 0.0f0)
```

toggle-shade [Method]

Defined on Classes:

graphic-object

Used to toggle between boundary and shade for the render of the object.

Arguments:

- **instance**

  An instance of a graphic-object.

Example 4-20. Toggle-shade Example

```
> (create-model 'cylinder-object)
;; returns an instance of a cylinder-object
> (draw (the))
;; draws the cylinder and returns nil
> (change-view :iso)
;; changes view
> (zoom :all)
;; fits the view and returns a x,y ???
(2.95804f0 2.366432f0)
> (toggle-shade (the))
;; returns : 2
> (toggle-shade (the))
;; returns : 64
> (toggle-shade (the))
;; returns 9578607
```

See Also: shade, unshade

unfacet [Method]

Defined on Classes:

graphic-object

Method to change the rendering method for a graphic-object to boundary. If the instance is currently in the display window the display will be updated. Note: If the instance being changed is not currently displayed, (unfacet (the ...)) is equivalent to (change-value (the ... render) 'boundary). If the instance is displayed this is an efficient means of updating the instance and the graphics.

Arguments:

- **Instance**

  An instance of a graphic-object.

Example 4-21. Unfacet Example

```
(create-model 'cylinder-object)
;; returns the instance of the cylinder-object
(draw (the))
;; returns 0
(facet (the))
;; returns 10239727
(unfacet (the))
;; returns 9972735
```

See Also: facet

unshade [Method]

Defined on Classes:

graphic-object

Method to change the render for a graphic-object to boundary. If the instance is currently in the display window the display will be updated.

If the instance being changed is not currently displayed, (unshade (the ...) t) is equivalent to (change-value (the ... render) 'boundary). If the instance is displayed this is an efficient means of updating the instance and the graphics.

Arguments:

- **instance**

  An instance of a graphic-object.

Optional Arguments:

- **:unshade-subobjects?**

  When t, unshades all the graphic-objects in the tree starting at instance.

  Default Value: nil

- **:update-window?**

  When t the display (canvas) is refereshed at the end of executing shade. When nil, the display (canvas) is not refreshed.

  Default Value: t

Example 4-22. Unshade Example

```
> (create-model 'cylinder-object)
;; returns an instance of a cylinder-object
> (draw (the))
; draws the cylinder and returns nil
> (unshade (the) 'facet) ;  returns nil
```

See Also: shade, toggle-shade

vertex-of-object [Method]

Defined on Classes:

graphic-object

This method returns a list of the coordinates for each of the vertices of an object or the coordinates for an individual vertex.

Arguments:

- **Instance**

  The object instance containing the vertex to query.

- **vertex-id**

  The id of the vertex of which to return the coordinates. The id is defined by the system as a number but may be aliased by the user by defining the get-vertex-id method.

Example 4-23. Vertex-of-object Example

```
> (create-model 'box-object)
;; returns an instance of box-object
> (vertex-of-object (the) 0)
;; returns the vertex
> (rotate-object (the) 90.0 '(1.0 0.0 0.0))
;; returns  (LIST (ROTATE 90.0 '(1.0 0.0 0.0) :AXIS-POINT '(0.0 0.0 0.0)
:GLOBAL? NIL))
> (vertex-of-object (the) 0)
;; returns vertex
```

See Also: center-of-edge, center-of-face, normal-to-face, center-of-object

volume-of-object [Method]

Defined on Classes:

graphic-object

The value returned by this method depends on the dimension of the object. If dimension = 1, the length is returned. If dimension = 2, the surface area is returned. If dimension = 3, the volume is returned.

Arguments:

- **Instance**

  An instance of a graphic-object.

Keyword Arguments:

- **point-tol**

  Faceting value tolerances effective only with the SHAPES-based solid modeler.

- **ang-tol**

  Faceting value tolerances effective only with the SHAPES-based solid modeler.

- **edge-tol**

  Faceting value tolerances effective only with the SHAPES-based solid modeler.

Example 4-24. Volume-of-object Example

```
> (create-model 'box-object)
;; returns an instance of a box-object
> (volume-of-object (the))
;; returns 1.0
> (change-value (the height) 2.0)
;; returns 2.0
> (change-value (the width) 2.0)
```

```
;; returns 2.0
> (change-value (the depth) 2.0)
;; returns 2.0
> (volume-of-object (the))
;; returns 8.0
> (create-model 'cylinder-object)
;; returns an instance of a cylinder-object
> (volume-of-object (the))
;; returns 1.5707963267948966
```

write-stl-file [Method]

Defined on Classes:

graphic-object

Saves the geometry of a graphic-object instance into a file using STL (Stereo Lithography) format.

Arguments:

- **Instance**

  Object instance of type graphic-object.

- **filename**

  String representing the full path and name of the output STL file.

Keyword Arguments:

- **name**

  Label (string) given to the STL file. It defaults to a string of the object-name of instance.

- **PntTol**

  Tolerance used for points in writing the STL file.

  Default Value: nil

- **AngTol**

  Tolerance used for angles in writing the STL file.

  Default Value: 0.25

- **EdgeTol**

  Tolerance used for edges length in writing the STL file. When a non-nil value is specified, PntTol and AngTol are ignored.

  Default Value: nil

zoom-object [Method]

Defined on Classes:

graphic-object

If the object is drawn in the display window, the window will be zoomed to fit the object.

Arguments:

- **Instance**

  Object instance of type graphic-object.

Additional Graphic Classes

layer-object [class]

This class adds the layer property to graphic-objects. This property can be used to manage different layers of objects in a model.

Inheritance: object

Properties:

- **layer**

  Contains the identifier for the object layer

  Default Formula: (default 0)

simple-geometry-class [class]

This class is used by the system primitives in order to create fast wire frame representations. When possible the system will use the simple geometric representation instead of the true geometric model in order to save time. The system automatically uses the appropriate geometric representation so that the user does not have to be concerned about the geometric model being used.

Inheritance: graphic-object

primitive-class [class]

All geometric objects that are drawn with simple representations inherit from this class. When the true geometry is required, the true geometry is computed and used but the simple geometry is used for faster drawing of boundary representations.

Inheritance: simple-geometry-class

sensitivity-object [class]

This class is used to control whether an object is selectable from the graphics display. If a user class inherits from sensitivity-object and another AML class (multiple inheritance), sensitivity-object must be stated before any graphic object in the "inherit-from" list.

Inheritance: object

Properties:

- **highlight-color**

  Controls the highlighting color of the object.

  Default Formula: 'red

- **mouse-sensitive?**

  Controls whether or not the object will be highlightable.

  Default Formula: t

object-sensitive? [Method]

Defined on Classes:

sensitivity-object

See Also: object-sensitive?

solid-object [class]

Solid-object provides three dimensional objects a property to determine if the geometry is solid or a hollow shell.

Inheritance: object

Properties:

- **solid?**

  When true the geometry will be a solid. When nil the geometry will be created as a hollow shell. Changing the property will cause the geometry to update.

  Default Formula: (default t)

Primitives

Classes

arc-object [class]

An arc-object is a segment of a circle that is defined in the x-y plane in 3-dimensional space.

Inheritance: primitive-class

Properties:

- **radius**

  Arc Radius

  Default Formula: 0.2

- **start-angle**

  Start angle of the arc

  Default Formula: 0.0

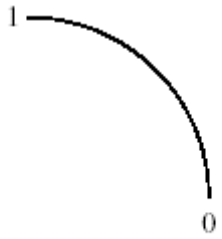
- **end-angle**

  End angle of the arc

  Default Formula: 180.00

See Also: arc-ctr-point-angle-nml, arc-ctr-radius-sta-enda, arc-3-points, arc-tg, arc-cl-cl-tg, arc-ln-cl-tg, arc-ln-ln-tg, arc-ln-pt-tg

Figure 4-2. Arc-object



The arc above displays the vertices of the arc-object. The vertex number 0 corresponds to the start-angle and the vertex number 1 corresponds to the end-angle.

arc-3-points [class]

The arc-3-points object defines an arc by specifying three points on the arc.

Inheritance: arc-object

Properties:

- **point1**

  The starting point of the arc.

  Default Formula: nil

- **point2**

  An arbitrary third point on the arc.

  Default Formula: nil

- **point3**

  The ending point of the arc.

  Default Formula: nil

See Also: arc-object

arc-ctr-point-angle-nml [class]

The arc-ctr-point-angle-nml defines an arc by specifying the center of the arc, a point on the chord of the arc, the normal of the arc, and the angle of the arc.

Inheritance: arc-object

Properties:

- **angle**

  The sweep angle of the arc chord.

  Default Formula: 180.0

- **normal**

  The normal to the arc.

Default Formula: '(0 0 1)

- **point**

  A point on the chord of the arc.

  Default Formula: '(1 0 0)

- **center**

  The center point of the arc.

  Default Formula: '(0 0 0)

See Also: arc-object

arc-ctr-radius-sta-enda [class]

The **arc-ctr-radius-sta-enda** object defines an arc by specifying its center point, its start-angle, its end angle, and its radius.

Inheritance: arc-object

Properties:

- **reference-vector**

  Default Formula: '(1 0 0)

- **normal**

  Default Formula: '(0 0 1)

- **center**

  Default Formula: '(0 0 0)

- **radius**

  Default Formula: 0.2

- **start-angle**

  Default Formula: 0.0

- **end-angle**

  Default Formula: 180.0

See Also: arc-object

box-object [class]

A box-object is a right rectangular prism described by height, width, and depth. A box-object may be a shell or a solid which is determined by the value of the solid? property inherited from solid-object. The vertices, edges, and faces of the box have user accessible id numbers. The methods getvertex-id, get-edge-id, and get-face-id may be written to give the vertices, edges, and faces custom names

Inheritance: position-object, layer-object

Properties:

- **height**

  The height is defined parallel to the y-axis.

Default Formula: 1.0

- **width**

  The width is defined parallel with the x-axis.

  Default Formula: 1.0

- **depth**

  The depth is defined parallel to the z-axis.

  Default Formula: 1.0

- **color**

  The value may be a symbol or a string.

  Default Formula: "white"

- **display?**

  When t, an instance of this class will be capable of being drawn. When nil, an instance cannot be displayed.

  Default Formula: t

- **render**

  This property defined the graphics rendering method. Values allowed are: 'boundary (wireframe graphics), 'shaded (shaded representation), 'facet (surface facets representation), 'isoline (grid of lines following the parametric lines of the surfaces), 'boundaryshaded, 'facet-shaded, 'isoline-shaded. Isoline modes do not work on the Parasolid-based solid modeler.

  Default Formula: 'boundary

- **line-width**

  Integer specifying the the width (in pixels) of the lines used to draw the object.

  Default Formula: 0

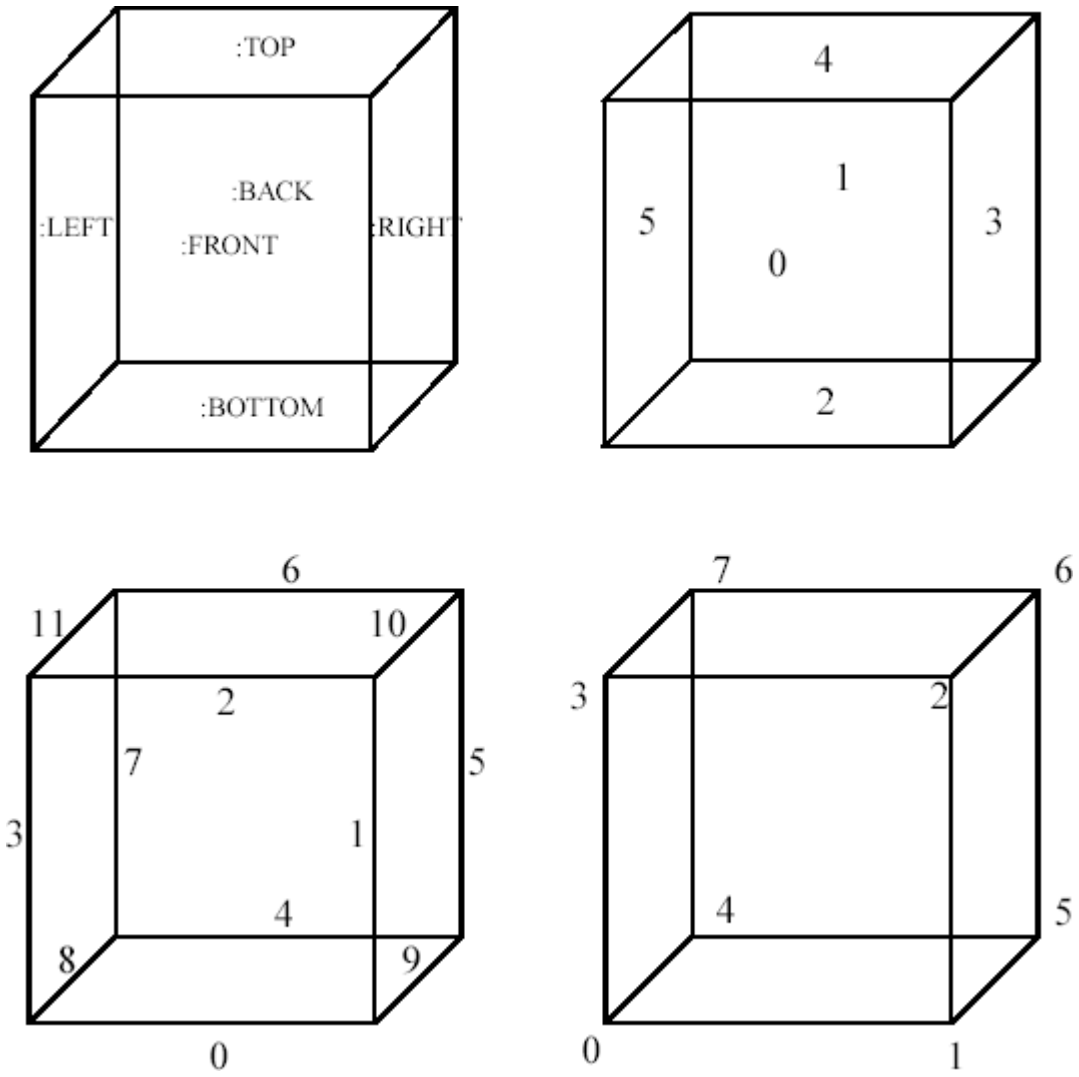
- **line-type**

  Integer specifying the style of the lines used to draw the object. The predefined line types range from 0 to 6: solid, dashed, dotted, dashed-dotted, dashed-double-dotted, dashed-tri-dotted, long dash. Any value above 6 (32bit integer) defines the pattern, bit 1 means the pixel is on, 0 means the pixel is off.

  Default Formula: 0.0

See Also: interactive-rotate, interactive-rotate-bounding-box, interactive-translate, interactive-translate-bounding-box.

Figure 4-3. Box-object



The box in the upper left shows the faces as named by the get-face-id method on box-object. The box in the upper right shows the names given the faces by default. The box in the lower left shows the edge names. The box in the lower right shows the vertex names.

circle-object [class]

A circle-object is a defined in the x-y plane in 3-dimensional space. A circle-object has no area. For a circular object with included area, use a disc-object.

Inheritance: primitive-class

Properties:

- **diameter**

  Default Formula: 1.0

See Also: circle-2-pts-dia-nml, circle-3-points, circle-ctr-nml-diameter, circle-tg, disc-object

circle-2-pts-dia-nml [class]

The circle-2-pts-dia-nml object defines a circle by specifying two points on the circumference and the normal to the circle. The diameter is the distance between the two points.

Inheritance: circle-object

Properties:

- **normal**

  Default Formula: '(0 0 1)

- **point1**

  Default Formula: '(-0.5 0.0 0.0)

- **point2**

  Default Formula: '(0.5 0.0 0.0)

See Also: circle-object

circle-3-points [class]

The circle-3-points object defines a circle by selecting 3 points on the circumference of the circle.

Inheritance: circle-object

Properties:

- **point1**

  Default Formula: '(1 0 0)

- **point2**

  Default Formula: '(0 1 0)

- **point3**

  Default Formula: '(-1 0 0)

See Also: circle-object

circle-ctr-nml-diameter [class]

The circle-ctr-nml-diameter object defines a circle by specifying the center, the normal, and the diameter.

Inheritance: circle-object

Properties:

- **normal**

  Default Formula: '(0 0 1)

- **center**

  Default Formula: '(0 0 0)

- **diameter**

  Default Formula: 1.0

See Also: circle-object

cone-object [class]

A cone-object is the capped version of the open cone that may be made a hollow shell by setting the solid? property to nil.

Inheritance: open-cone-object solid-object

See Also: open-cone-object, open-truncated-cone-object, truncated-cone-object

Figure 4-4. Cone-object

The cone on the left shows the system default names for the faces and edges of the cone-object. The cone on the right shows the names mapped by get-face-id.

cone-pipe-object [class]

This class creates a pipe that has different diameters at its ends.

Inheritance: primitive-class

Properties:

- **start-outer-diameter**

  The starting diameter of the pipe.

  Default Formula: 1.0

- **end-outer-diameter**

  The ending diameter of the pipe.

  Default Formula: 1.0

- **thickness**

  The wall thickness of the pipe.

  Default Formula: 0.1

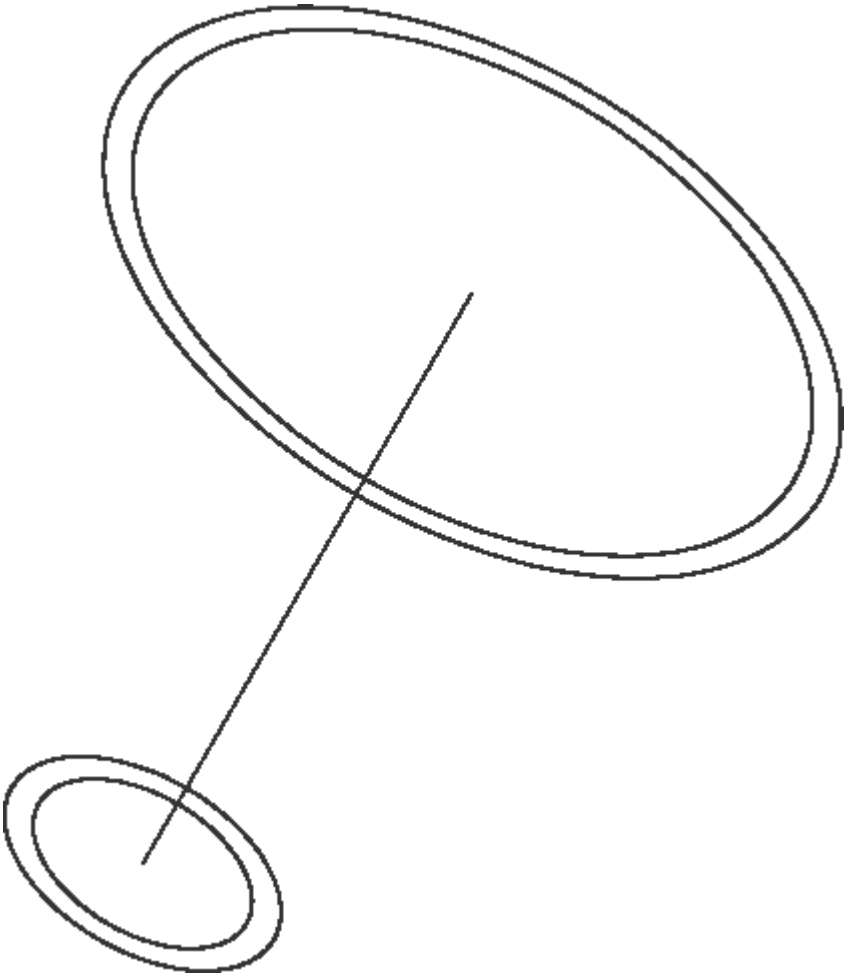
- **height**

  The length of the pipe.

  Default Formula: 2.0

See Also: open-truncated-cone-object, pipe-object

Figure 4-5. Cone-pipe-object



A pipe object with different diameters at both ends.

Example 4-25. cone-pipe-object example

```
(define-class REDUCER
  :inherit-from (cone-pipe-object)
  :properties (
               od-in 1.0
               od-out 0.375
               thickness 0.039
               start-outer-diameter !od-in
               end-outer-diameter !od-out
               height 1.0 )
  )
```

cyclide-object [class]

The cyclide-object.

Inheritance: solid-object

Properties:

- **a**

  (- a m) > 0

  Default Formula: 3.0

- **m**

  (- m c) > 0

  Default Formula: 2.0

- **c**

  Default Formula: 1.0

- **theta1**

  0 <= theta1 < theta2 <= (* 2.0 pi)

  Default Formula: 0.0

- **theta2**

  Default Formula: (* 2.0 pi)

- **phi1**

  0 <= phi1 <= phi2 <= (* 2.0 pi)

  Default Formula: 0.0

- **phi2**

  Default Formula: (* 2.0 pi)

- **solid?**

  Default Formula: t

cylinder-object [class]

A cylinder-object is an open-cylinder-object with caps on both ends and it can be a hollow shell by setting the solid? property to nil.

Inheritance: open-cylinder-object, solid-object

Properties:

- **height**

  The height is defined parallel to the y-axis.

  Default Formula: 2.0

- **diameter**

  The diameter is defined parallel with the x-axis.

  Default Formula: 1.0

- **color**

  The value may be a symbol or a string.

Default Formula: "white"

- **display?**

  When t, an instance of this class will be capable of being drawn. When nil, an instance cannot be displayed.

  Default Formula: t

- **render**

  This property defined the graphics rendering method. Values allowed are: 'boundary (wireframe graphics), 'shaded (shaded representation), 'facet (surface facets representation), 'isoline (grid of lines following the parametric lines of the surfaces), 'boundaryshaded, 'facet-shaded, 'isoline-shaded. Isoline modes do not work on the Parasolid-based solid modeler.

  Default Formula: 'boundary

- **line-width**

  Integer specifying the the width (in pixels) of the lines used to draw the object.

  Default Formula: 0

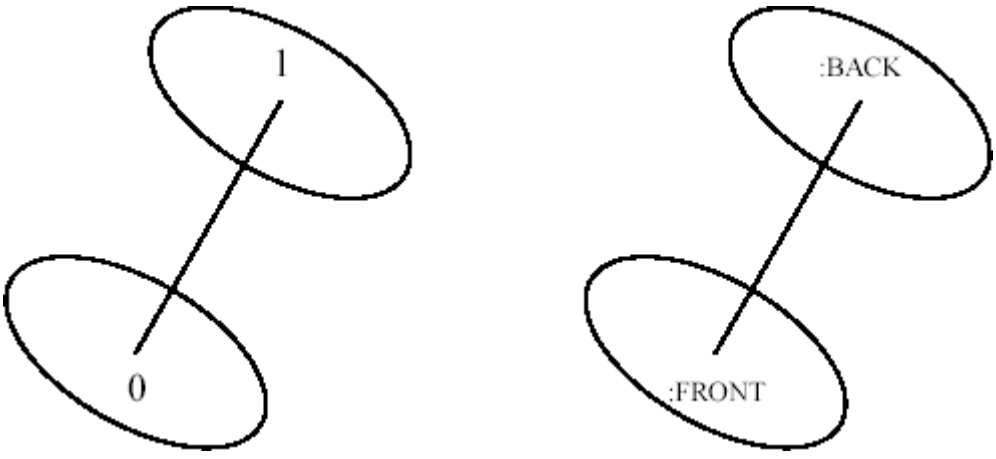
- **line-type**

  Integer specifying the style of the lines used to draw the object. The predefined line types range from 0 to 6: solid, dashed, dotted, dashed-dotted, dashed-double-dotted, dashed-tri-dotted, long dash. Any value above 6 (32bit integer) defines the pattern, bit 1 means the pixel is on, 0 means the pixel is off.

  Default Formula: 0.0

See Also: open-cylinder-object, truncated-cone-object

Figure 4-6. Cylinder-object



The cylinder on the left shows the system default names for the faces and edges of the cylinder. The cylinder on the right shows the names mapped by get-face-id.

disc-object [class]

A disc-object is a filled circle defined in the x-y plane in 3-dimensional space. A disc-object has area.

Inheritance: primitive-class

Properties:

- **diameter**

  Default Formula: 1.0

See Also: circle-object

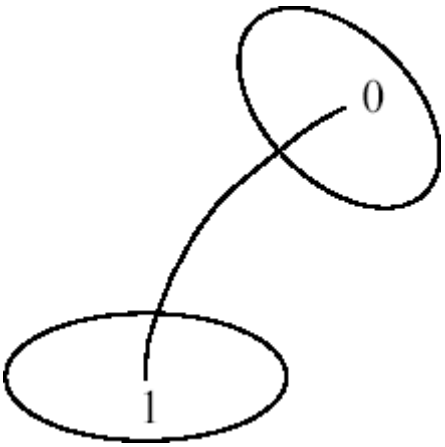
elbow-object [class]

An elbow-object is a system class that will draw fast revolutions of a disc.

Inheritance: open-elbow-object, solid-object

See Also: extended-elbow-object, open-elbow-object, open-elbow-object, variable-open-elbow-object

Figure 4-7. Elbow-object



The elbow above shows the faces of the elbow-object. The 0 face is the face that lies in the x, y plane when the object is not oriented.

ellipsoid-object [class]

The ellipsoid-object can be defined as a geometric shell or solid. Setting the solid? property to true will create a solid.

Inheritance: graphic-object, solid-object

Properties:

- **x-diameter**

  ellipsoid axis length along x-axis

  Default Formula: 1.0

- **y-diameter**

  ellipsoid axis length along y-axis

  Default Formula: 1.0

- **z-diameter**

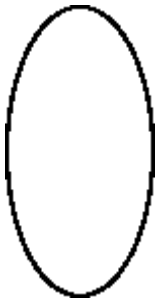
  ellipsoid axis length along z-axis

Default Formula: 1.0

The default lengths produce a sphere of diameter 1.

See Also: sphere-object

Figure 4-8. Ellipsoid-object



The boundary representation for an ellipsoid-object is the profile in the x, y plane.

extended-elbow-object [class]

The extended-elbow defines an elbow with extended straight sections on both sides of the bend. The elbow is defined by specifying the end points of the extended elbow and the normal into the extended elbow.

Inheritance: assembly-object

Properties:

- **objects-info**
- **bend-type**

  The type of elbow object to use for the bend. It can be 'elbow-object or 'pipe-elbow-object.

  Default Formula: 'elbow-object
- **run-type**

  The class of objects to use for the straightsection. It can be 'cylinder-object or 'pipe-object.

  Default Formula: 'cylinder-object
- **elbow-radius**

  The bend radius of the elbow.
- **thickness**

  The thickness of the pipe if pipe-object or pipe-elbowobject are used.

  Default Formula: 0.1
- **diameter**

  The outside diameter of the extended-elbow.

  Default Formula: 0.5
- **end-vector**

The normal to the end face.

Default Formula: (0.0 -1.0 0.0)

- **start-vector**

  The normal to the start face.

  Default Formula: (1.0 0.0 0.0)

- **end-point**

  The center point of the end face.

  Default Formula: (2.0 2.0 0.0)

- **start-point**

  The center point of the start face.

  Default Formula: (0.0 0.0 0.0)

- **run-1**

  the first straight section of the extended-elbow-object

- **bend**

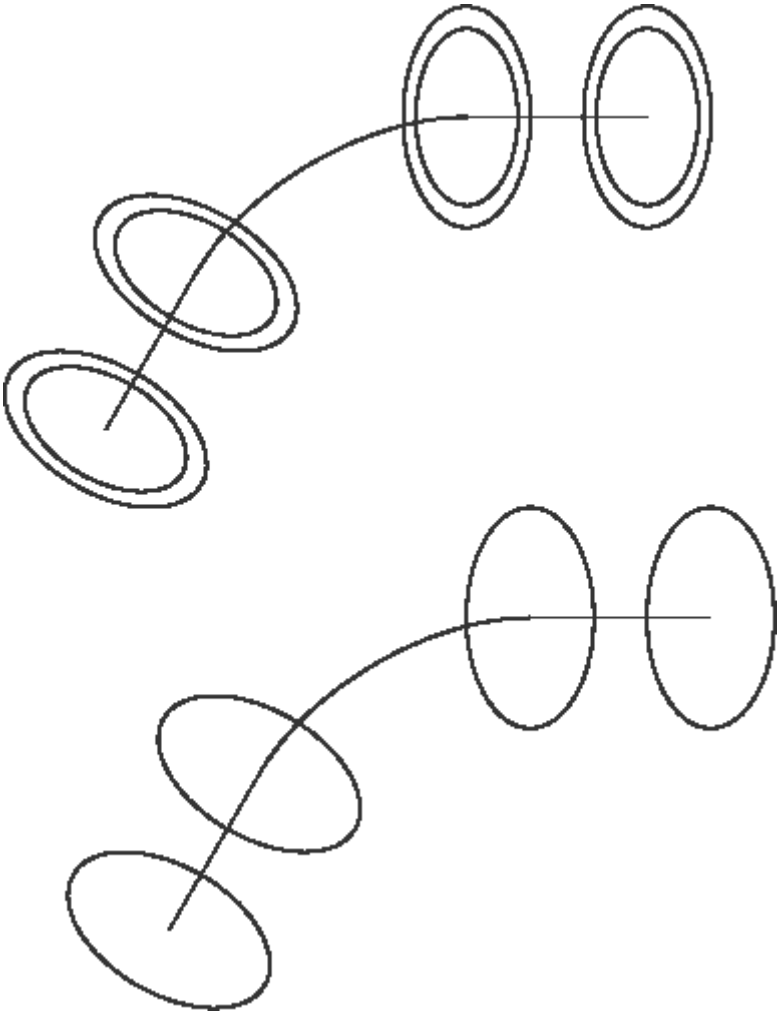
  the arched section of the extended-elbow-object

- **run-2**

  the second straight section of the extended-elbow-object

See Also: elbow-object, open-elbow-object, pipe-elbow-object, variable-pipe-elbow-object

Figure 4-9. Extended-elbow-object



The figure on top is with bend-type as 'pipe-elbow-object and 'run-type as 'pipe-object.

Example 4-26. Extended-elbow-object example

```
(define-class XELBOW-EXAMPLE
  :inherit-from (extended-elbow-object)
  :properties (
               elbow-radius 1.0
               start-point '(0 0 0)
               start-vector '(0 1 0)
               end-point '(2 2 0)
               end-vector '(-1 0 0)
               )
  )
```

line-object [class]

A line-object connects two points in 3-dimensional space.

Inheritance: primitive-class

Properties:

- **point1**

  May be a list or vector of x,y,z coordinates.

  Default Formula: '(-0.5 -0.5 -0.5)

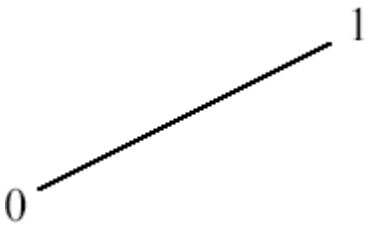
- **point2**

  May be a list or vector of x,y,z coordinates.

  Default Formula: '(0.5 0.5 0.5)

See Also: line-point-len-vec

Figure 4-10. Line-object



The line above displays the vertices of the object. The vertex number 0 corresponds to the point1 and the vertex number 1 corresponds to the point2.

line-point-len-vec [class]

A line-point-len-vec starts at one point and creates a line of the specified length in the specified direction.

Inheritance: edge-line

Properties:

- **original-point1**

  The location to start the line from.

- **vector**

  The direction the line is to extend from the point.

- **length**

  The length of the line to be created.

See Also: line-object

multiline-text-object [class]

This class creates a single object which contains multiple lines of text.

Inheritance: simple-geometry-class

Properties:

- **text-string**

  Contains the object text. Line changes are denoted by "\\" (e.g., "Line One\\Line Two\\Line Three").

- **text-info**

  A list of lists containing formatting information for the strings in text-string. The format of each entry is '(height font slant). If any of those values are nil, then the corresponding object property will be used (i.e., height, font-name, or slant).

- **height**

  The default text height if none is specified in text-info.

  Default Formula: 0.10

- **view-normal?**

  Determines if the text should always remain normal to the viewing plane.

- **font-name**

  The default font if none is specified in text-info.

  Default Formula: "stroked"

- **coordinates**

  The position of the object. The alignment of the entire object is about this point.

- **direction**

  A vector defining the direction of the text.

- **normal**

  A vector defining the direction from which text should be readable. This is ignored if view-normal? is t.

- **spacing**

  Spacing between characters in the text lines.

- **slant**

  Default text slant if none is specified in text-info.

- **alignment**

  The alignment for both individual lines and the entire "bounding box" of the multiline-text-object. The format is as a list of alignment keywords '(vertical horizontal). Valid vertical keywords are :top, :center, and :bottom. Valid horizontal keywords are :left, :center, and :right.

- **line-spacing**

  The vertical spacing between the bottoms of the lines given as a multiplier of text height (e.g., 2.0 gives double spacing and 1.0 will have the characters touching).

See Also: text-object

open-cone-object [class]

An open-cone-object is defined as an open ended hollow cone.

Inheritance: primitive-class

Properties:

- **diameter**

  The diameter of the end circle.

  Default Formula: 0.5

- **height**

  The height is defined parallel to the z-axis.

  Default Formula: 2.0

See Also: cone-object, open-truncated-cone-object

open-cylinder-object [class]

An open-cylinder-object is a open ended hollow cylinder.

Inheritance: primitive-class

Properties:

- **diameter**

  The diameter of the cylinder.

  Default Formula: 1.0

- **height**

  The height is defined parallel to the z-axis.

  Default Formula: 2.0

See Also: cylinder-object, pipe-object

open-elbow-object [class]

An open-elbow-object is a system class that will draw fast revolutions of a circle.

Inheritance: primitive-class

Properties:

- **angle**

  Default Formula: 90.0

- **elbow-radius**

  Default Formula: 1.0

- **diameter**

  Default Formula: 1.0

See Also: elbow-object, extended-elbow-object, pipe-elbow-object, variable-open-elbow-object

open-truncated-cone-object [class]

An open-truncated-cone-object is defined as an open ended hollow frustum of a cone.

Inheritance: primitive-class

Properties:

- **start-diameter**

  The diameter of the start face of the truncated cone.

  Default Formula: 1.0

- **end-diameter**

  The diameter of the end face of the truncated cone.

  Default Formula: 1.0

- **height**

  The height of the truncated cone.

  Default Formula: 2.0

See Also: cone-pipe-object, truncated-cone-object

pipe-elbow-object [class]

The pipe-elbow-object is an elbow with a thickness.

Inheritance: primitive-class

Properties:

- **angle**

  Default Formula: 90.0

- **elbow-radius**

  Default Formula: 1.0

- **thickness**

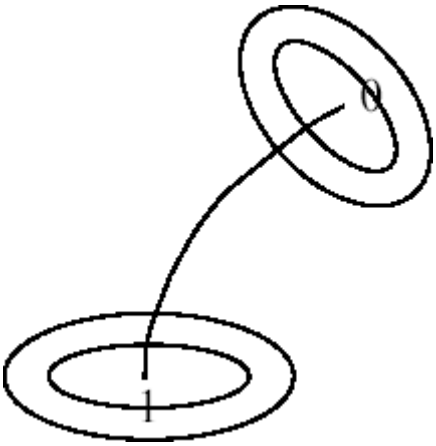
  Default Formula: 0.1

- **outer-diameter**

  Default Formula: 1.0

See Also: elbow-object, extended-elbow-object, open-elbow-object, variable-open-elbow-object, variable-pipe-elbow-object

Figure 4-11. Pipe-elbow-object



The pipe elbow above shows the faces of the pipe-elbow-object. The 0 face is the face that lies in the x, y plane when the object is not oriented.

pipe-object [class]

A pipe-object is a class that has been created as a cylinder with a thickness.

Inheritance: primitive-class

Properties:

- **outer-diameter**

  The diameter of the cylinder.

  Default Formula: 1.0

- **thickness**

  The difference between the inner and outer radii (the wall thickness).

  Default Formula: 0.01

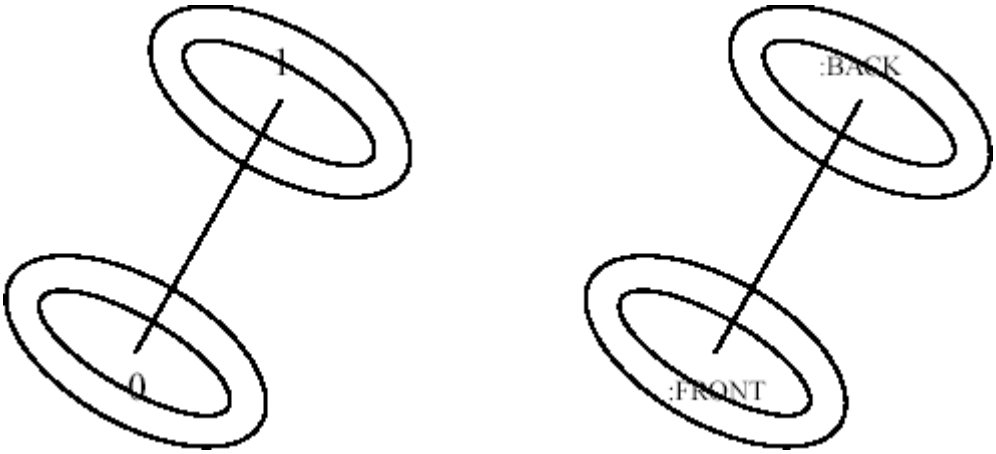
- **height**

  The height is defined parallel to the z-axis.

  Default Formula: 2.0

See Also: open-cylinder-object

Figure 4-12. Pipe-object



The pipe on the left shows the system default names for the faces and edges of the pipe. The pipe on the right shows the names mapped by get-face-id.

point-object [class]

A point-object is an object that defines a location in 3-dimensional space. It is represented graphically as a + at the point defined by the coordinates property.

Inheritance: primitive-class

Properties:

- **coordinates**

  May be a list or vector of x,y,z coordinates.

  Default Formula: (vector 0.0 0.0 0.0)

polygon-object [class]

The polygon-object is a polygon composed of line segments. If the vertices do not specify a closed polygon, the system will automatically connect the start and end points.

Inheritance: polyline-object

Properties:

- **vertices**

  The list of points defining the segments of the poly line-object.

- **dimension**

  The dimension determines if the object is a 1d entity (an outline) or a 2d entity (a surface).

  Default Formula: 1 for 1d. If 2, the vertices must form a planar surface.

See Also: polyline-object

Example 4-27. Polygon-object

```
(define-class PATH
```

```
  :inherit-from (polygon-object)
  :properties (
               vertices '((0.0 0.0 0.0) (1.0 0.0 0.0)
                          (1.0 1.0 0.0) (0.0 1.0 0.0))
               dimension 2
               )
  )
```

polyline-object [class]

The polyline-object is a line composed of line segments.

Inheritance: geom-object

Properties:

- **vertices**

  The list of points defining the segments of the poly line-object.

  Default Formula: nil

See Also: polygon-object

Example 4-28. Polyline-object example

```
(define-class PATH
   :inherit-from (polyline-object)
   :properties (
                vertices '((0.0 0.0 0.0) (1.0 0.0 0.0)
                           (1.0 0.0 1.0) (1.0 1.0 1.0))
     )
 )
```

rectangle-class [class]

The rectangle-class creates a rectangle of dimension 1 (wire) or 2 (sheet) centered at the origin in the XY plane similar to a sheet-object.

Inheritance: primitive-class

Properties:

- **height**

  Dimension along the Y axis.

  Default Formula: 1.0

- **width**

  Dimension along the X axis.

  Default Formula: 1.0

- **Dimension**

  Default Formula: 1 (1 for wire, 2 for sheet)

sheet-object [class]

A filled rectangle defined in the x-y plane in 3-dimensional space. A sheet-object has area.

Inheritance: primitive-class

Properties:

- **width**

  The width is defined parallel to the x-axis.

  Default Formula: 1.0

- **height**

  The height is defined parallel to the y-axis.

  Default Formula: 1.0

sphere-object [class]

The sphere-object may be defined as a geometric shell or solid. Setting the solid? property to true will create a solid.

Inheritance: primitive-class, solid-object

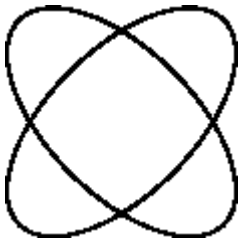
Properties:

- **diameter**

  Default Formula: 1.0

See Also: ellipsoid-object

Figure 4-13. Sphere-object



The graphic above shows the simple representation of the sphere.

text-object [class]

The text-object is used to place a text string on the display window. The text may be displayed so that the string is maintained normal to the view for readability or the text may move with the view (which may be difficult to read in some views).

Inheritance: primitive-class

Properties:

- **text-string**

The value must be a string that is to be displayed on the screen. The default value is to search up the instance tree for a text-string. If no text-string is found the name of the text-object instance is used.

- **view-normal?**

  This property determines if the text-object should always remain normal to the viewing plane.

  Default Formula: t

- **height**

  The height of the text.

  Default Formula: 0.1

- **coordinates**

  The center of the text.

  Default Formula: '(0.0 0.0 0.0)

- **direction**

  A vector that defines the direction of text.

  Default Formula: The default is '(1.0 0.0 0.0). This is ignored if view-normal? is t.

- **normal**

  A vector that defines the direction from which the text is readable.

  Default Formula: The default is '(0.0 0.0 1.0). This is ignored if view-normal? is t.

- **font-name**

  The font used to display the text.

  Default Formula: 'stroked

- **spacing**

  Additional space added between characters.

  Default Formula: 0.0

- **slant**

  The angle at which to slant characters (to produce "italicized" text).

  Default Formula: 0.0

- **alignment**

  A list of arguments which specify how the text should be aligned around the coordinates. The format is '(vertical horizontal), where the vertical value can be :top, :center, or :bottom and the horizontal value can be :left, :center, or :right.

  Default Formula: '(:center :center)

See Also: multiline-text-object

Example 4-29. Text-object example

```
(define-class TEXT-EXAMPLE
  :inherit-from (object)
  :properties (
```

```
              size 1.0
              )
  :subobjects (
              (part :class 'box-object
                    height ^^size
                    width ^^size
                    depth ^^size
                    )
              (label :class 'text-object
                     height (half ^^size)
                     text-string (format nil "~a" ^^size)
                     )
              )
  )
```

torus-class [class]

A torus-class is used to create a torus (a surface having genus one). It is controlled by minor-radius(the inner radius of the torus) and major-radius (the outer radius of the torus).

Inheritance: graphic-object

Properties:

- **minor-radius**

  Inner radius of the torus.

  Default Formula: 1.0

- **major-radius**

  Outer radius of the torus.

  Default Formula: 2.0

```
minor-radius must be > 0, major-radius != minor-radius, and
(major-radius+minor-radius) must be > 0
if major-radius = 0, sphere
if major-radius < 0, inner surface of torus (lemon torus)
if 0 < major-radius < minor-radius, outer surface of torus (apple torus)
if major-radius > minor-radius, full torus

torus will be created as centered at (0 0 0) and facing (0 0 1)
```

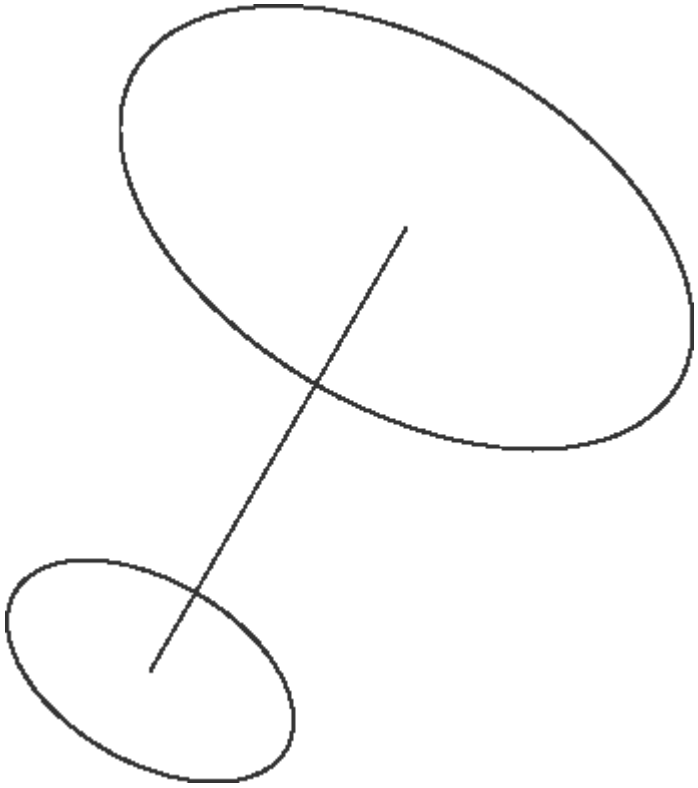
truncated-cone-object [class]

A truncated-cone-object is the capped version of the open-truncated-cone-object. It can be a hollow shell by setting the solid? property to nil.

Inheritance: open-truncated-cone-object, solid-object

See Also: cone-pipe-object, open-truncated-cone-object

Figure 4-14. Truncated-cone-object



A truncated-cone-object.

variable-open-elbow-object [class]

This class creates an open elbow that has different diameters at each end.

Inheritance: primitive-class

Properties:

- **bend-radius**

  The radius of the center line of the elbow.

  Default Formula: 2.0

- **start-diameter**

  The starting diameter of the elbow.

  Default Formula: 1.0

- **end-diameter**

  The ending diameter of the elbow.

  Default Formula: 2.0

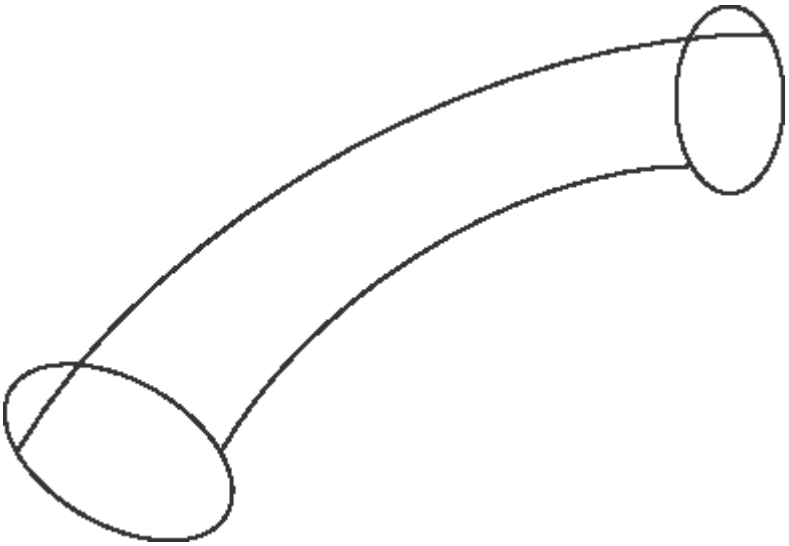
- **bend-angle**

  The sweep angle of the elbow.

Default Formula: 90.0

See Also: elbow-object, extended-elbow-object, open-elbow-object, open-truncated-cone-object, variable-pipe-elbow-object

Figure 4-15. Variable-open-elbow-object



An open-elbow-object with different diameters at both ends.

Example 4-30. Variable-open-elbow-object example

```
(define-class ELBOW
  :inherit-from (variable-open-elbow-object)
  :properties (
               bend-radius (default 1.0)
               start-diameter (default (/ 3.0 8.0))
               end-diameter (default (/ 1.0 2.0))
               bend-angle (default 90.0)
               )
  )
```

variable-pipe-elbow-object [class]

This class creates a pipe elbow that has different diameters at each end.

Inheritance: primitive-class

Properties:

- **bend-radius**

  The radius of the center line of the elbow.

  Default Formula: 2.0

- **start-diameter**

  The starting diameter of the elbow.

  Default Formula: 1.0

- **end-diameter**

  The ending diameter of the elbow.

  Default Formula: 2.0

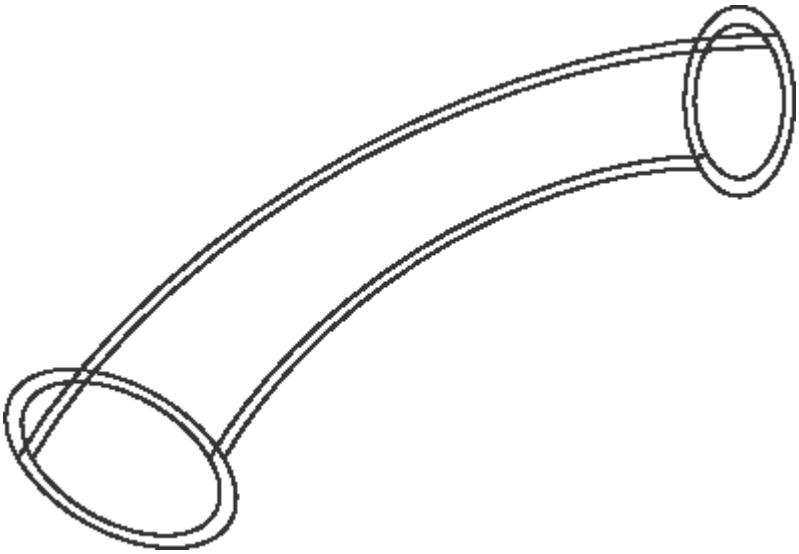
- **bend-angle**

  The sweep angle of the elbow.

  Default Formula: 90.0

See Also: cone-pipe-object, pipe-elbow-object, variable-open-elbow-object

Figure 4-16. Variable-pipe-elbow-object



A pipe-elbow-object with different diameters at both ends.

Example 4-31. Variable-pipe-elbow-object example

```
(define-class ELBOW
  :inherit-from (variable-pipe-elbow-object)
  :properties (
               bend-radius (default 1.0)
               start-diameter (default (/ 3.0 8.0))
               end-diameter (default (/ 1.0 2.0))
               bend-angle (default 90.0)
               thickness (default 0.035)
               )
  )
```

vector-class [class]

This class creates a vector geometry including an arrow head.

Properties:

- **base-point**

Coordinates of the start point of the vector.

Default Formula: '(0.0 0.0 0.0)

- **direction**

  List of the X,Y,Z components of the direction vector.

  Default Formula: '(1.0 0.0 0.0)

- **length**

  Specifies the length of the vector.

  Default Formula: 1.0

- **color**

  Default Formula: 'green

Tangents

These objects create lines, arcs, or circles which are tangent to other lines, arcs, or circles. The following objects are available:

arc-ln-pt-tg - an arc tangent to a line and through a point

arc-cl-cl-tg - an arc tangent to two circles or arcs

arc-ln-cl-tg - an arc tangent to a line and a circle or arc

arc-ln-ln-tg - an arc tangent to two lines

arc-tg - an arc tangent to combinations of two objects of types circle, arc, line, and point

circle-cl-cl-tg - a circle tangent to two circles or arcs

circle-ln-cl-tg - a circle tangent to a line and a circle or arc

circle-ln-ln-tg - a circle tangent to two lines

circle-tg - a circle tangent to combinations of two objects of types circle, arc, line, and point

line-cl-cl-tg - a line tangent to two circles or arcs

line-pt-cl-tg - a line through a point and tangent to a circle or arc

In all cases, arc- and circle-objects may be used interchangeably to specify tangent objects. In general, there will be more than one object tangent to any pair of line-, circle-, or arc-objects. For example, there may be four lines tangent to two circles or as many as eight circles tangent to two circles. The tangent objects employ a common method of selecting which case is the correct one. The selection is handled through four properties common to all of the objects:

selection-info

selection-method

hold-relationship

ignore-hold-relationship

The selection-method contains the name of the method which will be called to differentiate between the different possible tangents. Selection-info contains information required by the selection method to

make the selection. Hold-relationship governs when selections may be made. After a selection has been made, its relationship to the related objects will be held even if the geometry of the related objects and thus of the object itself changes. Unless holdrelationship is nil, any changes made to selection-method and selection-info will be ignored. When hold-relationship is made nil, any changes in geometry required by the current values of the selection properties will be made and hold-relationship will be set t. This selection process was implemented because some selection methods may depend on situations which do not remain valid after objects are moved or otherwise modified. The behavior of holdrelationship can be turned off with the ignore-hold-relationship property. When this property is nil, hold-relationship functions as described. When t, the selection is always adjusted when properties change. As an example of the selection process, consider the case of a line tangent to two circles. Depending on the relative positions of the circles, there may be as many as four lines tangent to these circles. Two of the lines are internal (they cross over each other between the circles) and two are external. A possible selection method in this case is lcct-select-index. This method requires an index from 0 to 3 which specifies which of the four lines should be used for the tangent object. The index value is stored in the property selection-info. When the value of holdrelationship is made nil the next demand on the object will cause the selection of the appropriate line. Hold-relationship will automatically be set to t to retain the selection. The index value will be used even when the circles are moved. In other words, once the tangent line with index 0 (one of the external tangents) is chosen, any changes made to the geometry of the circles will not affect the selection. However, the actual endpoints of the line will change to remain tangent to the circles. All objects have at least two available selection methods. The first, tg-select-index, is an indexed method which allows the appropriate tangent to be chosen directly. The other, tgselect- interactive, is an interactive method which allows the desired tangent to be selected from the display. User-developed selection methods may be used in place of the predefined methods.

Classes and Methods

arc-cl-cl-tg [class]

The arc-cl-cl-tg object defines an arc tangent to two circles or arcs. The tangent arc will be one of eight possible arcs.

Inheritance: arc-ctr-radius-sta-enda, basic-tangent-object

Properties:

- **radius**

  The radius of the tangent arc.

- **circle-1**

  The first circle or arc to be tangent to (arc-object or circle-object).

- **circle-2**

  The second circle or arc to be tangent to (arc-object or circle-object).

- **complement**

  If t, draw the complement to the standard arc.

- **selection-info**

  Information governing the choice of tangent circle.

- **selection-method**

  The name of the method used to select the tangent circle. The possible values are: 'tg-select-index and 'tg-select-interactive. The **tg-select-index** method will choose which arc to use based on a simple index. The eight possible arcs are numbered 0 through 7. The selection-info contains this index. selection-info - a single number indicating the desired tangent arc. The **tg-select-interactive** method will display the center locations of any of the eight possible tangent arcs which are valid for the user to select interactively. selection-info - doesn't matter.

- **hold-relationship**

  Determines whether to change selection based on method and info.

- **ignore-hold-relationship**

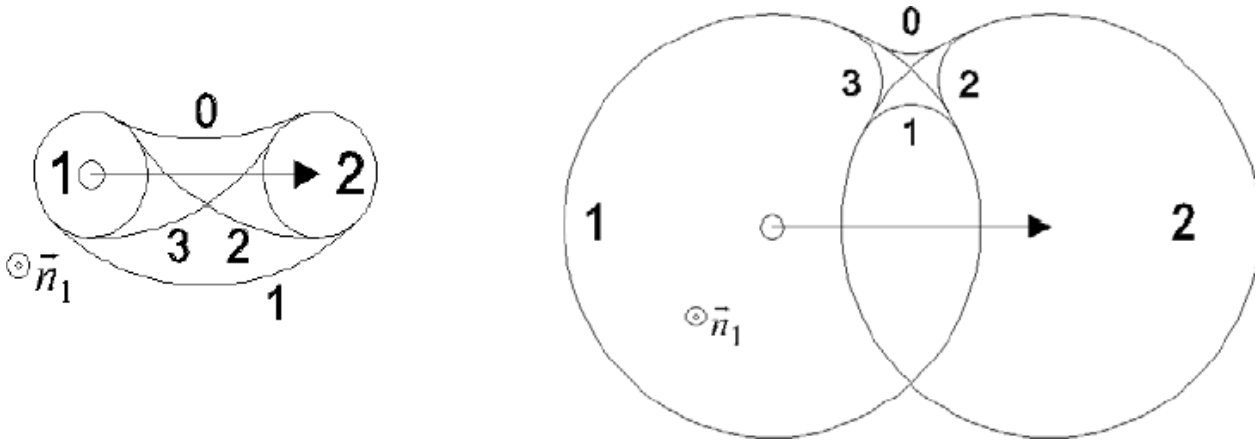
  Used to turn off the hold-relationship behavior (default nil, use hold-relationship).

  Default Formula: nil

- **center-data**

  A list containing the centers of all of the possible tangent arcs.

Figure 4-17. Arc-cl-cl-tg



The index of the tangent arc is relative to the vector from the center of circle 1 to the center of circle 2 and the normal of circle 1. The figures show the tangent arcs with indices 0 through 3. Tangent arcs 4 through 7 are the mirror reflections on the other side of the center-center vector.

arc-ln-cl-tg [class]

The arc-ln-cl-tg object defines an arc tangent to a line and a circle or arc. The tangent arc will be one of eight possible arcs.

Inheritance: arc-ctr-radius-sta-enda, basic-tangent-object

Properties:

- **radius**

  The radius of the tangent arc.

- **tline**

The line to be tangent to (line-object).

- **tcircle**

  The arc or circle to be tangent to (arc-object or circle-object).

- **complement**

  If t, draw the complement to the standard arc.

  Default Formula: nil

- **selection-info**

  Information governing the choice of tangent circle.

- **selection-method**

  The name of the method used to select the tangent circle. The possible values are: 'tg-select-index and 'tg-select-interactive. The **tg-select-index** method will choose which arc to use based on a simple index. The eight possible arcs are numbered 0 through 7. The selection-info contains this index. selection-info - a single number indicating the desired tangent arc. The **tg-select-interactive** method will display the center locations of any of the eight possible tangent arcs which are valid for the user to select interactively. selection-info - doesn't matter.

- **hold-relationship**

  Determines whether to change selection based on method and info.

- **ignore-hold-relationship**

  Used to turn off the hold-relationship behavior (default nil, use hold-relationship).

  Default Formula: nil

- **arc-data**

  A list containing information about all of the possible tangent arcs. The information for each arc consists of the center of the arc and contact points with the line and circle.

Example 4-32. Arc-ln-cl-tg Example

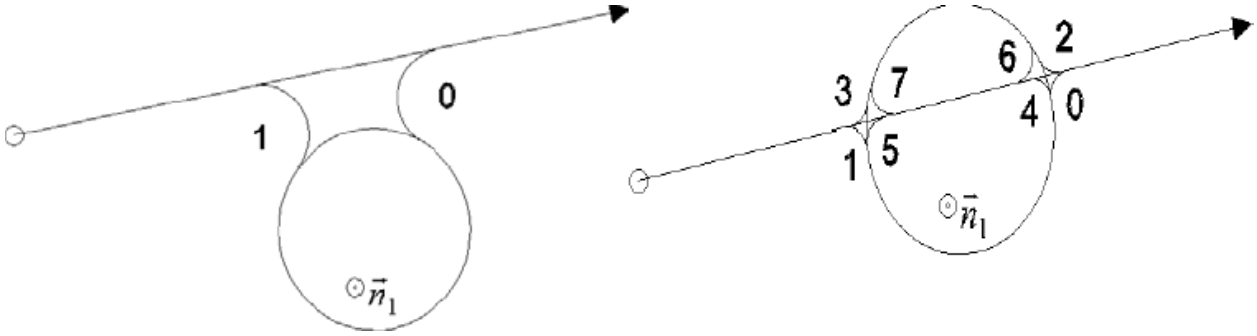
```
;;;; ------------------------------------------------------------------------
--
;;;;                                                    11180 Reed Hartman
Hwy
;;;;               TechnoSoft Inc.                      Cincinnati, OH
45242
;;;;             Copyright (c) 1993 - 2009              Phone: (513) 985-
9877
;;;;                                                    Fax:   (513) 985-
0522
;;;;
;;;;
;;;; This code sample is provided as an example of how to perform a
particular
;;;; kind of task using foundation software provided by TechnoSoft.  It is
;;;; provided "as is" without warranty of any kind, expressed or implied.
;;;;
;;;; Licensed users of the Adaptive Modeling Language are free to use and
```

```
;;;; modify this code as long as the the original credits and disclaimers are
;;;; maintained.
;;;; ------------------------------------------------------------------------
--
;;;;  Author: Trapper Schuler
;;;;  Created: Mon Jul 06 20:52:17 2009
;;;; ------------------------------------------------------------------------
--
;;;;  Purpose: arc-ln-cl-tg class examples.
;;;;

(in-package :aml)

;;;-----------------------------------------------------
;;; Class   : arc-ln-cl-tg-example-1
;;; Inherit :
;;; Purpose :
;;; Notes   :
;;; Author  :
;;; History
;;;   Created on :
;;;   Modified   :
;;;

(define-class arc-ln-cl-tg-example-1
  :inherit-from (object)
  :properties (
               radius 0.5
               complement? nil
               )
  :subobjects (
               (top-line-object :class line-object
                                point1 (list -2 1 0)
                                point2 (list 2 1 0)
                                )
               (bottom-line-object :class line-object
                                   point1 (list -2 -1 0)
                                   point2 (list 2 -1 0)
                                   )
               (circle-object :class circle-object)
               (arc-object-0 :class arc-ln-cl-tg
                             radius ^^radius
                             tline ^^top-line-object
                             tcircle ^^circle-object
                             selection-info 0
                             complement ^^complement?
                             color 'cyan
                             )
               (arc-object-1 :class arc-ln-cl-tg
                             radius ^^radius
                             tline ^^top-line-object
                             tcircle ^^circle-object
                             selection-info 1
```

```
                              complement ^^complement?
                              color 'magenta
                              )
                (arc-object-2 :class arc-ln-cl-tg
                              radius ^^radius
                              tline ^^bottom-line-object
                              tcircle ^^circle-object
                              selection-info 2
                              complement ^^complement?
                              color 'orange
                              )
                (arc-object-3 :class arc-ln-cl-tg
                              radius ^^radius
                              tline ^^bottom-line-object
                              tcircle ^^circle-object
                              selection-info 3
                              complement ^^complement?
                              color 'green
                              )
                )
  )

(define-method property-classification-list arc-ln-cl-tg-example-1 ()
  '(
    ("Main Properties"
     (
      (radius (automatic-apply? t) ui-property-field-class)
      (complement? (automatic-apply? t) ui-toggle-property-button-class)
      )
     )
    )
  )

;;;-------------------------------------------------------
;;; Class   : arc-ln-cl-tg-example-2
;;; Inherit :
;;; Purpose :
;;; Notes   :
;;; Author  :
;;; History
;;;   Created on :
;;;   Modified   :
;;;

(define-class arc-ln-cl-tg-example-2
  :inherit-from (series-object)
  :properties (
               radius 0.15
               complement? nil

               ;; Series Properties
               init-form '(
                           radius ^^radius
```

```
                              complement ^^complement?
                              tline ^^line-object
                              tcircle ^^circle-object
                              selection-info ^index
                              color (get-random-color)
                              )
                    quantity 8
                    class-expression 'arc-ln-cl-tg
                    series-prefix 'arc-ln-cl-tg
                    recreate? nil
                    )
  :subobjects (
                    (line-object :class line-object
                                 point1 (list -2 0 0)
                                 point2 (list 2 0 0)
                                 )
                    (circle-object :class circle-object)
                    )
  )

(define-method property-classification-list arc-ln-cl-tg-example-2 ()
  '(
    ("Main Properties"
     (
      (radius (automatic-apply? t) ui-property-field-class)
      (complement? (automatic-apply? t) ui-toggle-property-button-class)
      )
     )
    )
  )
```

Figure 4-18. Arc-ln-cl-tg



The index of the tangent arc is relative to the orientation of the line and circle and the normal of the circle.

arc-ln-ln-tg [class]

The arc-ln-ln-tg object defines an arc tangent to two lines. The tangent arc will be one of four possible arcs.

Inheritance: arc-ctr-radius-sta-enda, basic-tangent-object

Properties:

- **radius**

  The radius of the tangent arc.

- **line-1**

  The first line to be tangent to (line-object).

- **line-2**

  The second line to be tangent to (line-object).

- **complement**

  If t, draw the complement to the standard arc.

  Default Formula: nil

- **selection-info**

  Information governing the choice of tangent circle.

- **selection-method**

  The name of the method used to select the tangent circle. The possible values are: 'tg-select-index or 'tg-select-interactive or 'allt-select-point. The **tg-select-index** method will choose which arc to use based on a simple index. The four possible arcs are numbered 0 through 3. The selection-info contains this index. selection-info - a single number indicating the desired tangent arc. The **tg-select-interactive** method will display the center locations of any of the four possible tangent arcs which are valid for the user to select interactively. selection-info - doesn't matter. The **allt-select-point** method is used to select the required arc based on the location of a given point. The two lines divide the plane into four regions. The tangent arc will be in the same region as the point. selection-info - a list containing the x,y,z coordinates of the selection point.

- **hold-relationship**

  Determines whether to change selection based on method and info.

- **ignore-hold-relationship**

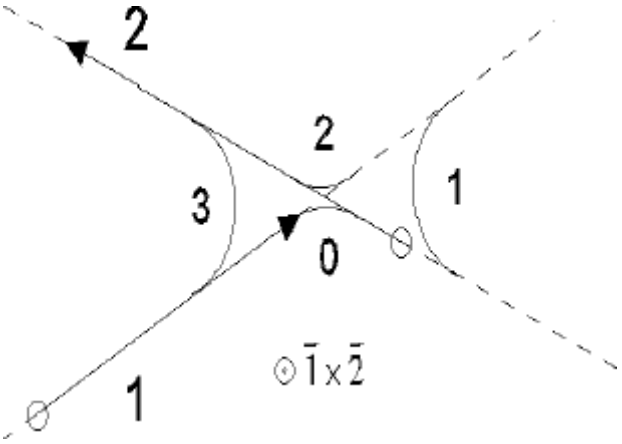
  Used to turn off the hold-relationship behavior (default nil, use hold-relationship).

  Default Formula: nil

- **arc-data**

  A list containing lists which define the geometry of the tangent arcs.

Figure 4-19. Arc-ln-ln-tg



The index of the tangent arc is relative to the orientation of the lines and the cross product of the lines.

arc-ln-pt-tg [class]

The arc-ln-pt-tg object defines an arc tangent to a line and ending at a specified point. The tangent arc will be one of two possible arcs.

Inheritance: arc-ctr-radius-sta-enda, basic-tangent-object

Properties:

- **radius**

  The radius of the tangent arc.

- **tline**

  The line to be tangent to (line-object).

- **tpoint**

  The point specifying the other end of the arc (either a list or a point-object).

- **complement**

  If t, draw the complement to the standard arc.

  Default Formula: nil

- **selection-info**

  Information governing the choice of the tangent arc.

- **selection-method**

  The name of the method used to select the tangent arc. The possible values are: 'tg-select-index or 'tg-select-interactive. The **tg-select-index** method will choose which arc to use based on a simple index. The two possible arcs are numbered 0 through 1. The selection-info contains this index. selection-info - a single number indicating the desired tangent arc. The **tg-select-interactive** method will display the center locations of any of the two possible tangent arcs which are valid for the user to select interactively. selection-info - doesn't matter.

- **hold-relationship**

Determines whether to change the selection based on method and info.

- **ignore-hold-relationship**

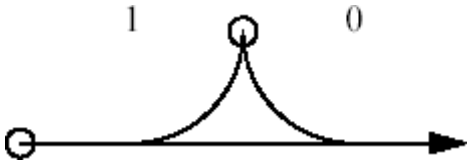
  Used to turn off the hold-relationship behavior.

  Default Formula: nil

- **arc-data**

  A list containing information on the two possible arcs.

Figure 4-20. Arc-ln-pt-tg



The index of the tangent arc is relative to the orientation of the line and point.

arc-tg [class]

The arc-tg object defines an arc tangent to a combination of line-, circle-, and arc-objects. The combination of line-object and point-object can also be used.

Inheritance: arc-ctr-radius-sta-enda, basic-tangent-object

Properties:

- **object-1**

  The first object to be tangent to. May be of type circle-object, arc-object, line-object, or point-object. If object-2 is a point-object then object-1 must be a line-object.

- **object-2**

  The second object to be tangent to. May be from same types as object-1. If object-1 is a point-object, then object-2 must be a line-object.

- **radius**

  The radius for the tangent arc.

- **complement**

  If t, draw the complement to the standard arc.

  Default Formula: nil

- **selection-info**

  Information governing the choice of the tangent arc.

- **selection-method**

  The name of the method used to select the tangent arc. The possible values are: 'tg-select-index or 'tg-select-interactive. The **tg-select-index** method will choose which arc to use based on a simple index. The selection-info property contains this index. selection-info - a single number indicating the desired tangent arc. The **tg-select-interactive** method will display the center locations of possible tangent arcs for the user to select interactively. selection-info - doesn't matter.

- **hold-relationship**

  Determines whether to change the selection based on method and info.

- **ignore-hold-relationship**

  Used to turn off the hold-relationship behavior.

  Default Formula: nil

- **arc-data**

  A list containing information on the possible arcs.

The configuration(the figure) of the possible tangent arcs is the same as for the equivalent individual tangent objects: arc-ln-ln-tg, arc-ln-cl-tg, arc-cl-cl-tg, and arc-ln-pt-tg.

basic-tangent-object [class]

All of the tangent objects inherit from basic-tangent-object. This provides them with methods for selecting appropriate tangents as well as a common base for object selection. This object is not intended to be used by itself. In addition, the methods written on this class are referenced in properties of tangent objects and not called directly on objects.

Inheritance: object

tg-select-interactive [Method]

Defined on Classes:

basic-tangent-object

This method provides tangent objects with the ability to have the user interactively select which of the possible tangents to use.

Arguments:

- **instance**

  The tangent object

tg-select-index [Method]

Defined on Classes:

basic-tangent-object

This method provides tangent objects withthe ability to select amongst the possible tangents by index. The object property selection-info holds the index.

Arguments:

- **instance**

  The tangent object

circle-cl-cl-tg [class]

The circle-cl-cl-tg object defines a circle tangent to two circles or arcs. The tangent circle will be one of eight possible circles.

Inheritance: circle-ctr-nml-diameter, basic-tangent-object

Properties:

- **diameter**

  The diameter of the tangent circle.

- **circle-1**

  The first circle or arc to be tangent to (circle-object or arc-object).

- **circle-2**

  The second circle or arc to be tangent to (circle-object or arc-object).

- **selection-info**

  Information governing the choice of tangent circle.

- **selection-method**

  The name of the method used to select the tangent circle. The possible values are: 'tg-select-index or 'tg-select-interactive. The **tg-select-index** method will choose which circle to use based on a simple index. The eight possible circles are numbered 0 through 7. The selection-info contains this index. The external tangents correspond to indices 0, 1, 4, and 5 and the internal tangents to 2, 3, 6, and 7. selection-info - a single number indicating the desired tangent arc. The **tg-select-interactive** method will display the center locations of any of the eight possible tangent circles which are valid for the user to select interactively. selection-info - doesn't matter.

- **hold-relationship**

  Determines whether to change selection based on method and info.

- **ignore-hold-relationship**

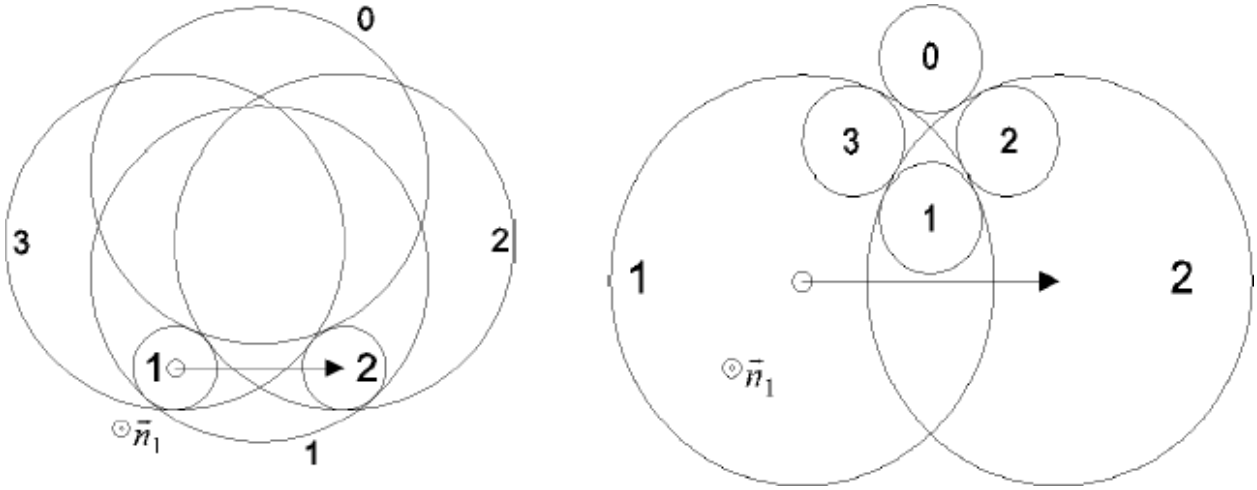
  Used to turn off the hold-relationship behavior (default nil, use hold-relationship).

  Default Formula: nil

- **center-data**

  A list containing the center-points of the tangent circles.

Figure 4-21. circle-cl-cl-tg



The index of the tangent circle is relative to the vector from the center of circle 1 to the center of circle 2 and the normal of circle 1. The figures show the tangent circles with indices 0 through 3. Tangent circles 4 through 7 are the mirror reflections on the other side of the center vector.

circle-ln-cl-tg [class]

The circle-ln-cl-tg object defines a circle tangent to a line and a circle or arc. The tangent circle will be one of eight possible circles.

Inheritance: circle-ctr-nml-diameter, basic-tangent-object

Properties:

- **diameter**

  The diameter of the tangent circle.

- **tcircle**

  The circle or arc to be tangent to (circle-object or arc-object).

- **tline**

  The line to be tangent to (line-object).

- **selection-info**

  Information governing the choice of tangent circle.

- **selection-method**

  The name of the method used to select the tangent circle. The possible values are: 'tg-select-index or tg-select-interactive. The **tg-select-index** method will choose which circle to use based on a simple index. The eight possible circles are numbered 0 through 7. The selection-info contains this index. selection-info - a single number indicating the desired tangent arc. The **tg-select-interactive** method will display the center locations of any of the eight possible tangent circles which are valid for the user to select interactively. selection-info - doesn't matter.

- **hold-relationship**

Determines whether to change selection based on method and info.

- **ignore-hold-relationship**

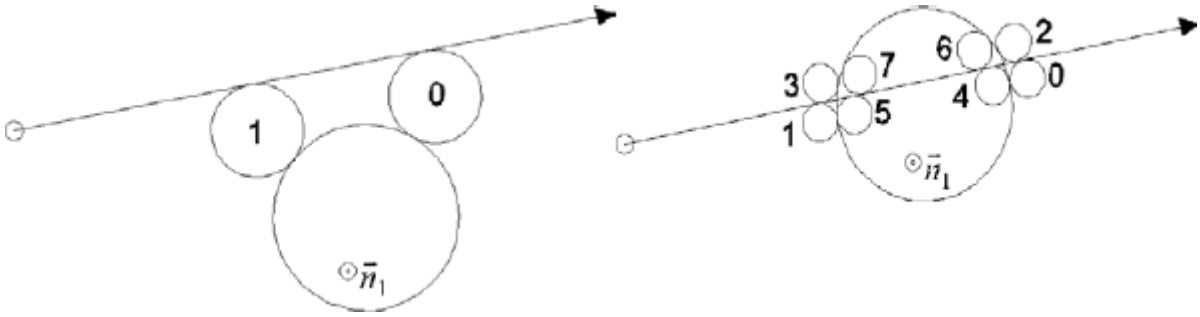
  Used to turn off the hold-relationship behavior (default nil, use hold-relationship).

  Default Formula: nil

- **center-data**

  A list containing the center points of the tangent circles.

Figure 4-22. Circle-ln-cl-tg



The index of the tangent circle is relative to the orientation of the line and circle and the normal of the circle.

circle-ln-ln-tg [class]

The circle-ln-ln-tg object defines a circle tangent to two lines. The tangent circle will be one of four possible circles.

Inheritance: circle-ctr-nml-diameter, basic-tangent-object

Properties:

- **diameter**

  The diameter of the tangent circle.

- **line-1**

  The first line to be tangent to (line-object).

- **line-2**

  The second line to be tangent to (line-object).

- **selection-info**

  Information governing the choice of tangent circle.

- **selection-method**

  The name of the method used to select the tangent circle. The possibe values are: 'tg-select-index or 'tg-select-interactive. The **tg-select-index** method will choose which circle to use based on a simple index. The four possible circles are numbered 0 through 3. The selection-info contains this index. selection-info - a single number indicating the desired tangent arc. The **tg-select-interactive** method

will display the center locations of any of the four possible tangent circles which are valid for the user to select interactively. selection-info - doesn't matter.

- **hold-relationship**

  Determines whether to change selection based on method and info.

- **ignore-hold-relationship**

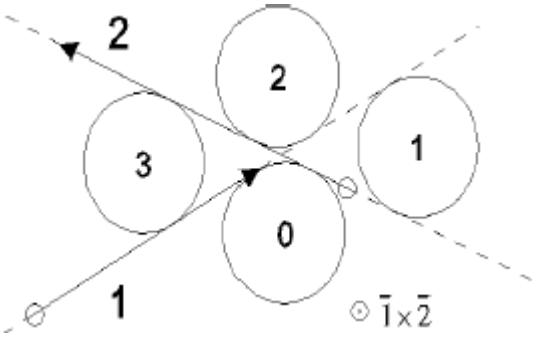
  Used to turn off the hold-relationship behavior (default nil, use hold-relationship).

  Default Formula: nil

- **center-data**

  A list containing the center points of the tangent circles.

Figure 4-23. circle-ln-ln-tg



The index of the tangent circle is relative to the orientation of the lines and the cross product of the lines.

circle-tg [class]

The circle-tg object defines an arc tangent to a combination of line, circle, and arc-objects.

Inheritance: circle-ctr-nml-diameter, basic-tangent-object

Properties:

- **object-1**

  The first object to be tangent to. May be of type circle-object, arc-object, or line-object.

- **object-2**

  The second object to be tangent to. May be from same types as object-1.

- **radius**

  The radius for the tangent circle.

- **selection-info**

  Information governing the choice of the tangent circle.

- **selection-method**

  The name of the method used to select the tangent circle. The possible values are: 'tg-select-index or 'tg-select-interactive. The **tg-select-index** method will choose which circle to use based on a simple index. The selection-info property contains this index. selection-info - a single number indicating the

desired tangent arc. The **tg-select-interactive** method will display the center locations of the possible tangent circles for the user to select interactively. selection-info - doesn't matter.

- **hold-relationship**

  Determines whether to change the selection based on method and info.

- **ignore-hold-relationship**

  Used to turn off the hold-relationship behavior.

  Default Formula: nil

- **center-data**

  A list containing information on the possible circles.

The configuration of the possible tangent arcs is the same as for the equivalent individual tangent objects:circle-ln-ln-tg, circle-ln-cl-tg, and circle-cl-cl-tg.

line-cl-cl-tg [class]

Inheritance: line-object basic-tangent-object

Properties:

- **circle-1**

  The first circle or arc to be tangent to (circle-object or arc-object).

- **circle-2**

  The second circle or arc to be tangent to (circle-object or arc-object).

- **selection-info**

  Information governing the choice of tangent lines.

- **selection-method**

  The name of the method used to select tangent line. The possible values are: 'lcct-select-two-points or 'lcct-select-side-type or 'lcct-select-two-sides or 'tg-select-interactive or 'tg-select-index. The **lcct-select-two-points** method will choose the tangent line based on two points. Consider two circles connected by a vector from the center of circle-1 to the center of circle-2. The first selection point will determine which side of the "center-vector" the first endpoint of the tangent line is on. The second selection point will determine which side of the "center-vector" the second endpoint of the tangent is on. By using this method, the selection of the tangent line can depend on the geometry of other objects. selection-info - a list of two points (each a list of x,y,z coordinates). The **tg-select-interactive** method will display any possible tangent lines (either two or four) for the user to select interactively. Selection is made by the tangent point on the first circle. selection-info - doesn't matter. The **tg-select-index** method will choose which tangent line to use based on a simple index. The four possible lines are numbered 0 through 3. The selection-info contains this index. The external tangents correspond to indices 0 and 1, the internal tangents to 2 and 3. selection-info - a single number indicating the desired tangent arc. The **lcct-select-side-type** This method will choose which side of the first circle the tangent line touches and the type of tangent line based on two values given in selection-info. The first value indicates the side of the first circle that the line is tangent to in a manner similar to that described for lcct-selecttwo-sides. The second value determines whether the tangent is external or internal (crossover). A positive value will create an external tangent while a negative value will create an internal tangent. selection-info - a list of two numbers, the signs of which determine where the

tangent line starts and whether it is an internal or an external tangent. The **lcct-select-two-sides** method will choose which side of each circle the tangent line touches. The selection info contains two numbers which indicate which side of the circles should be used. These numbers indicate a position based on the vector connecting the centers of the two circles and the normal to the first circle. A positive value indicates the side of the "center-vector" given by a positive rotation from that vector about the normal to the circle. A negative value is the other side. For the second circle, the rotation is still about the normal of the first circle (i.e., for both circles, the same sign in selection info indicates the same side of the "center-vector"). selection-info - a list of two numbers, the signs of which determine where the tangent line is drawn.

- **hold-relationship**

  Determines whether to change selection based on method and info.

- **ignore-hold-relationship**

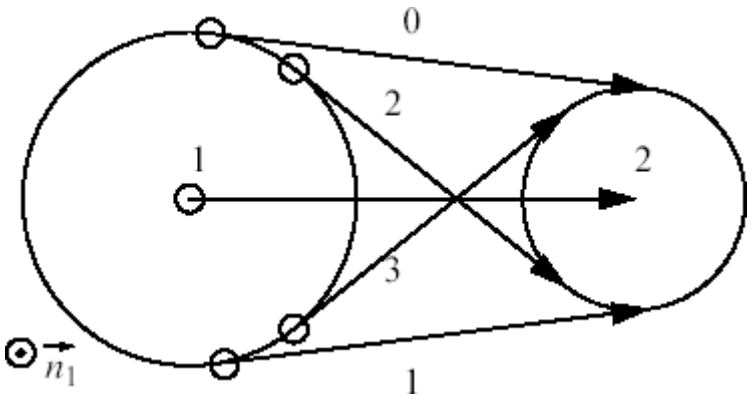
  Used to turn off the hold-relationship behavior (default nil, use hold-relationship).

  Default Formula: nil

- **tangent-lines**

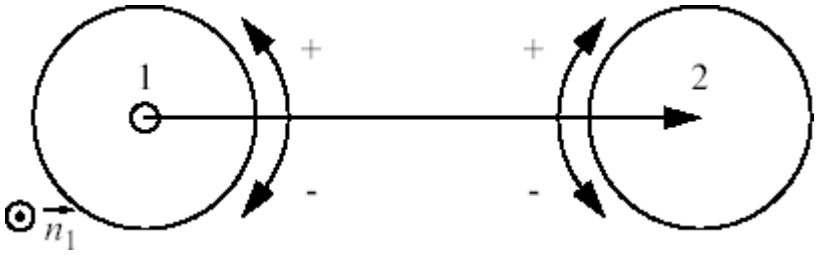
  This property gives the list of the tangent point on circle-1 and a direction vector for each of the lines tangent to the two circles or arcs. The vector addition of the tangenet point on circle-1 and the direction vector gives the coordinates of the points on circle-2. This property is an output property and should not be set by the user. Example: '(((0 0.5 0) (1 0 0)) ((0 -0.5 0) (1 0 0)) ...)

Figure 4-24. line-cl-cl-tg



The index of the selected tangent line is relative to the vector connecting the center of circle 1 to the center of circle 2 and the normal of circle 1.

Figure 4-25. lcct-select-two-sides



line-pt-cl-tg [class]

The line-pt-cl-tg object defines a line tangent to a circle or arc and starting at a given point. The tangent line will be one of two possible lines.

Inheritance: line-object, basic-tangent-object

Properties:

- **point1**

  A list containing x,y,z coordinates of point defining end of line.

- **tangent-object**

  The circle or arc the line should be tangent to (circle-object or arc-object).

- **selection-info**

  Information governing the choice of tangent lines.

- **selection-method**

  The name of the method used to select tangent line. The possible values are: 'lpct-select-side or 'tg-select-index or 'tg-select-interactive. The **lpct-select-side** method will select the line based on which side of the circle or arc it is tangent to. The rationale is similar to the method lcct-select-two-sides. Based on the vector from the point to the center of the object, a positive value of selection-info will be on the side of a positive rotation about the normal of the object from the vector. selection-info - a number, the sign of which indicates the location of the tangent line. The **tg-select-interactive** method will display the two possible tangent lines for the user to select interactively. Selection is made by the tangent point on the arc or circle. selection-info - doesn't matter. The **tg-select-index** method will choose which tangent line to use based on a simple index. The two possible tangents are numbered 0 and 1. selection-info - a single number indicating the desired tangent arc.

- **hold-relationship**

  Determines whether to change selection based on method and info.

- **ignore-hold-relationship**

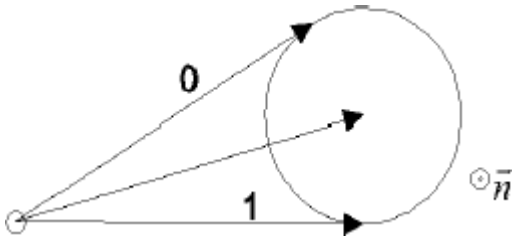
  used to turn off the hold-relationship behavior (default nil, use hold-relationship).

  Default Formula: nil

- **tangent-points**

  A list of the two contact points between the tangent lines and tangent-object.

Figure 4-26. line-pt-cl-tg



Example 4-33. Line-pt-cl-tg example

```
(define-class line-pt-cl-tg-example-class
  :inherit-from (line-pt-cl-tg)
  :properties (
           point1 '(-3 0 0)
           tangent-object ^circle
           )
  :subobjects (
           (circle :class 'circle-object
              )
           )
  )

(create-model 'name-generator)
;; returns a new name-generator model
(add-object (the) (generate-name (the) 'example) 'line-pt-cl-tg-example-
class)
```

Geometric Multi-Object Methods

Methods

pt-closest [Method]

Defined on Classes:

line-object,

point-object

This method returns the point on the from-object that is closest to the to-object. The point is returned as a list of x,y,z coordinates. The function has an optional argument, :limits, which affects the way lines are treated. If :limits is nil, then the line is treated as extending infinitely in both directions; the result of pt-closest will not necessarily lie within the endpoints of the line. If :limits is t, then the result of pt-closest will be constrained to be within the end points of the line.

Arguments:

- **to-object**

  A line-object, point-object, or list of coordinates specifying a point.

- **from-object**

  A line-object or a circle-object.

Keyword Arguments:

- **:limits**

  When supplied as t the resulting point will lie on a line segment. Default allows the point to lie anywhere along an infinite line.

  Default Value: nil

pt-intersection [Method]

Defined on Classes:

line-object,

circle-object,

arc-object

This method returns a point corresponding to the intersection of the two objects. The point is returned as a list of x,y,z coordinates. Object1 and object2 may be any objects which inherit from line-object, arc-object, or circle-object (a special case also exists if one of the objects inherits from point-object). Cases which involve arc-objects or circle-objects may have more than one intersection point. In these cases, only one point will be returned. If the other point is desired, the keyword option :num must be used (i.e.,(PT-INTERSECTION object1 object2 :num 1)) will return the second intersection between the two objects). The default value for :num is 0. If all intersections are desired, the keyword :all may be used. In this case, (PT-INTERSECTION object1 object2 :all t) will return a list of all the intersection points between the two objects. The default value for :all is nil. The :all keyword takes precedence over a specified :num. In all cases, if there is no applicable intersection the return value is nil. Pt-intersection operates differently if one of the objects inherits from line-object and the other object inherits from point-object (or is a list of x,y,z coordinates). In this case, pt-intersection returns the projection of the point onto the line-object. A line drawn from the point to the pt-intersection will be perpendicular to the original line-object. Other than the special cases for point-object, pt-intersection will fail if any object which does not inherit from line-object, arc-object, or circle-object is passed to it.

Arguments:

- **object1**

  A line-object, arc-object, circle-object, or point-object.

- **object2**

  A line, arc, circle, or point object.

Keyword Arguments:

- **:num**

  The number of the intersection to return.

  Default Value: 0

- **:all**

  When supplied as t all intersection points are returned.

  Default Value: nil

pt-midpoint [Method]

Defined on Classes:

line-object,

arc-object

This method returns a point that is the midpoint of the specified object. The object can be either a line-object or an arc-object.

Arguments:

- **object**

  A line-object or arc-object

pt-percentage [Method]

Defined on Classes:

line-object,

circle-object,

arc-object

This method returns the point which is the specified percentage of the object length from the beginning of the object. Allowable objects are line-objects, circle-objects, and arc-objects. If the percentage is within the range 0 to 1, the point will lie on the object. Circles and arcs will repeat along the curve. For lines, numbers outside this range will produce points off of the line. Points along circles and arcs start with 0 at the start vector of the object. The parameter :limits controls the behavior of this function with arcs. If :limits is t, then the endpoints of the arc will be at percentage values of 0 and 1. If :limits is nil, then the arc will be treated as a full circle, with 0 and 1 specifying the same point.

Arguments:

- **object**

  A line-object, circle-object, or arc-object.

- **percentage**

  A number specifying position along the length of object. The range 0 to 1 is within the object length. Numbers outside this range specify points beyond the object.

Keyword Arguments:

- **:limits**

  If nil, then arcs are treated as circles (the range 0 to 1 continues outside the arc length). If t, then the range 0 to 1 is within the arc length.

pt-quadrant [Method]

Defined on Classes:

line-object,

circle-object

This function returns a point that locates the specified quadrant of the object. Allowable quadrant numbers are 0 through 4. If object is a circle-object, then the quadrant indicates the 0, 90, 180, 270, and 360 degree points on the radius of the circle (number 0 and 4 return the same point). If object is a line-object, then the quadrant number specifies the quarter of the line segment (i.e., 0 is one end, 2 is the midpoint, and 4 is the other end).

Arguments:

- **object**

  A line-object or circle-object.

- **number**

  The quadrant number (0 to 4) corresponding to desired position.

Geometric Operations

Classes

assembly-object [class]

The assembly-object creates a single geometry by assembling the supplied objects into a single entity. Any common points will be duplicated. An assembly-object is more efficient than a sewn-object when connectivity is not important. An assembly-object has a single color, linewidth, and line-type.

Inheritance: simple-geometry-class

Properties:

- **object-list**

  A list of object instances that will be assembled.

Example 4-34. Assembly-object example

```
(define-class ASSEMBLY-LINES
  :inherit-from (assembly-object)
  :properties (
               object-list (select-object :from (the superior)
                                          :class 'line-object)
               )
  :subobjects (
               (l1 :class 'line-object
                   point1 '(0.2 0.0 0.0)
                   point2 '(0.8 0.0 0.0))
               (l2 :class 'line-object
                   point1 '(1.0 0.0 0.0)
                   point2 '(0.0 1.0 0.0))
               (l3 :class 'line-object
                   point1 '(0.0 0.8 0.0)
                   point2 '(0.0 0.2 0.0))
               )
  )
;; A triangular shape that is not connected at the corners will
;; be created.
```

See Also: group-object, sewn-object

basic-object [class]

The basic-object is an object that is an assembly of objects that are created by dividing an object instance into its component geometries. Each new object that results from dividing the sourceobject into its component geometries is created as a subobject of the basic-object instance. The subobjects created inherit from the class geom-subobject.

Inheritance: geom-series

Properties:

- **source-object**

  The object instance that will be divided into component geometries.

  Default Formula: nil

- **display?**

  The default will not display the object.

  Default Formula: nil

Example 4-35. Basic-object Example

```
(define-class BASIC-BOX
  :inherit-from (BASIC-OBJECT)
  :properties (
               source-object !box
               )
  :subobjects (
               (box :class 'box-object
                       solid? nil)
               )
 )

;;The component geometries for the box that is not solid are 6 faces.
;;(the basic-box-0000), (the basic-box-0001), (the basic-box-0002),
;;(the basic-box-0003), (the basic-box-0004), and (the basic-box-0005)
;;are all geom-subobjects that may be used independently.
```

bounded-object [class]

The bounded-object creates geometry that is the halfspace of the objects supplied. When given a list of coplanar 1 dimensional objects (curves, lines, ...) that form a closed loop, a bounded-object will create a surface bounded by those objects. When given a list of surfaces that surround a closed volume, a bounded-object will create a 3- dimensional solid bounded by those surfaces.

Inheritance: geom-object

Properties:

- **object-list**

  A list of object instances that will be used to create a halfspace.

- **dimension**

  The geometric dimensionality of the new geometry.

  Default Formula: 2

- **copy?**

  Default true will not destroy the original objects in the object list.

Default Formula: t

- **normal**

  If dimension is 2, then this specifies the normal to the plane of the bounded object. Default is nil which causes the normal to be computed internally. Specifying the normal will reduce computation time for the geometry. If the dimension is 3, this property is ignored.

  Default Formula: nil

Example 4-36. Bounded-object example

```
(define-class BOUNDED-SURFACE-EXAMPLE
  :inherit-from (bounded-object)
  :properties (
               width 1.0
               height 1.0
               fillet-radius 0.05
               xc (* 0.5 ^width)
               yc (* 0.5 ^height)
               object-list (list ^line-1 ^line-2 ^line-3 ^line-4
                                 ^arc-1 ^arc-2 ^arc-3 ^arc-4)
               (line-1 :class 'line-object
                       point1 (list (* -0.9 ^^xc) (-^^yc) 0)
                       point2 (list (* 0.9 ^^xc) (-^^yc) 0)
                       )
               (line-2 :class 'line-object
                       point1 (list ^^xc (* -0.9 ^^yc) 0)
                       point2 (list ^^xc (* 0.9 ^^yc) 0)
                       )
               (line-3 :class 'line-object
                       point1 (list (* 0.9 ^^xc) ^^yc 0)
                       point2 (list (* -0.9 ^^xc) ^^yc 0)
                       )
               (line-4 :class 'line-object
                       point1 (list (-^^xc) (* 0.9 ^^yc) 0)
                       point2 (list (-^^xc) (* -0.9 ^^yc) 0)
                       )
               (arc-1 :class 'arc-ln-ln-tg
                      radius ^^fillet-radius
                      line-1 ^^line-1
                      line-2 ^^line-2
                      hold-relationship nil
                      selection-info 3
                      )
               (arc-2 :class 'arc-ln-ln-tg
                      radius ^^fillet-radius
                      line-1 ^^line-2
                      line-2 ^^line-3
                      hold-relationship nil
                      selection-info 3
                      )
               (arc-3 :class 'arc-ln-ln-tg
                      radius ^^fillet-radius
```

```
                    line-1 ^^line-3
                    line-2 ^^line-4
                    hold-relationship nil
                    selection-info 3
                    )
                (arc-4 :class 'arc-ln-ln-tg
                    radius ^^fillet-radius
                    line-1 ^^line-4
                    line-2 ^^line-1
                    hold-relationship nil
                    selection-info 3
                    )
                )
  )

(define-class BOUNDED-SOLID-EXAMPLE
  :inherit-from (bounded-object)
  :properties (
                cyl-height 2.0
                cyl-diameter 1.0
                object-list (list ^cyl-base ^cyl-surface ^cyl-top)
                (cyl-base :class 'disc-object
                          diameter ^^cyl-diameter
                          orientation (list
                                       (translate
                                        (list 0 0 (* -0.5 ^^cyl-height)))
                                       )
                          )
                (cyl-surface :class 'open-cylinder-object
                             height ^^cyl-height
                             diameter ^^cyl-diameter
                             )
                (cyl-top :class 'disc-object
                         diameter ^^cyl-diameter
                         orientation (list
                                      (translate
                                       (list 0 0 (* 0.5 ^^cyl-height)))
                                      )
                         )
                )
  )
```

capped-surface-class [class]

A capped-surface-object takes an open surface (such as an open-cylinder, or a skin-surfacefrom- curves-object, ...) as a source-object, and creates either a capped surface (solid? = nil) or a solid bounded by the capped surface (solid? = t).

Inheritance: geom-object

Properties:

- **source-object**

  An instance of the open surface to cap.

- **solid?**

  Default creates a capped solid. When nil, the surface is only capped and no solid geometry is created.

  Default Formula: t

classify-object [class]

The classify-object is an object instance that is an assembly of objects that are created by performing a classify operation. Each new object that results from classifying is created as a subobject instance inheriting from the class geom-subobject. This class is not supported when running the AML Parasolid-based modeler; the functionality provided by this class can be achieved using other classes like difference-object, intersection-object...

Inheritance: geom-series

Properties:

- **object1**

  An object instance that will be classified with respect to object2.

  Default Formula: nil

- **object2**

  An object instance that will be classified with respect to object1.

  Default Formula: nil

- **code1**

  The type of classifications for object1.

  Default Formula: '(in on out)

- **code2**

  The type of classifications for object2.

  Default Formula: '(in on out)

- **display?**

  The default will not display the object

  Default Formula: nil

Example 4-37. Classify-object example

```
(define-class CLASSIFY-TEST
   :inherit-from (CLASSIFY-OBJECT)
   :properties (
                object1 !box
                object2 !can
                )
   :subobjects (
                (box :class 'box-object)
                (can :class 'cylinder-object
                height 2.0)
                )
```

```
  )

;; The classify-test will create 6 subobjects. The object suffixes
;; correspond to code1 and code2 values in that order.
;; (the classify-test-0000) is the object instance resulting from
;; object1 in object2.
;; (the classify-test-0001) is the object instance resulting from
;; object1 on object2.
;; (the classify-test-0002) is the object instance resulting from
;; object1 out of object2.
;; (the classify-test-0003) is the object instance resulting from
;; object2 in object1.
;; (the classify-test-0004) is the object instance resulting from
;; object2 on object1.
;; (the classify-test-0005) is the object instance resulting from
;; object2 out object1.
```

complement-halfspace-object [class]

A complement-halfspace-object defines all space (infinite) that is not defined by a given halfspace-object.

Inheritance: geom-object

Properties:

- **source-object**

  The space of this object should be complemented.

- **display?**

  Do not draw the complement-halfspace-object. Because the object is infinite graphic commands that require size will cause problems.

  Default Formula: nil

See Also: halfspace-object

divide-object [class]

The divide-object is an object that is an assembly of objects that are created by dividing an object instance. Each new object that results from dividing the source-object is created as a subobject instance inheriting from the class geom-subobject.

Inheritance: geom-series

Properties:

- **source-object**

  The object instance that will be divided.

  Default Formula: nil

- **dividing-object**

  The object instance that will divide the source-object.

Default Formula: nil

- **display?**

  The default nil, will not display the object.

Example 4-38. Divide-object example

```
(define-class SPLIT
  :inherit-from (divide-object)
  :properties (
               dividing-object (the sheet)
               source-object (the stock)
               )
  :subobjects (
               (stock :class 'cylinder-object
                      solid? t)
               (sheet :class 'sheet-object
                      height 5.0
                      width 5.0
                      )
               )
  )

;;The split object will create two cylinders as a result of dividing
;;the stock in two. The subobjects of the split object are created as
;;split-0000 and split-0001.
```

geom-copy-object [class]

The geom-copy-object creates geometry by copying the geometry of an object. The copied geometry is dependent on the original and any changes made to the original are reflected in the copy.

Inheritance: primitive-class

Properties:

- **source-object**

  An object instance from which to create a geometric copy.

Example 4-39. Geom-copy-object Example

```
> (create-model 'object)
;; returns an instance of object
> (add-object (the) 'box 'box-object)
;; creates a box
> (add-object (the) 'copy 'geom-copy-object :init-form '(sourceobject (the
box)))
```

geom-object [class]

A generic graphic object that is used to define generic functionality for geometric objects such as unions, intersections, differences, etc. This class should not be instantiated by the user.

Inheritance: graphic-object

Properties:

- **geom**

  This holds the geometric entity for the graphic instance.

  Default Formula: nil

geom-series [class]

This class is used as a base class to derive sub-geom-object, divide-object, classify-object, and basic-object and should not be instantiated by the user.

Inheritance: series-object, assembly-object

Properties:

- **class-expression**

  The class of the subobjects to be added.

  Default Formula: 'geom-subobject

- **object-list**

  Default Formula: (children (the superior))

- **derived-geoms**

  This is an internal property.

- **quantity**

  Default Formula: (length ^derived-geoms)

- **display?**

  Default Formula: nil

geom-subobject [class]

This class is used to create subobjects for the geom-series class. It should not be instantiated by the user.

Inheritance: geom-object

Properties:

- **display?**

  Default Formula: t

group-object [class]

A group-object creates a single geometry by assembling the supplied objects into a single entity. This object is similar to the assembly-object but the component objects retain their original color, line-type, and line-width. No boolean operation should be performed on a group object.

Inheritance: assembly-object

Properties:

- **object-list**

A list of graphic-object instances that will be grouped.

See Also: assembly-object

halfspace-object [class]

The halfspace-object defines space that extends infinitely, perpendicular to a rectangular sheet.

Inheritance: geom-object

Properties:

- **point**

  The center of the square sheet used to display the halfspace-object

- **vector**

  A direction vector that is perpendicular to the sheet passing through the point. The halfspace created is defined in the direction of the vector.

- **sheet-size**

  The size of the square sheet that will define the halfspace.

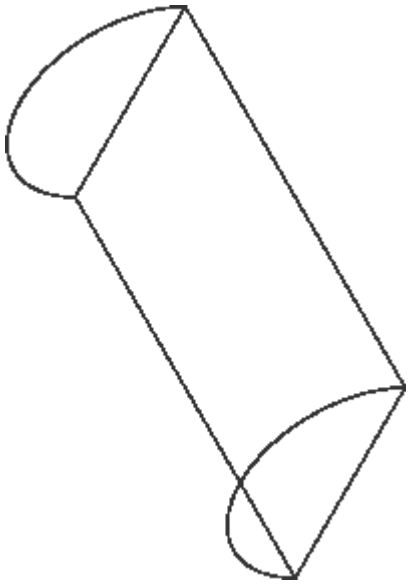
- **display?**

  When Default - do not draw the halfspace-object. Because the object is infinite, graphic commands that require size will cause problems.

  Default Formula: nil

- **drawn-geom**

Figure 4-27. Halfspace-object



The figure shows the intersection of the cylinder with the halfspace-object.

Example 4-40. Halfspace-object example

```
(define-class HALFCYLINDER-EXAMPLE
  :inherit-from (intersection-object)
  :properties (
               object-list (list !cylinder !halfspace)
               )
  :subobjects (
               (cylinder :class 'cylinder-object
                         solid? t
                         display? nil)
               (halfspace :class 'halfspace-object
                          point '(0.0 0.0 0.0)
                          vector '(0.0 1.0 0.0)
                          )
               )
  )
```

imprint-class [class]

This class creates geometry generated by imprinting a list of tool objects into the surface or boundary of a target object. For example, imprinting a sheet-object instance into another intersecting sheet-object instance results in an intersection curve embedded in the target sheet.

Inheritance: geom-object

Properties:

- **target-object**

  Object instance specifying the target.

  Default Formula: nil

- **tool-object-list**

  List of object instances defining the tool.

  Default Formula: nil

- **tolerance**

  Number specifying the distance within which a geometric component (edge/face) of the tool has to be from a geometric component (edge/ face) of the target in order for it to be imprinted on it. This tolerance plays an important role in imprinting a curve on a surface for example in order to determine the trace of the curve on the surface.

  Default Formula: 1.0e-4

Note : It is not recommended to use this class if running the AML SHAPES-based solid modeler.

mirror-object [class]

The mirror-object takes a geometric object and creates a mirror image geomtry out of it.

Inheritance: geom-object

Properties:

- **axis-basis?**

t for a mirror and nil for a reflection.

Default Formula: t

- **basis-vector**

  A list of two vectors that defines the mirror plane.

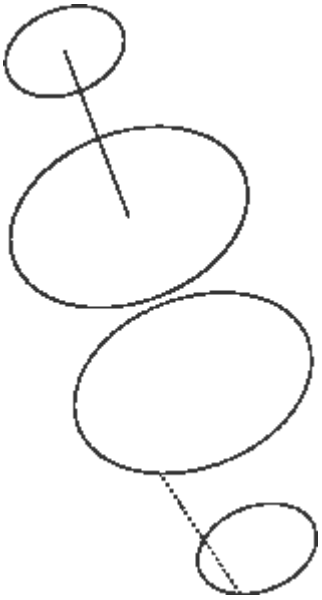
- **point-on-mirror**

  Specifies the plane location.

- **source-object**

  Instance to be mirrored.

Figure 4-28. Mirror-object



The mirrored object is shown with dashed lines.

Example 4-41. Mirror-object example

```
(define-class MIRROR-EXAMPLE
  :inherit-from (mirror-object)
  :properties (
               source-object ^cpipe
               basis-vector '((1 0 0) (0 1 0))
               point-on-mirror '(0 0 1)
               )
  :subobjects (
               cpipe :class 'cone-pipe-object
                     start-outer-diameter 0.5
                     end-outer-diameter 1.0
                     height 1.0
                     )
  )
```

offset-class [class]

The offset-class is used to offset a 1d, 2d, or a 3d geometric object by specifying an offset value. Restriction: 1d geometric objects have to be planar.

Inheritance: geom-object

Properties:

- **source-object**

  Object to offset.

  Default Formula: nil

- **offset**

  Offset distance.

  Default Formula: 0.0

- **method**

  For curve offset: :round -- using fillet to fill gaps (default). :linear -- linear extension of the curve to fill gaps.

  Default Formula: nil

- **normal**

  Normalized normal of the plane the curve lies on (for curve offset).

  Default Formula: nil

- **faces-list**

  This property is for surface offset only. It is a list of faces (sub geoms) to offset.

  Default Formula: nil

- **face-offset-values-list**

  List of face offset values (same length as faces-list) corresponding to the faces in faces-list for other faces, the offset value will be the value of offset property.

  Default Formula: nil

- **create-step-face?**

  This property is for surface offset only. If t, then side faces will be created along any smooth edges between the faces which have different offset value.

  Default Formula: nil

- **allow-disjoint?**

  This property is for surface offset only. If t, the result can have multiple disjoint geometries.

  Default Formula: nil

saved-geom-object [class]

This creates geomery based on a saved geom file.

Inheritance: geom-object

Properties:

- **file-name**

  A string specifying the name [and path] of the file containing the saved geom.

  Default Formula: nil

- **scale-factor**

  This is the factor that the saved object is scaled by after retrieval.

  Default Formula: 1.0

- **text-mode?**

  When t, then the geometry file is in ascii format, when nil then it's in binary format.

  Default Formula: t

- **retrieved-geoms**

  This property shows the disassembled parts when the geom is an assembly type.

Note:

1. If AML is running the PARASOLID-based solid modeler, the file-name property should not specify the name of the file exactly as it is on disk. Geom file names have an "X_T" (WINDOWS FAT) or "xmt_txt" (UNIX and WINDOWS NTFS) extension that should not be specified in the file-name string.

Example: a- WINDOWS: If the property file-name is "c:\temp\box.geom", the saved-geom-object instance expects the exsitence of the geom file "c:\temp\box.geom.X_T" on FAT file systems, or "c:\temp\box.geom.xmt_txt" on NTFS file systems. b- UNIX: Similarly to NTFS, if the property file-name is "/tmp/box.geom", the saved-geom-object instance expects the exsitence of the geom file "/tmp/box.geom.xmt_txt". A geom file can always be renamed from an "X_T" extension to an "xmt_txt" extension (and vice versa) when switching from a file system to the other.

2. Geoms that are saved while running a certain solid modeler cannot be retrieved if running another solid modeler. Geom files are solid-modeler dependent.

scale-object [class]

The scale-object will create a copy of the given object geometry and scale that geometry to the desired size.

Inheritance: geom-object

Properties:

- **source-object**

  The object instance to be copied and scaled.

- **scale-factor**

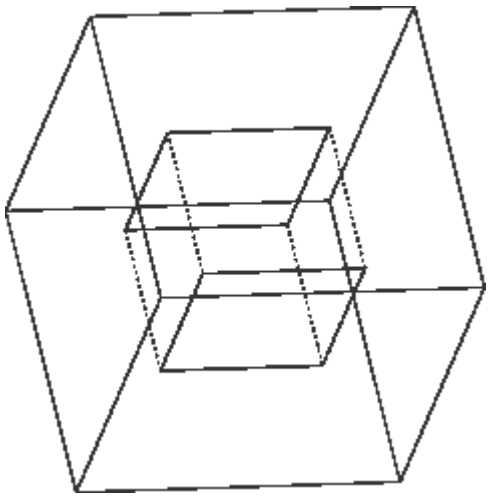
  The amount to scale the source-object. This property can be a real value (for uniform scaling) or a list of x,y, and z scale values for nonuniform scaling.

- **ref-point**

List of x,y and z coordinates specifying the scaling reference point in global space.

Default Formula: '(center-of-object (the superior))

Figure 4-29. Scale-object



The original box is shown with dashed lines.

Example 4-42. Scale-object example

```
(define-class SCALE-EXAMPLE
  :inherit-from (scale-object)
  :properties (
              source-object (the superior box)
              scale-factor 2.0
              )
  :subobjects ((box :class 'box-object))
  )
```

sewn-object [class]

The sewn-object will create a single geometry by sewing adjacent curves or surfaces together. When sewing non-adjacent objects, the results is the same as using the class assembly-object. The common points/edges/surfaces will appear only once in the resulting geometry. Adjacent objects must share a common boundary in order for them to be sewn: When dealing with curves, adjacent curves must have exactly the same coordinates for their common vertex. When dealing with surfaces, adjacent edges must be within the tolerance provided. Sewn-object doesn't work if any edge is shared by more than two surfaces. For example, sewn-object it works fine if you sew two open cylinders together but it will fail if you add a disc at the place where the two cylinders are being sewn.

Inheritance: geom-object

Properties:

- **object-list**

  A list of object instances that will be sewn together. This property does not need to be sorted.

Default Formula: nil

- **max-dimension**

  For sewing faces, if max-dimanesion is 3 and the faces form a closed region, a solid body will be created. If max-dimension > 3, a non-manifold body is allowed. For sewing edges, point-sewn-tolerance is used for detecting gap between adjacent edges.

- **tolerance**

  Distance within which the adjacent edges of two surfaces should be for the surfaces to be sewn. This property is ignored when dealing with curves.

  Default Formula: 0.0001

- **point-sewn-tolerance**

  The tolerance used for seewing edges.

See Also: assembly-object

Example 4-43. Sewn-object Example

```
(define-class SEWN-LINES
  :inherit-from (sewn-object)
  :properties (
               object-list (select-object :from (the superior)
                                          :class 'line-object)
               )
  :subobjects (
               (l1 :class 'line-object
                   point1 '(0.0 0.0 0.0)
                   point2 '(1.0 0.0 0.0)
                   )
               (l2 :class 'line-object
                   point1 '(1.0 0.0 0.0)
                   point2 '(0.0 1.0 0.0)
                   )
               (l3 :class 'line-object
                   point1 '(0.0 1.0 0.0)
                   point2 '(0.0 0.0 0.0)
                   )
               )
  )

;; A triangle will be created.
```

solid-from-faces-class [class]

It converts the closed region formed from a list of faces to a solid.

Inheritance: union-object

Properties:

- **object-list**

  It is the list of faces which should form a closed region.

Default Value: nil

stl-object [class]

This class creates a surface from an imported STL file. The STL file must be in text format (not binary).

Inheritance: primitive-class

Properties:

- **stl-file**

  The complete file path and name where the STL file exists.

  Default Formula: nil

sub-geom-object [class]

The sub-geom-object creates subobjects that are component geometries of the source geometry. Each new object that results from breaking the geometry down into its sub geometry is created as an instance of geom-subobject.

Inheritance: geom-series

Properties:

- **source-object**

  The object instance that will be divided into its sub-geometries.

  Default Formula: nil

- **sub-geom-dimension**

  The geom dimension for the subobjects. 0 for vertices, 1 for edges, 2 for faces, or 3 for solids.

  Default Formula: 0

Example 4-44. Sub-geom-object example

```
(define-class BOX-FACES
  :inherit-from (sub-geom-object)
  :properties (
               source-object !box
               sub-geom-dimension 2
               )
  :subobjects (
               (box :class 'box-object
                    display? nil ; do not draw
                    )
               )
  )

;;The box-faces will have 6 subobjects: (the box-faces-0000), (the box-
;;faces-0001), (the box-faces-0002), (the box-faces-0003), (the box-
;;faces-0004), and (the box-faces-0005) representing the 6 faces of the
;;box.
```

surface-divide-class [class]

This class has two subobjects, front-part and back-part which defines the geometry in front/back of the dividing surface.

Inheritance: geom-object

Properties:

- **source-object**

  The object that needs to be split.

- **dividing-surface-object**

  The surface object that is used for splitting.

- **front-point**

  It is a point location (x y z) that defines the front part portion of the split surface.

surface-thickened-class [class]

This class creates a solid by thickening a surface in both directions controlled by back-thickness and front-thickness.

Inheritance: geom-object

Properties:

- **source-object**

  The surface to be thickened

  Default Formula: nil

- **front-thickness**

  The front thickness

  Default Formula: 0.0

- **back-thickness**

  The back thickness

  Default Formula: 0.0

trim-object [class]

The trim-object will create geometry that is the resultant of a trim operation. The trimming operation is defined by selecting an object which will act as the trimming object and a point on the object to be trimmed that will define which part of the object to be retained.

Inheritance: geom-object

Properties:

- **source-object**

  The object instance which will be trimmed.

- **trimming-object**

The object instance which will be used to trim the source-object.

- **point-on-object**

  A point on or in the trim-object that will define the portion of the trimmed object to retain. If the trimming-object is a sheet, then the point object should only need to be on either side of the sheet in order to specify the side of the source-object that should be retained. If the trimming-object is any other type of object (for example a solid), then the point has to lie on the boundary of the source object.

- **to-trim**
- **trim-with**

Example 4-45. Trim-object example 1

```
(define-class TRIM-EXAMPLE
  :inherit-from (trim-object)
  :properties (
               source-object (the stock)
               trimming-object (the sheet)
               point-on-object '(0.0 0.0 0.0))
  :subobjects (
               (stock :class 'cylinder-object
                      solid? t)
               (sheet :class 'sheet-object
                      height 5.0
                      width 5.0
                      orientation (list
                                   (translate
                                    (vector 0.0 0.0
                                            (half (the stock height))))
                                   (rotate 45.0 :x-axis
                                          :axis-point (vector 0.0 0.0
                                                              (half (the
stock height))))))
               )
  )
```

Example 4-46. Trim-object example 2

```
(define-class test-trim-object-class
  :inherit-from(object)
  :subobjects (
               (point-object-1 :class 'point-object
                               coordinates (the superior superior trim-
object-1 point-on-object)
                               line-width 5
                               )
               (cylinder-object-1 :class 'cylinder-object
                                  orientation (list (translate '(3 0 0)))
                                  )
               (sheet-object-1 :class 'sheet-object
                               width 1.5
                               height 1.5
```

```
                                orientation (list (translate '(3 0 0)))
                                )
               (trim-object-1 :class 'trim-object
                              source-object ^^cylinder-object-1
                              trimming-object ^^sheet-object-1
                              point-on-object (convert-coords ^^sheet-object-
1 (list 0 0 (* 5 (the superior superior cylinder-object-2 height))))
                              )
               (point-object-2 :class 'point-object
                               coordinates (the superior superior trim-
object-2 point-on-object)
                               line-width 5
                               )
               (cylinder-object-2 :class 'cylinder-object
                                  orientation (list (translate '(-3 0 0)))
                                  )
               (box-object-2 :class 'box-object
                             width 1.5
                             height 1.5
                             depth 0.75
                             orientation (list (translate '(-3 0 0)))
                             )
               (trim-object-2 :class 'trim-object
                              source-object ^^cylinder-object-2
                              trimming-object ^^box-object-2
                              point-on-object (convert-coords ^^box-object-2
(list 0 0 (/ (the superior superior cylinder-object-2 height) 2)))
                              )
               )
  )
```

trimmed-surface-object [class]

A trimmed-surface-object takes a surface and a list of curves that lie EXACTLY in that surface and creates a surface trimmed to the boundaries of the curves. The curves specified can be sewn or disjoint but need to form a closed loop.

Inheritance: geom-object

Properties:

- **surface-object**

  Instance of a single surface object.

- **object-list**

  List of curve objects forming a closed loop.

- **outside?**

  When nil, it creates a surface defined by the inner area of the curves loop. When non nil, creates the outer surface.

  Default Formula: nil

Curves and Surfaces

Classes

curve-segment-from-curve-class [class]

This class extracts a curve segment from a larger curve object. This class works on single and multiple edge curve objects.

Inheritance: simple-geometry-class

Properties:

- **curve-object**

  The object instance of a curve geometry. It can be any single or multiple edge curve object.

- **start-point-coords**

  Coordinates of the start point for the curve segment.

- **end-point-coords**

  Coordinates of the end point for the curve segment.

curve-object [class]

A parametric curve object. This class cannot be instantiated by the user, it is inherited into the instantiable curve classes.

Inheritance: object

Properties:

- **dimension**

  Specifies that this is a 1d entity (a curve). This formula should not be changed.

  Default Formula: 1

surface-object [class]

A parametric surface object. This class cannot be instantiated by the user, it is inherited into the instantiable surface classes.

Inheritance: object

Properties:

- **dimension**

  Specifies that this is a 2d entity (a surface). This formula should not be changed.

  Default Formula: 2

nurb-object [class]

Used to create NURBs (Non-Uniform Rational Bsplines). This is inherited into nurb-curveobject and nurb-surface-object. It should not be instantiated by itself.

Inheritance: graphic-object

Properties:

- **points**

  A list of points defining the object

- **knots**

  A set of values used to control the shape of the NURB.

- **rational?**

  Set to t or nil to specify whether the nurb is rational. For t has to specify points as '(x y z w)

  Default Formula: nil

- **homogeneous?**

  If set to t, rational? has to be t and each point needs a weight and hence is specified as '(x y z w). When t, it means the point coordinates are given in homogeneous system, the actual 3D point is (x*w, y*w, z*w).

  Default Formula: nil

nurb-curve-object [class]

Used to create NURB curves. The user can specify the degree of the curve independent of the number of points used to specify the curve, hence getting more control on the shape of the curve. The curve is only guaranteed to pass through the start and the end points.

Inheritance: nurb-object, curve-object

Properties:

- **degree**

  The degree of the polynomial defining the curve.

  Default Formula: (- (length ^points) 1).

- **points**

  A list of points defining the curve.

  Default Formula: nil

- **rational?**

  Set to t or nil to specify whether the nurb is rational.

  Default Formula: nil

- **homogeneous?**

  If set to t, each point needs a weight and hence is specified as '(x y z w).

  Default Formula: nil

Example 4-47. Nurb-curve-object Example

```
(define-class nurb-curve-test
  :inherit-from (nurb-curve-object)
```

```
  :properties (
        points '((0 0 0) (1 -1 0) (2 1 0) (3 2 0) (4 1 0) (5 0 0) (6
           -2 0) (7 0 0))
        degree 2
        )
 )
```

nurb-surface-object [class]

This is used to create a NURB surface when given a grid of points. The user can specify the degrees of the surface independent of the number of points used to specify the surface, hence getting more control on the shape.

Inheritance: nurb-object, surface-object

Properties:

- **degrees**

  The degrees of the polynomials along the 2 directions of the grid defining the shape of the surface along that direction.

  Default Formula: (list (- (length (first ^points)) 1) (- (length ^points) 1))

- **points**

  A list of list of points defining the control grid.

  Default Formula: nil

Example 4-48. Nurb-surface-object Example

```
(define-class nurb-surface-test
  :inherit-from (nurb-surface-object)
  :properties (
               points '(((0 0 0) (1 -1 0) (2 1 0) (3 2 0) (4 1 0) (5 0 0) (6
-2 0) (7 0 0))
                        ((0 0 1) (1 -1 1) (2 1 1) (3 2 1) (4 1 1) (5 0 1) (6
-2 1) (7 0 1))
                        ((0 0 2) (1 -1 2) (2 1 2) (3 2 2) (4 1 2) (5 0 2) (6
-2 2) (7 0 2))
                        ((0 0 3) (1 -1 3) (2 1 3) (3 2 3) (4 1 3) (5 0 3) (6
-2 3) (7 0 3))
                       )
               degrees '(2 3)
               )
  )
```

bezier-object [class]

Used to create Bezier curves and surfaces. This is inherited into bezier-curve-object and beziersurface-object. It should not be instantiated by itself.

Inheritance: graphic-object

Properties:

- **points**

  A list of points defining the object.

  Default Formula: nil

- **rational?**

  Set to t or nil to specify whether the nurb is rational.

  Default Formula: nil

bezier-curve-object [class]

Used to create Bezier curves. The degree of the curve is one less than the number of points. The curve is guaranteed to pass through the first and last point only.

Inheritance: bezier-object, curve-object

Properties:

- **points**

  A list of points defining the curve.

  Default Formula: nil

Example 4-49. Bezier-curve-object Example

```
(define-class bezier-curve-test
  :inherit-from (bezier-curve-object)
  :properties (
        points '((0 0 0) (1 -1 0) (2 1 0) (3 2 0) (4 1 0) (5 0 0) (6 -2 0) (7
0 0))
        )
  )
```

bezier-surface-object [class]

This is used to create Bezier surface when given a grid of points. Degree of the surface in any direction is one less than the number of points in that direction.

Inheritance: nurb-object, surface-object

Properties:

- **points**

  A list of list of points defining the control grid.

  Default Formula: nil

Example 4-50. Bezier-surface-object Example

```
(define-class bezier-surface-test
  :inherit-from (bezier-surface-object)
  :properties (
```

```
        points '(((0 0 0) (1 -1 0) (2 1 0) (3 2 0) (4 1 0) (5 0 0) (6 -2 0)
(7 0 0))
                ((0 0 1) (1 -1 1) (2 1 1) (3 2 1) (4 1 1) (5 0 1) (6 -2 1) (7
0 1))
                ((0 0 2) (1 -1 2) (2 1 2) (3 2 2) (4 1 2) (5 0 2) (6 -2 2) (7
0 2))
                ((0 0 3) (1 -1 3) (2 1 3) (3 2 3) (4 1 3) (5 0 3) (6 -2 3) (7
0 3))
                 )
        )
  )
```

conic-curve-class [class]

The curve is defined by three control points. The first and last are the endpoints of the conic arc,while the placement of the inner control point helps determine the shape of the curve.

Inheritance: nurb-curve-object

Properties:

- **control-points-coordinates-list**

  A list of 3 point coordinates in the following order: start-point, control-point and end-point. This parameter can also be a list that can describe more than one conic-curve-class segment. The format is still a flat list of points. It takes three points to define a conic curve. If you provide more than three points, you can describe more than one conic-curve segment. Since the segments are assumed to be connected, you need to have 3 points for the first segment, but only need to add 2 points for each additional segment. The conic-curve-class assumes the segments are connected so it figures out how the points are related.

```
1 Segment:  3 Points
2 Segments: 5 Points
3 Segments: 7 Points
4 Segments: 9 Points
...
```

  Default Formula: nil

- **reference-point-location-ratio**

  If M is the reference point on a line connecting the start and end point, and if N is the point on the curve where the line connecting the control point and the reference point, it is the ratio of AM and AC (AM/AC). It can take one value (and it is assumed to be applied to each of the conic-curve-class segments, or it can be a list where it will be applied in a one to one relationship to the conic-curve-class segments.

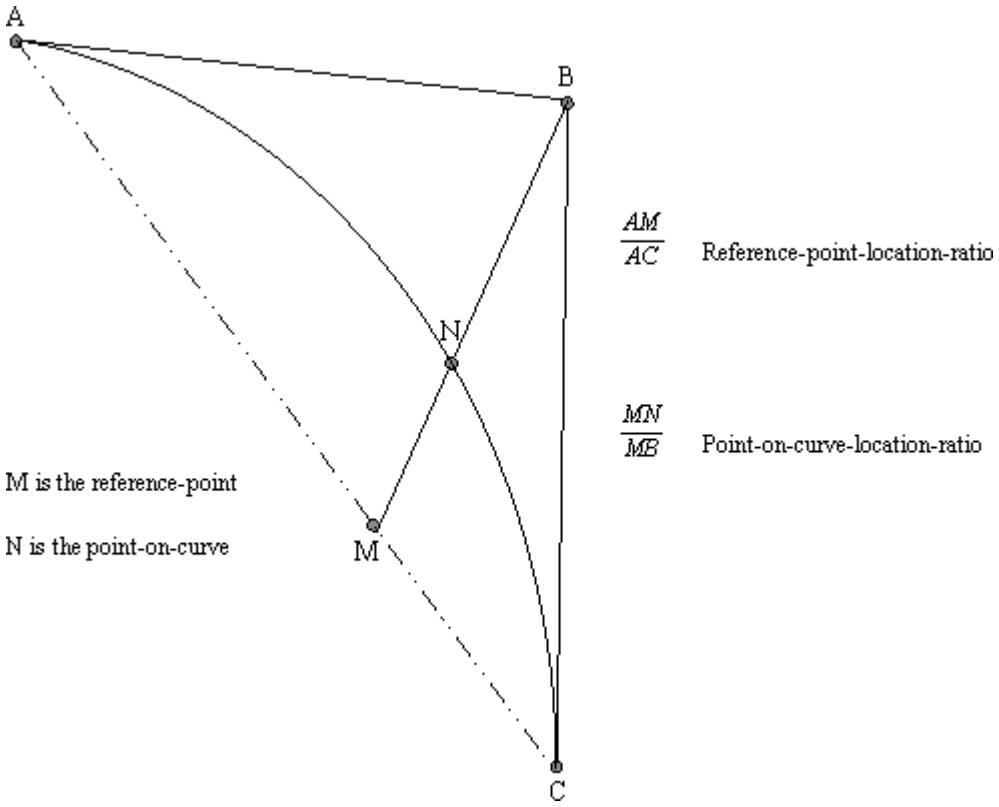
  Default Formula: nil

- **point-on-curve-location-ratio**

  Ratio of MN and MB (MN / MB). It can take one value (and it is assumed to be applied to each of the conic-curve-class segments, or it can be a list where it will be applied in a one to one relationship to the conic-curve-class segments.

Default Formula: 0.5

Figure 4-30. Conic-curve-class



curve-from-intersection-of-extruded-curves-class [class]

This class takes two curve objects and extrudes them in specified directions. The final geometry is the intersection of those two extruded surfaces.

Inheritance: geom-object

Properties:

- **curve-1-object**

  Object to be extruded in extrusion-1-vector direction.

- **curve-2-object**

  Object to be extruded in extrusion-2-vector direction.

- **extrusion-1-vector**

  The first direction for extruding. e.g. '(1 0 0).

- **extrusion-2-vector**

  The second direction for extruding. e.g. '(0 1 0).

Currently, the extrusion vectors must be selected from '(1 0 0), '(0 1 0), and '(0 0 1). Other values are not guaranteed to work and should be avoided.

curve-from-points-class [class]

This class creates a NURB from a set of given points coordinates. The NURB will go through all the points.

Inheritance: curve-object, geom-object

Properties:

- **points-coordinates-list**

  List of point XYZ coordinates that will form the curve.

- **closed?**

  When non-nil, creates a closed nurb by connecting the last point in the points-coordinates-list to the first one.

  Default Formula: nil

- **periodic?**

  This property is only effective when closed? is non-nil. When dealing with a closed curve and when this property is given a non-nil value, the geometry of the curve will be tangent continuous at the closing point (First point in the coordinates list).

- **derivs-list**

  This controls the tangencies of the start and end point of each curve segment.

curve-from-points-hermite-class [class]

This class creates a "hermite" curve given point coordinates and derivatives at those points.

Inheritance: curve-object, geom-object

Properties:

- **points-coordinates-list**

  List of points (XYZ coordinate list per point) that the curve goes through.

- **derivs-list**

  List of derivative pairs '(start end) at each segment, defined by adjacent two points.

curve-from-points-spline-class [class]

This class creates a NURB curve passing through the points defined in points-coordinates-list.

Inheritance: curve-object, geom-object

Properties:

- **points-coordinates-list**

  List of point XYZ coordinates that will form the curve.

  Default Formula: nil

- **closed?**

  When non-nil, creates a closed nurb by connecting the last point in the points-coordinates-list to the first one.

Default Formula: (default nil)

- **periodic?**

  This property is only effective when closed? is non-nil. When dealing with a closed curve and when this property is given a non-nil value, the geometry of the curve will be tangent continuous at the closing point (First point in the coordinates list).

  Default Formula: (default nil)

- **derivs-list**

  This controls the tangencies of the start and end point of each curve segment. When non-nil, each point must have a value as (x y z) or nil.

  Default Formula: nil

- **parameters-list**

  This property defines the parametre for each point. When non-nil, it must be given for each point.

  Default Formula: nil

- **param-range**

  This property defines the parameter range of the curve. By default, the range is '(0 1).

  Default Formula: nil

- **knots**

  This property defines the knots distribution of the curve. By default, it is generated by the system automatically.

  Default Formula: nil

- **degree**

  This property defines the degree of the curve. By default it is a cubic curve.

  Default Formula: 3

curve-in-surface-object [class]

This class creates a curve that lies on a surface object specified.

Inheritance: geom-object

Properties:

- **points-list**

  List of points defining the curve. The format of this list must depend on the value of the param-points property.

- **tolerance**

  Number specifying the distance to the source-object surface in which the points-list coordinates must be within.

  Default Formula: 1000.0

- **source-object**

  Object instance of the surface to create the curve in. Must be of dimension 2 (i.e. a surface geom).

Default Formula: nil

- **param-points?**

  When non-nil, specifies that points-list is a list of UV values in the parameter space of the source-object. Example: (list '(0.1 0.1) '(0.1 0.2) ...). When nil, specifies that points-list is a list of XYZ coordinate lists in model space. Example: (list '(0 0 0) '(0.1 0.1 0) '(0.25 0.1 0) ...).

  Default Formula: t

- **dimension**

  If the curve defined by points-list is closed, a value of 2 specifies that a surface will be generated with a new boundary defined by pointslist. A value of 1 will create a curve.

  Default Formula: 1

- **normal**

  XYZ list that specifies the direction in which the points in points-list will be projected on the source-object surface.

  Default Formula: nil

- **smooth?**

  When non-nil, the curve will be smooth. When nil, the points defined in points-list will be connected with straight lines.

  Default Formula: nil

curve-conic-from-line-arc-class [class]

This class takes a list of line and arc segments and creates a conic nurb curve starting from the given start-point.

Inheritance: conic-curve-class

Properties:

- **object-list**

  A list of line and arc objects.

  Default Formula: nil

- **start-curve**

  This is the start segment.

  Default Formula: nil

- **start-point**

  Starting point of the start curve.

  Default Formula: nil

project-curve-to-surface-object [class]

This class creates a curve geometry by projecting a curve onto a surface.

Inheritance: curve-in-surface-object

Properties:

- **curve-object**

  Object instance of a curve geometry. It must be a single edge curve object.

- **source-object**

  Object instance of a surface geometry.

- **n-points**

  Number of points that will form the resulting curve geometry,

interpolated-object [class]

This class is used to create interpolated curves and surfaces. This class provides generic functionality for instantiable interpolated geometry classes. It should not be instantiated by the user.

Inheritance: graphic-object

interpolated-curve-object [class]

Used to create interpolated curves that pass through the specified points and is created by performing a functional fit through the points. The user can optionally specify parameter values and tangent vectors for each of the points.

Inheritance: interpolated-object, curve-object

Properties:

- **points**

  A list of points defining the curve.

  Default Formula: nil

- **params**

  A list of parameter values corresponding to the points.

  Default Formula: nil

- **derivs**

  A list of vectors (x y z) corresponding to each point.

  Default Formula: nil

Example 4-51. Interpolated-curve-object

```
(define-class interpolated-curve-test
  :inherit-from (interpolated-curve-object)
  :properties (
        points '((0 0 0) (1 -1 0) (2 1 0) (3 2 0) (4 1 0) (5 0 0) (6 -2 0) (7
0 0))
        derivs '((10 0 0) (10 10 0) (10 5 0) (10 0 0) (-10 -10 0) (-10 -5 0)
(10 10 0) (10 0 0))
                )
   )
```

interpolated-surface-object [class]

Used to create interpolated surfaces that pass through the specified points grid, and created by performing a functional fit through the points. The user can optionally specify parameter values and tangent vectors for each of the points.

Inheritance: interpolated-object, surface-object

Properties:

- **points**

  A list of points defining the curve.

  Default Formula: nil

- **params**

  A list of parameter values corresponding to the points.

  Default Formula: nil

- **derivs**

  A list of vectors (x y z) corresponding to each point.

  Default Formula: nil

Example 4-52. Interpolated-surface-object Example

```
(define-class interpolated-surface-test
  :inherit-from (interpolated-surface-object)
  :properties (
        points '(((0 0 0) (1 -1 0) (2 1 0) (3 2 0) (4 1 0) (5 0 0) (6 -2 0)
(7 0 0))
                ((0 0 1) (1 -1 1) (2 1 1) (3 2 1) (4 1 1) (5 0 1) (6 -2 1) (7
0 1))
                ((0 0 2) (1 -1 2) (2 1 2) (3 2 2) (4 1 2) (5 0 2) (6 -2 2) (7
0 2))
                ((0 0 3) (1 -1 3) (2 1 3) (3 2 3) (4 1 3) (5 0 3) (6 -2 3) (7
0 3))
                )
        )
  )
```

skin-object [class]

This class provides generic skin curves and surface functionality. It is inherited into instantiable skin classes and must not be instantiated by the user.

Inheritance: graphic-object

skin-curve-from-curves-object [class]

This object takes a list adjacent curves and creates a single skin curve from them.

Inheritance: geom-object

Properties:

- **curve-objects**

  List of ordered contiguous curve instances to skin.

  Default Formula: nil

- **n-points-per-curve**

  Number of points to generate on each of the curve objects.

  Default Formula: 20

skin-surface-from-points-object [class]

Used to create skinned surfaces that pass through the specified points grid, and created by performing a functional fit through the points.

Inheritance: skin-object, surface-object

Properties:

- **points**

  A list of list of points defining the surface.

  Default Formula: nil

Example 4-53. Skin-surface-from-points-object Example

```
(define-class skin-surface-from-points-test
  :inherit-from (skin-surface-from-points-object)
  :properties (
        points '(((0 0 0) (1 -1 0) (2 1 0) (3 2 0) (4 1 0) (5 0 0) (6 -2 0)
(7 0 0))
                ((0 0 1) (1 -1 1) (2 1 1) (3 2 1) (4 1 1) (5 0 1) (6 -2 1) (7
0 1))
                ((0 0 2) (1 -1 2) (2 1 2) (3 2 2) (4 1 2) (5 0 2) (6 -2 2) (7
0 2))
                ((0 0 3) (1 -1 3) (2 1 3) (3 2 3) (4 1 3) (5 0 3) (6 -2 3) (7
0 3))
                 )
          )
  )
```

skin-surface-gen-curves-grid-object [class]

Used to create skinned surfaces from a list of curves in 1 direction. A list of cross curves is generated for greater accuracy. You can specify the number of cross-curves as a parameter.

Inheritance: skin-object, surface-object

Properties:

- **curve-objects**

  A list of curves defining the surface.

Default Formula: nil

- **num-points**

  Number of cross curves to generate.

  Default Formula: 5

- **pnt-tol**

  The curves will be within this tolerance of the surface.

  Default Formula: 0.001

- **ang-tol**

  This is the sine of the maximum angle between the tangent of the curves and the resulting surface geom. If ang-tol >= 1.0, no angular-tolerance is applied.

  Default Formula: 0.001

Example 4-54. Skin-surface-gen-curves-grid-object Example

```
(define-class skin-surface-gen-curves-grid-test
  :inherit-from (skin-surface-gen-curves-grid-object)
  :properties (
               curve-objects (l ist ^curve-1 ^curve-2 ^curve-3 ^curve-4)
               )
  :subobje      cts (
                     (curve-1 :class 'skin-curve-object
                              points '((0 0 0) (1 -1 0) (2 1 0) (3 2 0) (4 1
0) (5 0 0) (6

-2 0) (7 0 0))
                              )
                     (curve-2 :class 'skin-curve-object
                              points '((0 0 1) (1 -1 1) (2 1 1) (3 2 1) (4 1
1) (5 0 1) (6

-2 1) (7 0 1))
                              )
                     (curve-3 :class 'skin-curve-object
                              points '((0 0 2) (1 -1 2) (2 1 2) (3 2 2) (4 1
2) (5 0 2) (6

-2 2) (7 0 2))
                              )
                     (curve-4 :class 'skin-curve-object
                              points '((0 0 3) (1 -1 3) (2 1 3) (3 2 3) (4 1
3) (5 0 3) (6

-2 3) (7 0 3))
                              )
                     )
  )
```

skin-surface-from-polylines-patches-object [class]

This object takes a list of list of points with each list of points being treated as a cross-section. These points are skinned into surfaces using four points at a time and finally all the surfaces are sewn together to form a single object. The skinned surfaces are created in the following order: surface 0: points 0 and 1 from list 0 and points 0 and 1 from list 1. surface 1: points 1 and 2 from list 0 and points 1 and 2 from list 1. And so on. Note that each point list must have the same number of points.

Inheritance: geom-object

Properties:

- **point-lists**

  List of list of points defining the cross sections.

  Default Formula: nil

- **closed?**

  If each cross-section is give as n points, and closed? is t, the object checks if the first point of each cross-section is equal to the last point of the cross-section, and if it is not the same, it repeats the first point at the end of the list to close it when building the surfaces. If closed? is nil, no points are added to the list while building the surface.

Example 4-55. Skin-surface-from-polylines-patches-object Example

```
(define-class surf-from-patches
  :inherit-from (skin-surface-from-polylines-patches-object)
  :properties (
               point-lists '(
                                  ((0 0 0) (1 0 0) (2 1 0) (2 2 0) (1 3 0) (0 3
0) (-1 2 0) (-1 1 0))
                                  ((0 0 1) (1 0 1) (2 1 1) (2 2 1) (1 3 1) (0 3
1) (-1 2 1) (-1 1 1))
                                  ((0 0 2) (1 0 2) (2 1 2) (2 2 2) (1 3 2) (0 3
2) (-1 2 2) (-1 1 2))
                                  ((0 0 3) (1 0 3) (2 1 3) (2 2 3) (1 3 3) (0 3
3) (-1 2 3) (-1 1 3))
                                  )
               closed? t
          )
   )
```

surface-from-three-edge-curves-class [class]

This class generates a surface patch from three connected curve objects. The three curves must form a closed loop. The geometry of each curve object must be a single curve.

Inheritance: graphic-object

Properties:

- **edge-1-object**

  Curve object instance.

- **edge-2-object**

  Curve object instance.

- **edge-3-object**

  Curve object instance.

Example 4-56. Surface-from-three-edge-curves-class Example

```
(define-class triangular-surface-test-class
  :inherit-from (surface-from-three-edge-curves-class)
  :properties (
               edge-1-object ^line-1
               edge-2-object ^line-2
               edge-3-object ^line-3
               )
  :subobjects (
               (line-1 :class 'line-object
                       point1 '(0 0 0)
                       point2 '(1 0 0)
                       )
               (line-2 :class 'line-object
                       point1 '(1 0 0)
                       point2 '(0.5 1 0)
                       )
               (line-3 :class 'line-object
                       point1 '(0.5 1 0)
                       point2 '(0 0 0)
                       )
               )
  )
```

surface-from-uv-curves-class [class]

Creates a surface through a set of curves.

Inheritance: graphic-object skin-object

Properties:

- **u-curves-objects-list**

  List of cross-section curve objects along the u direction. Every curve in the list must have the same number of vertices (Same number of sub-geoms of dimension 0, including the intersection points with the vcurves). Each u curve has to be topologically oriented in the same direction unless the ucurves-orient-points-set-list property is specified.

  Default Formula: nil

- **u-curves-tangency-specifications-lists**

  The general format of the u-curves-tangency-specifications-lists is as follows:

  ```
  (list
  ```

```
  (list curve-object tangency-type (list ...))
  (list curve-object tangency-type (list ...))
  (list curve-object tangency-type (list ...))
  ...)
```

where: curve-object = one of the curves in the u-curves-objects-lists; tangency-type = one of {'vector 'vectors 'surfaces 'plane}; (list ...) = A list of options depending on tangency-type. List of options for:

```
;;tangency-type = 'vector
;;==========================
(list vector multiplier)

;;tangency-type = 'vectors
;;==========================
(list
  (list point-coords vector multiplier)
  (list point-coords vector multiplier)
  (list point-coords vector multiplier)
  ...)

;;tangency-type = 'surfaces
;;==========================
(list
  (list surface-object multiplier)
  (list surface-object multiplier)
  (list surface-object multiplier)
  ...)

;;tangency-type = 'plane
;;==========================

(list normal-vector multiplier)
```

Default Formula: nil

- **ucurves-orient-points-set-list**

  When specified, it is used to orientat the u curves. It requires a start point and second point for each ucurve in the following format:

```
(list (list start-point-ucurve-1 second-point-ucurve-1)
     (list start-point-ucurve-2 second-point-ucurve-2)
     ...)
```

  If this property is nil and v curves are given, the first two v curves will be used to orient the u curves.

  Default Formula: nil

- **v-curves-objects-list**

  List of flow curve objects along the v direction. Each v curve has to be tangent continuous.

  Default Formula: nil

- **v-curves-tangency-specifications-lists**

  Same format as u-curves-tangency-specifications-lists.

  Default Formula: nil

- **surface-topology-method**

  Specifies the desired topology of the surface.

  Default Formula: (nth 0 '(:minimal :columns :grid))

- **periodic?**

  If the first u curve and the last u curve are the same, when periodic? is t, it creates a periodic smooth surface.

  Default Formula: nil

- **tolerance**

  Used if approximation of the u curves is needed.

  Default Formula: 1.0e-5

Example 4-57. Surface-from-uv-curves-class Example

```
(define-class surface-from-uv-curves-sample-class
  :inherit-from (object)
  :subobjects (
               (u-curve-1 :class 'line-object
                          point1 '(-5 0 -2.5)
                          point2 '(5  0 -2.5)
                          )
               (u-curve-2 :class 'line-object
                          point1 '(-5 0 2.5)
                          point2 '(5  0 2.5)
                          )
               (v-curve-1 :class 'line-object
                          point1 '(-5 0 -2.5)
                          point2 '(-5 0 2.5)
                          )
               (v-curve-2 :class 'line-object
                          point1 '(5  0 -2.5)
                          point2 '(5  0 2.5)
                          )
               (extrusion-1 :class 'extrusion-object
                            swept-object ^^u-curve-1
                            vector       '(0 1 -1)
                            distance     4
                            )
               (extrusion-2 :class 'extrusion-object
                            swept-object ^^u-curve-2
                            vector       '(0 1 1)
                            distance     4
                            )
               (surface-from-u-curves :class 'surface-from-uv-curves-class
```

```
                                        u-curves-objects-list (list ^^u-curve-1
^^u-curve-2)
                                        )
               (surface-from-uv-curves :class 'surface-from-uv-curves-class
                                        u-curves-objects-list (list ^^u-curve-
1 ^^u-curve-2)
                                        v-curves-objects-list (list ^^v-curve-
1 ^^v-curve-2)
                                        )
               (surface-from-u-curves-with-vector-tangency :class 'surface-
from-uv-curves-class
                                                           u-curves-objects-
list (list ^^u-curve-1 ^^u-curve-2)
                                                           u-curves-tangency-
specifications-lists
                                                           (list
                          (list ^^u-curve-1 'vector (list '(0 -1 1) 2.0))
                          (list ^^u-curve-2 'vector (list '(0 -1 -1) 4.0)))
                                                           )
               (surface-from-u-curves-with-surface-tangency :class 'surface-
from-uv-curves-class
                                                            u-curves-objects-
list (list ^^u-curve-1 ^^u-curve-2)
                                                            u-curves-
tangency-specifications-lists
                                                            (list
                                                             (list ^^u-curve-
1 'surfaces (list (list ^^extrusion-1 2.0)))
                                                             (list ^^u-curve-
2 'surfaces (list (list ^^extrusion-2 4.0))))
                                                            )
               (surface-from-u-curves-with-plane-tangency :class 'surface-
from-uv-curves-class
                                                          u-curves-objects-
list (list ^^u-curve-1 ^^u-curve-2)
                                                          u-curves-tangency-
specifications-lists
                                                          (list
                                                           (list ^^u-curve-1
'plane (list '(0 0 1) 4.0))
                                                           (list ^^u-curve-2
'plane (list '(0 0 1) 4.0)))
                                                          )
               )
  )
```

cross-section-along-curve [class]

This class is used for inheritance purposes only. Any object inheriting from this class and a geometry object describing a cross section is automatically oriented along the given curve using the properties specified. The properties below describe how the cross section is oriented. All these properties except point-on-curve are automatically computed when used to generate a series of cross sections along the curve, hence they should not be changed by the user.

Inheritance: graphic-object

Properties:

- **curve-object**

  This is the curve along which the cross section is oriented.

- **point-on-curve**

  Specifies the point on the cross section which should be on the curve.

  Default Formula: '(0 0 0)

- **point-on-path**

  Specifies the location of the cross section along the curve. The cross section is moved such that the point-on-curve coincides with the point specified by this property.

- **normal**

  The local z axis of the cross section is aligned with this direction. This direction is computed as the tangent to the curve at point-on-path.

- **y-dir**

  The local y-axis of the cross section is aligned with this direction.

- **cross-section-normal**

  Vector in global coorinates on the cross-section aligned with the tangent to the path curve.

  Default Formula: '(0 0 1)

- **cross-section-y-dir**

  Vector in global coorinates on the cross-section aligned with the ydirection of the path curve.

  Default Formula: '(0 1 0)

circular-cross-section-along-curve [class]

This object defines a circular cross section to be used to generate a series of cross sections along a specified curve.

Inheritance: cross-section-along-curve, circle-object

cross-sections-along-curve [class]

This object is used to create a series of cross sections placed at specified locations along a curve. The only type of cross section already defined is the circular-cross-section-along-curve. If another type of cross section needs to be specified, it can be defined by inheriting from both a cross-section-along-curve and a geometric object defining the cross section geometry. For example, an elliptical cross section can be defined by inheriting from cross-section-along-curve and an ellipse-object.

Inheritance: series-object

Properties:

- **cross-section-type**

  Default Formula: 'circular-cross-section-along-curve

- **cross-section-init-form**

  A quoted list specifying the properties and formulae for the subobjects of the series.

  Default Formula: nil

- **num-cross-sections**

  Number of cross sections to be placed along path-curve.

  Default Formula: 10

- **path-curve**

  The curve describing the path along which the cross sections are to be placed.

  Default Formula: nil

- **points**

  Specifies the locations at which each of the cross sections is placed along path-curve.

  Default Formula: (generate-points-on-curve ^path-curve ^num-cross-sections)

- **y-dir**

  This is used as the direction for aligning the local y axis of the first cross section. The default is an arbitrary normal to the tangent computed at the first point on the curve. The y-dir property for the following cross sections are computed using the y-dir for the previous cross section.

- **cross-section-normal**

  Vector in global coordinates on the cross-section aligned with the tangent to the path curve.

  Default Formula: '(0 0 1)

- **cross-section-y-dir**

  Vector in global coordinates on the cross-section aligned with the y-direction of the path curve.

  Default Formula: '(0 1 0)

Polygonal Surfaces (Webs)

The VGL microtopology module provides AML the capability of working with faceted, polygonal or approximate representation of geometry. The faceted representation can be triangular, quadrilateral, or a combination of both, for surfaces.

It also enables the creation of geometry from mesh or grid geometry. Using microtopology, and its WEB data structures, the user can work with complex geometries. Some of the applications of the microtopology module in AML are:

1. The surface creation data is given as facets, such as in an STL file, or CAT scan data, or in the form of points that can be triangulated.

2. The surface creation data is given as a mesh, that needs to be converted to a geometric model.

3. Surface data given for NURB representations is too large in which case, a WEB representation is more efficient for booleans as well as for surface creation.

4. Surface data given for NURB representations is too rough, and WEBS can be easier to create.

5. Complex editing operations need to be performed on surfaces such as modifying their shape by simplification, adding complexity, or changing sub-regions.

Once surfaces or curves have been created using webs, they can participate in solid modeling operations or geometric operations using the same VGL functions and AML graphic and geometric objects.

Further, these surfaces can be modified by adding or deleting facets or vertices, moving vertices, or reducing entire facets to a single point.

Classes

web-object [class]

Any web object inherits from this generic web class. Note: An instance of this should not be created, this is just an abstract class.

Inheritance: geom-object

web-surface-object [class]

This class takes point and connectivity data for a surface, and generates the web surface as its geometry. The web surface can have either triangles, or quadrilaterals, or both. If the property cli-format? is t, the method property is ignored. The method property can take the following values: 0 => Triangular data only, nodes file with x,y,z per line, 3 lines form 1 triangle. 1 => Quadrilateral data only, nodes file with x,y,z per line, 4 lines form 1 quad. 2 => nodes file with x,y,z, con-file with connectivity given through indices. 4 => XML input. When specifying connectivity, try to ensure that there are no duplicate vertices, although if you set the property cleanup? to t then it removes duplicate nodes. This does not work in all cases yet. If you do not have duplicate vertices, explicitly set cleanup? to nil. Note that the indices in the con-file for method 2 begin from 0.

Inheritance: web-object

Properties:

- **cli-format?**

  Specifies if it is SPECTRUM CLI format.

  Default Formula: nil

- **method**

  Specifies the input data format.

  Default Formula: 2

- **nodes-file**

  Gives the point data for a web surface.

  Default Formula: "/tmp/nodes"

- **con-file**

  Gives the connectivity information for the surface.

  Default Formula: "/tmp/con"

- **cleanup?**

Specifies if it should remove duplicate nodes.

Default Formula: t

- **quad-element?**

  When t, a quad element in the web geom is created as a patch face with four edges, otherwise, it is created as two triangular patches.

  Default Formula: nil

Example 4-58. Web-surface-object Example

```
(define-class tri-web-surface
  :inherit-from (web-surface-object)
  :properties (
               cleanup? t
               nodes-file "nodes-tri.dat"
               method 0
               )
  )
(define-class quad-web-surface
  :inherit-from (web-surface-object)
  :properties (
               cleanup? t
               nodes-file "nodes-quad.dat"
               method 1
               )
  )
(define-class mixed-web-surface
  :inherit-from (web-surface-object)
  :properties (
               cleanup? nil
               nodes-file "nodes.dat"
               con-file "con.dat"
               method 2
               )
  )
(define-class vis-file-web
  :inherit-from (web-surface-object)
  :properties (
               cli-format? t
               cleanup? nil
               method 2
               nodes-file "vis-file.dat"
               )
  )
```

Example Data Files: 1) "nodes-tri.dat"

```
0 0 0
1 0 0
1 1 0
1 1 0
```

```
2 1 0
2 2 0
```

2) "nodes-quad.dat"

```
0 0 0
1 0 0
0 1 0
1 1 0
1 1 0
2 1 0
1 2 0
2 2 0
```

3) "nodes.dat"

```
0 0 0
1 0 0
2 0 0
3 0 0
0 1 0
1 1 0
2 1 0
3 1 0
```

4) "con.dat"

```
0 1 4 5
1 2 5 6
2 3 7
```

5) "vis-file.dat"

```
problem "inflow_vents"
  time information
    number of time steps 1
    time steps
    1 0
    end time steps
  end time information

  region "inflow_vents"
    nature "fluid"
    number of nodal coordinates 18
    nodal coordinates
      1     2.31770E+03  -5.75673E+02   1.20591E+03
```

```
      2     2.31099E+03  -5.76688E+02   1.18985E+03
      3     2.30417E+03  -5.77648E+02   1.17382E+03
      4     2.31741E+03  -5.53308E+02   1.20603E+03
      5     2.31069E+03  -5.54977E+02   1.18996E+03
      6     2.30387E+03  -5.56595E+02   1.17393E+03
      7     2.31712E+03  -5.30944E+02   1.20615E+03
      8     2.31039E+03  -5.33267E+02   1.19007E+03
      9     2.30357E+03  -5.35541E+02   1.17403E+03
     10     2.31629E+03   4.79643E+02   1.20683E+03
     11     2.31713E+03   5.04125E+02   1.20692E+03
     12     2.31797E+03   5.28608E+02   1.20701E+03
     13     2.30935E+03   4.79426E+02   1.19053E+03
     14     2.30231E+03   4.79136E+02   1.17427E+03
     15     2.31009E+03   5.03903E+02   1.19065E+03
     16     2.30295E+03   5.03616E+02   1.17441E+03
     17     2.31082E+03   5.28380E+02   1.19077E+03
     18     2.30359E+03   5.28097E+02   1.17456E+03
    end node coordinates
    element set "inflow_vents"
      material id 0
      nodes per element 4
      topology "quad"
      number of elements 8
      connectivity
        1        2        5        4        1
        2        3        6        5        2
        3        5        8        7        4
        4        6        9        8        5
        5       13       15       11       10
        6       14       16       15       13
        7       15       17       12       11
        8       16       18       17       15
    end connectivity
   end element set
  end region

end problem
```

web-line-object [class]

Takes a list of points and generates a 1-dimensional web.

Inheritance: web-object

Properties:

- **vertices**

  The vertices needed to define the line.

  Default Formula: nil

Example 4-59. Web-line-object Example

```
(define-class web-line-example
  :inherit-from (web-line-object)
  :properties (
        vertices '((0 0 0) (1 0 0) (2 3 0) (5 6 0))
        )
 )
```

web-from-geom-object [class]

Takes a source object giving the geometry to be converted to a web, and based on 'pt-tol' and 'ang-tol', generates facets to be used as web input data, and creates the web representation for the source object. code 1 => Use Angular Tolerance. 2 => Use Point Tolerance. 4 => Use Angular Tolerance and Point Tolerance.

Inheritance: web-object

Properties:

- **source-object**

  Non-web source object for generating web.

  Default Formula: nil

- **pt-tol**

  Point tolerance for faceting.

  Default Formula: 0.1

- **ang-tol**

  Angular tolerance for faceting.

  Default Formula: 0.25

- **code**

  Determines whether to use point or angular tol.

  Default Formula: 1

Example 4-60. Web-from-geom-object Example

```
(define-class web-from-geom-example
  :inherit-from (web-from-geom-object)
  :properties (
        source-object ^cylinder
        )
  :subobjects (
        (cylinder :class 'cylinder-object
        display? nil
    )
        )
 )
```

web-surface-from-curves-object [class]

Takes a set of curves, generates 'n-points' points on each, generates rectangular facets and creates a web surface.

Inheritance: web-surface-object

Properties:

- **curve-objects**

  List of curves for generating web.

  Default Formula: nil

- **n-points**

  Number of points to be generated on each curve.

  Default Formula: 100

- **use-files?**

  If this property is set to t, then the points and connectivity are first written to files and the web surface is created by reading the file data. If it is set to nil, the web surface is created directly from the data in memory and is much faster. However, in case of large number of points, it is recommended to set this property to t.

  Default Formula: t

Example 4-61. Web-surface-from-curves-object Example

```
(define-class web-surface-from-curves-example
  :inherit-from (web-surface-from-curves-object)
  :properties (
        cleanup? nil
        curve-objects (list ^curve1 ^curve2)
        )
  :subobjects (
        (curve1 :Class 'curve-skin-object
          points-list '((0 0 0) (1 0 0) (3 2 0) (5 7 0))
          display? nil
      )
        (curve2 :Class 'curve-skin-object
          points-list '((1 0 10) (3 0 11) (5 2 12) (9 7 13))
          display? nil
      )
        )
  )
```

web-surface-from-points-object [class]

Takes the list of point lists for several curves, and generates facets between pairs of curves, and hence a web surface. It creates quad facets as long as there are an equal number of points and then matches the rest of the points using triangles. (See Example)

Inheritance: web-surface-object

Properties:

- **points-list**

  List of list of points for each curve.

  Default Formula: nil

- **mesh-data**

  It is a list of 6 values in the following order: number of nodes, nodesptr, number of triangles, tris-ptr, number of quadilaterals, and quads-ptr. Nodes-ptr, tris-ptr, and quads-ptr are data pointers. This property is to be used for information only and should not be changed.

  Default Formula: (if (the superior use-files?) (when (the web-data) (vgl::create-generic-web-pointers (the nodes-file) :con-file (the con-file) :method (the method) :cleanup (the cleanup?))) (vgl::create-web-pointers-from-points-lists (the superior points-list)))

- **use-files?**

  If this property is set to t, then the points and connectivity are first written to files and the web surface is created by reading the file data. If it is set to nil, the web surface is created directly from the data in memory and is much faster. However, in case of large number of points, it is recommended to set this property to t.

  Default Formula: t

Example 4-62. Web-surface-from-points-object Example

```
(define-class web-surface-from-points-example
  :inherit-from (web-surface-from-points-object)
  :properties (
        points-list '(
           ((0 0 0) (1 1 0) (2 3 0) (3 5 0))
           ((0 0 2) (1 1 2) (2 3 2) (3 5 2) (4 5 2))
           ((0 0 4) (1 1 4) (2 3 4) (3 5 4) (4 5 4) (5 6 4))
           )
        cleanup? nil
        )
)
```

web-from-edges [class]

Takes the 2 points lists, and creates a web by generating rectangular and triangular facets.

Inheritance: web-surface-object

Properties:

- **Points-list1**

  Points for first curve.

  Default Formula: nil

- **Points-list2**

  Points for second curve.

Default Formula: nil

Example 4-63. Web-from-edges Example

```
(define-class web-from-edges-example
  :inherit-from (web-from-edges)
  :properties (
        points-list1 '((0 0 0) (1 1 0) (2 3 0) (3 5 0))
        points-list2 '((0 0 2) (1 1 2) (2 3 2) (3 5 2) (4 5 2))
        cleanup? nil
        )
  )
```

Functions

create-web-data [Function]

Takes a list of curves, generates 'n' points on each curve along its parameter space and generates node and connectivity files for a web between these curves.

Arguments:

- **curves**

  List of curves between which the web is to be generated.

- **n**

  Number of points approximating each curve.

- **crds-file**

  Nodes file for the web.

- **con-file**

  Connectivity file for the web.

create-web-data-from-point-sets [Function]

Takes 2 lists of points and generates node and connectivity files for a web between curves represented by these lists, by creating triangular and quadrilateral facets.

Arguments:

- **points-list1**

  First list of points.

- **points-list2**

  Second list of points.

- **crds-file**

  Nodes file for the web.

- **con-file**

Connectivity file for the web.

create-web-data-from-points [Function]

Takes lists of points and generates node and connectivity files for a web between curves represented by these lists by creating facets.

Arguments:

- **curves-points-list**

  List of points representing curves.

- **crds-file**

  Nodes file for the web.

- **con-file**

  Connectivity file for the web.

Booleans

Classes

boolean-object [class]

The boolean-object is the general object that other more specific boolean operation objects inherit from.

Inheritance: geom-object

Properties:

- **object-list**

  Expects a list of graphic-object instances.

  Default Formula: nil

- **simplify?**

  t removes common boundaries in the geometry, nil leaves the common boundaries.

  Default Formula: t

difference-object [class]

The difference-object will create a single geometric instance consisting of only the regions of two objects that are not in common. If the first object in the property object-list is a NULL geom, then the result is a NULL geom.

Inheritance: boolean-object

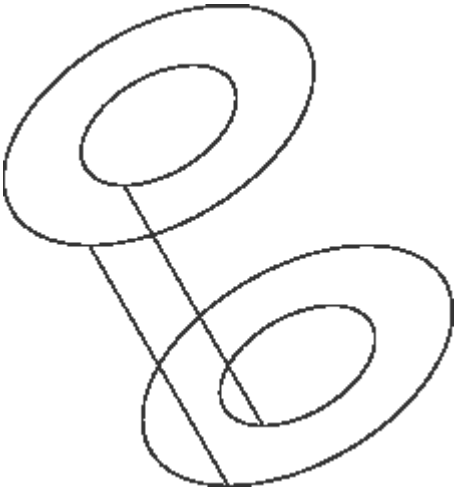
Properties:

- **simplify?**

  t removes common boundaries in the geometry, nil leaves the common boundaries. It is treated internally as nil if the dimensions are mixed.

  Default Formula: t

Figure 4-31. Difference-object



The tube object is created as the difference of two cylinders.

Example 4-64. Difference-object example

```
(define-class TUBE
  :inherit-from (difference-object)
  :properties (
               id 0.5
               od 1.0
               height 1.0
               color 'wheat
               render 'shaded
               display? t
               object-list (list (the stock) (the hole))
               )
  :subobjects (
               (stock :class 'cylinder-object
                      height ^^height
                      diameter ^od
                      solid? t
                      display? nil
                      )
               (hole :class 'cylinder-object
                     height ^^height
                     diameter ^id
                     solid? t
                     display? nil
                     )
               )
  )
```

intersection-object [class]

The intersection-object will create a single geometric instance consisting of only the common regions between any number of given geometric instances. If any object in object-list is NULL geom, the result is NULL geom.

Inheritance: boolean-object

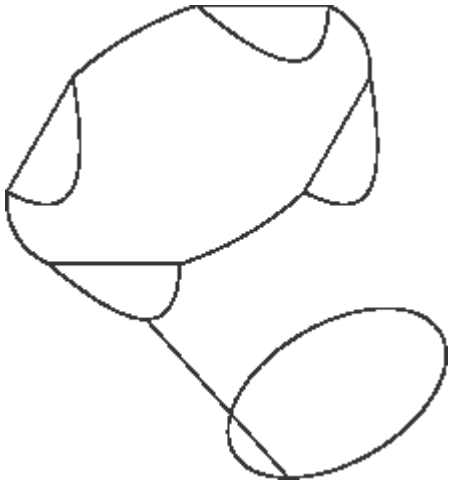
Properties:

- **simplify?**

  t removes common boundaries in the geometry, nil leaves the common boundaries. It is treated internally as nil if the dimensions are mixed.

  Default Formula: t

Figure 4-32. Intersection-object



The figure shows the intersection of the box and cone objects.

Example 4-65. Intersection-object example

```
(define-class INTERSECTION-EXAMPLE
  :inherit-from (intersection-object)
  :properties (
               object-list (list (the part) (the shaft))
               )
  :subobjects (
               (part :class 'box-object
                     solid? t
                     display? nil
                     )
               (shaft :class 'cone-object
                      height 4.0
                      diameter 1.75
                      solid? t
                      display? nil
                      )
```

```
                )
  )
```

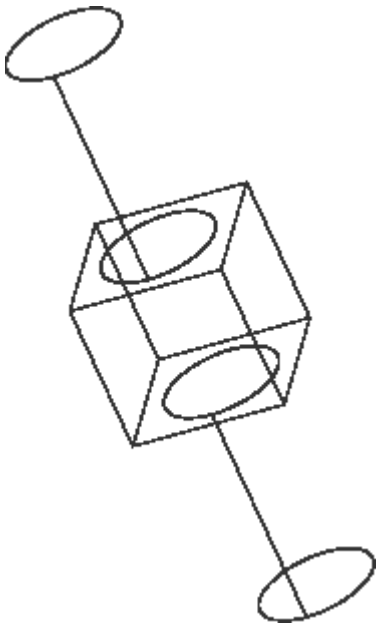
When you mix dimensions (say a solid cylinder and a sheet), we have to specify the object with the lower dimension as the first entity in the object-list.

union-object [class]

The union-object will create a single geometric instance by adding two geometric instances together.

Inheritance: boolean-object

Figure 4-33. Union-object



The union of a box and cylinder object.

Example 4-66. Union-object example

```
(define-class UNION-EXAMPLE
  :inherit-from (union-object)
  :properties (
               object-list (list (the part) (the shaft))
               )
  :subobjects (
               (part :class 'box-object
                     solid? t
                     display? nil
                     )
               (shaft :class 'cylinder-object
                      height 4.0
                      diameter 0.75
                      solid? t
                      display? nil
```

)
)
)

Sweeps

Classes

extrusion-object [class]

The extrusion-object will create a translation sweep along a vector. The resulting geometry may be solid or shell and capped or not capped. When the geometry is solid it must be capped.

Inheritance: geom-object solid-object

Properties:

- **swept-object**

  The object that is to be extruded.

  Default Formula: nil

- **vector**

  The direction to extrude.

  Default Formula: (default '(0.0 0.0 1.0))

- **distance**

  The length of extrusion.

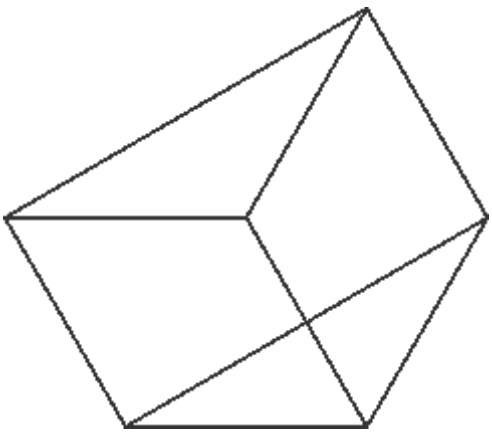
  Default Formula: (default 1.0)

extrude-in-both-directions?

If this is set to t, the extrusion will happen along the extrusion vector and in the opposite direction.

Default Formula: nil

Figure 4-34. Extrusion-object



The sewn triangular plane is extruded a distance of 10 along the z-direction.

Example 4-67. Extrusion-object example

```
(define-class example-extrusion-class
  :inherit-from (extrusion-object)
  :properties (
               swept-object ^shape
               vector '(0.0 0.0 10.0)
               distance 1.0
               capped? t
               )
  :subobjects (
               (shape :class 'polygon-object
                        vertices '((0 0 0) (1 0 0) (0 1 0))
               )
  )
```

general-sweep-class [class]

This class creates a sheet or solid body by sweeping a set of wire or sheet profiles along a curve path.

Inheritance: geom-object

Properties:

- **profile-objects-list**

  A list of profile objects for sweeping, could be wire or sheet (face).

  Default Value: nil

- **path-points-coords-list**

  These are the corresponding points on the path-object for each profile.

  Default Value: nil

- **profile-match-point-coords-list**

  Defines a matching vertex coordinates on each profile with the neighboring profile.

  Default Value: nil

- **simplify?**

  Needs to be nil if more than one profile exists in profile-objects-list.

  Default Value: t

- **swept-object**

  Only used for backward compatibility.

- **path-object**

  Object instance specifying the sweep path. It should be an instance whose geom is a single curve. The sweep direction is based on the construction of the path object.

- **start-point**

  Only used for backward compatibility.

- **tangential-sweep?**

When non-nil, the sweep is tangential, i.e. the swept profile orients in order to maintain a constant angle relative to the tangent to the path at every point.

Default Formula: nil

- **topology-form**

  Three values are allowed: 0 to produce minimum number of faces, 1 to produce columns of faces corresponding to the geometry of the swept-object , and 2 to produce a grid of faces.

  Default Formula: 0

- **twist-points**

  A list of point coordinates on the path curve where twist values will be applied.

  Default Formula: nil

- **twist-values**

  A list of twist angles (in radians) corresponding to the twist-points list specified. A twist is a rotation of the swept profile about the tangent to the path.

  Default Formula: nil

- **scale-points**

  A list of point coordinates on the path curve where scale values will be applied.

  Default Formula: nil

- **scale-values**

  A list of scale factors corresponding to the scale-points list. A scalue value must not be zero. Scale values will scale the profile at the corresponding location on the path.

  Default Formula: nil

Note: To generate a point that lies exactly on a curve/surface, refer to the method find-closest-point-on-object.

Example 4-68. general-sweep-class example

```
 ;;;; ------------------------------------------------------------------------
---
;;;;                                                    11180 Reed Hartman
Hwy
;;;;               TechnoSoft Inc.                      Cincinnati, OH
45242
;;;;             Copyright (c) 1993 - 2009              Phone: (513) 985-
9877
;;;;                                                    Fax:   (513) 985-
0522
;;;;
;;;;
;;;; This code sample is provided as an example of how to perform a
particular
;;;; kind of task using foundation software provided by TechnoSoft.  It is
;;;; provided "as is" without warranty of any kind, expressed or implied.
;;;;
;;;; Licensed users of the Adaptive Modeling Language are free to use and
```

```
;;;; modify this code as long as the the original credits and disclaimers are
;;;; maintained.
;;;; ------------------------------------------------------------------------
--
;;;;  Author: Trapper Schuler
;;;;  Created: Thu Jul 02 10:58:44 2009
;;;; ------------------------------------------------------------------------
--
;;;;  Purpose: general-sweep-class examples.
;;;;

(in-package :aml)

;;;-----------------------------------------------------
;;; Class   : general-sweep-example-1-class
;;; Inherit :
;;; Purpose : Create a spring coil based on coil length, number of coils,
coil diameter, and wire diameter.
;;; Notes   : Remember to shade the spring-coil-object instance.
;;; Author  : Trapper Schuler
;;; History
;;;   Created on :
;;;   Modified   :
;;;
(define-class general-sweep-example-1-class
  :inherit-from (object)
  :properties (
               coil-length 5
               number-of-coils 10
               coil-diameter 1
               wire-diameter 0.1
               )
  :subobjects (
               (coil-path-object :class 'line-object
                                 point1 '(0 0 0)
                                 point2 (list 0 ^^coil-length 0)
                                 )
               (coil-swept-object :class 'disc-object
                                  diameter ^^wire-diameter
                                  orientation (list
                                               (translate (list (half ^^coil-
diameter) 0 0))
                                               )
                                  )
               (spring-coil-object :class 'general-sweep-class
                                   path-object ^^coil-path-object
                                   swept-object ^^coil-swept-object
                                   start-point (vertex-of-object
                                                ^^coil-path-object
                                                0
                                                )
                                   twist-points (list
                                                 (vertex-of-object
```

```
                                                    ^^coil-path-object
                                                    0
                                                    )
                                                   (vertex-of-object
                                                    ^^coil-path-object
                                                    1
                                                    )
                                                   )
                                        twist-values (list
                                                    0
                                                    (* ^^number-of-coils PI)
                                                    )
                                        tangential-sweep? t
                                        )
                    )
  )

(define-method property-classification-list general-sweep-example-1-class ()
  '(
    ("Main Properties"
     (
      coil-length
      number-of-coils
      coil-diameter
      wire-diameter
      )
     )
    )
  )

;;;-----------------------------------------------------
;;; Class   : general-sweep-example-2-class
;;; Inherit :
;;; Purpose : Create a spring coil based on coil length, number of coils,
coil diameter, and wire diameter.  This example also makes use of the scale-
points and scale-values properties of the 'general-sweep-class class.
;;; Notes   : Remember to shade the spring-coil-object instance.
;;; Author  : Trapper Schuler
;;; History
;;;   Created on :
;;;   Modified   :
;;;
(define-class general-sweep-example-2-class
  :inherit-from (object)
  :properties (
               coil-length 5
               number-of-coils 10
               coil-diameter 1
               wire-diameter 0.1
               scale-value 2
               )
  :subobjects (
               (coil-path-object :class 'line-object
```

```
                                point1 '(0 0 0)
                                point2 (list 0 ^^coil-length 0)
                                )
                (coil-swept-object :class 'disc-object
                                   diameter ^^wire-diameter
                                   orientation (list
                                                (translate (list (half ^^coil-
diameter) 0 0))
                                                )
                                   )
                (spring-coil-object :class 'general-sweep-class
                                    path-object ^^coil-path-object
                                    swept-object ^^coil-swept-object
                                    start-point (vertex-of-object
                                                 ^^coil-path-object
                                                 0
                                                 )
                                    twist-points (list
                                                  (vertex-of-object
                                                   ^^coil-path-object
                                                   0
                                                   )
                                                  (vertex-of-object
                                                   ^^coil-path-object
                                                   1
                                                   )
                                                  )
                                    twist-values (list
                                                  0
                                                  (* ^^number-of-coils PI)
                                                  )
                                    scale-points (list
                                                  ;; Point 1
                                                  (vertex-of-object
                                                   ^^coil-path-object
                                                   0
                                                   )
                                                  ;; Point 2
                                                  (mid-point
                                                   (vertex-of-object
                                                    ^^coil-path-object
                                                    0
                                                    )
                                                   (vertex-of-object
                                                    ^^coil-path-object
                                                    1
                                                    )
                                                   )
                                                  ;; Point 3
                                                  (vertex-of-object
                                                   ^^coil-path-object
                                                   1
                                                   )
```

```
                                        )
                                scale-values (list
                                        1
                                        ^^scale-value
                                        1
                                        )
                                tangential-sweep? t
                                )
            )
  )

(define-method property-classification-list general-sweep-example-2-class ()
  '(
    ("Main Properties"
     (
      coil-length
      number-of-coils
      coil-diameter
      wire-diameter
      scale-value
      )
     )
    )
  )
```

rotation-sweep-object [class]

The rotation-sweep-object is used to revolve an object instance about a vector. If a surface (2-dimensional geometry) is revolved the resulting object will be a solid (3-dimensional geometry) and if a line (1-dimensional geometry) is revolved the result will be a surface (2-dimensional geometry).

Inheritance: geom-object

Properties:

- **swept-object**

  The object instance that is to be swept.

- **angle**

  The angle to sweep the swept-object. The measurement is in degrees.

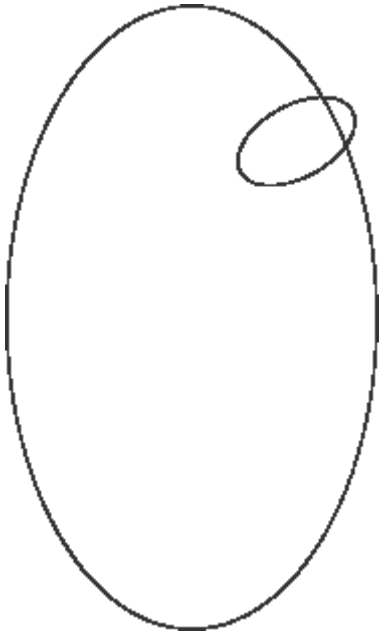
- **axis-point**

  The point defining the sweep axis start point.

- **axis-vector**

  The vector to sweep about.

Figure 4-35. Rotation-sweep-object



The circle is swept about the y-axis through the origin to produce the torus.

Example 4-69. Rotation-sweep-object Example

```
(define-class TORUS
  :inherit-from (rotation-sweep-object)
  :properties (
        swept-object !cross-section
        axis-point '(0.0 0.0 0.0)
        axis-vector '(0.0 1.0 0.0)
        )
  :subobjects (
        (cross-section :class 'circle-object
                diameter 1.0
                orientation (list (translate '(2.0 0.0 0.0)))
                )
        )
  )
```

tangential-sweep-object [class]

The tangential-sweep-object provides the ability to sweep an object along an arbitrary path. In a tangential sweep, the swept profile orients in order to maintain a constant angle relative to the tangent to the path at every point. If a surface (2-dimensional geometry) is swept the resulting object will be a solid (3-dimensional geometry) and if a line (1-dimensional geometry) is swept the result will be a surface (2-dimensional geometry).

Inheritance: geom-object

Properties:

- **swept-object**

  The object instance to be swept.

  Default Formula: nil

- **path-object**

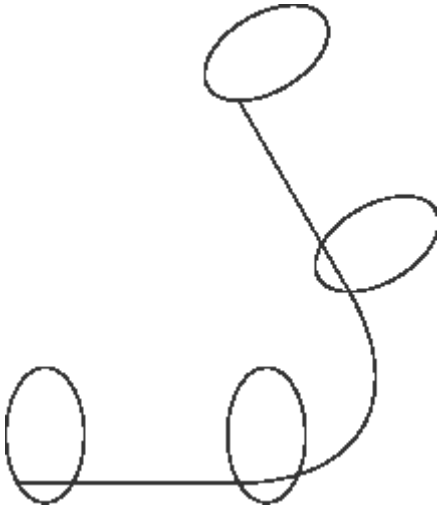
  The object instance that will be the path to sweep along. This path must contain a vertex at the point from which the sweptobject will start from. For example, a circle-object does not have any vertices, so the user would have to imbed a point-object in the circle-object using an instance of an imprint-class.

  Default Formula: nil

- **reference-point**

  The starting point for the path. The swept-object should be positioned at the reference-point.

Figure 4-36. Tangential-sweep-object



The circle is swept along the tangent to the sewn-object.

Example 4-70. Tangential-sweep-object Example

```
(define-class TANGENTIAL-SWEEP
  :inherit-from (tangential-sweep-object)
  :properties (
               swept-object !hoop
               path-object !path
               )
  :subobjects (
               (hoop :class 'circle-object
                     diameter 0.25)
               (path :class 'sewn-object
                     object-list (list !segment1 !segment2 !segment3))
               (segment1 :class 'line-object
                         point1 '(0.0 0.0 0.0)
```

```
                    point2 '(0.0 0.0 0.5))
               (segment2 :class 'arc-object
                    start-angle 0.0
                    end-angle 90.0
                    radius 0.5
                    orientation (list (rotate -90.0 '(0.0 1.0 0.0))
                                      (rotate 90.0 '(1.0 0.0 0.0))
                                      (translate (vector 0.0 0.5 0.5)))
                    )
               (segment3 :class 'line-object
                    point1 '(0.0 0.5 1.0)
                    point2 '(0.0 1.0 1.0))
               )
)
```

translation-sweep-object [class]

The translation-sweep-object is an extrusion along an arbitrary path object. In a translation sweep, the swept profile does not change orientation throughout the sweep; it does not adjust with the tangent to the path. If a surface (2-dimensional geometry) is translated the resulting object will be a solid (3-dimensional geometry) and if a line (1-dimensional geometry) is translated the result will be a surface (2-dimensional geometry).

Inheritance: geom-object

Properties:

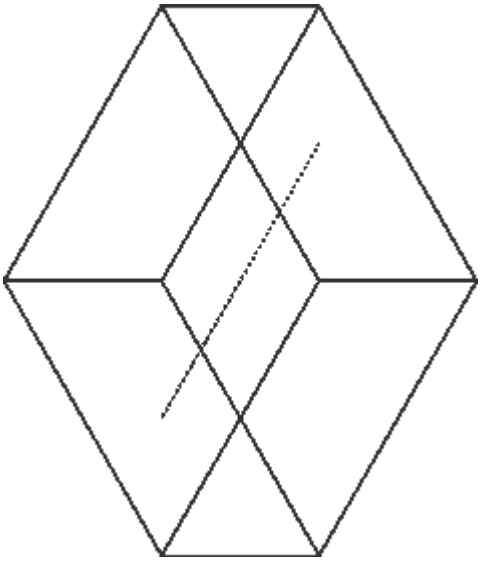
- **swept-object**

  The object instance that will be swept.

- **path-object**

  The object instance that will be the sweep path.

Figure 4-37. Translation-sweep-object



The prism is obtained by performing a translation-sweep of the sheet along the line-object.

Example 4-71. Translation-sweep-object Example

```
(define-class PRISM
  :inherit-from (translation-sweep-object)
  :properties (
               swept-object !cross-section
               path-object !sweep-path
               )
  :subobjects (
               (cross-section :class 'sheet-object
                              height 1.0
                              width 2.0
                              )
               (sweep-path :class 'line-object
                           point1 (vector 0.0 0.0 0.0)
                           point2 (vector 1.0 1.0 1.0)))
  )
```

Geom control functions

Functions

The functions described in this section deal with geometry entities through geom IDs. Geom IDs are stored in the geom property of graphic objects or can be generated by direct calls to the virtual geometry layer creation functions (see Appendix D). Working at the level of geom IDs should only by performed by advanced AML users.

draw-geom [Function]

Draws a geometry entity given its geom ID.

Arguments:

- **geom-id**

  Geom ID of the geometry to draw.

- **draw-type**

  Drawing types allowed are: 'boundary, 'shaded, 'isoline (OPENGL SHAPES server), 'facet, 'boundary-shaded, 'facet-shaded, 'isolineshaded (OPENGL SHAPES server).

- **color**

  Color to draw with. Can be a color name (see appendix C) or a list of RGB values between 0 and 1 (inclusive).

Keyword Arguments:

- **:line-width**

  Integer specifying the width (in pixels) of the lines used to draw the object.

  Default Value: 0 (Thinnest).

- **:line-type**

  Integer specifying the style of the lines used to draw the object. The predefined line types range from 0 to 6: solid, dashed, dotted, dashed-dotted, dasheddouble-dotted, dashed-tri-dotted, long dash. Any value above 6 (32bit integer) defines the pattern, bit 1 means the pixel is on, 0 means the pixel is off.

  Default Value: 0

find-closest-point-on-geom [Function]

The function finds the closest point on a geom to the input point and returns its coordinates. The input point MUST lie within a distance "tolerance" of the geom otherwise the function returns nil. The function behaves exactly like the method find-closest-point-on-object. The function will return the parameters of the vertex on the geom corresponding to that point, if the values? flag is true. In that case, the return value is a list of (closest-point vertex-parameters).

Arguments:

- **geom**

  Input geom to find the closest point on.

- **point**

  Input point for which the closest point is being found on the geom.

- **tolerance**

  Distance from the surface within which the input point must lie to be able to find the closest point.

Keyword Arguments:

- **values?**

  Defaults to nil, if t, the returned value is a list of (closest-point vertex-parameters). If nil, it returns the coordinates of the closest point.

Example 4-72. Find-closest-point-on-geom Example

```
(create-model 'name-generator)
(add-object (The) 'sheet 'sheet-object)

(find-closest-point-on-geom (the geom (:from !sheet)) '(0 0 0.0001) 0.001)
> (0.0 0.0 0.0)

(find-closest-point-on-geom (the geom (:from !sheet)) '(0 0 0.0001) 0.001
:values? t)
> ((0.0 0.0 0.0) (0.0 0.0))
```

generate-distributed-points-between-two-points [Function]

This function behaves similarly to generate-distributed-points-along-curve-objects and generates a set of points along the straight line between two specified points following a specified distribution between the two points.

Arguments:

- **start-point-coords**

  The coordinates of the start point.

- **end-point-coords**

  The coordinates of the end point.

- **num-points**

  Number of points to be generated.

Keyword Arguments:

- **start-distance**

  Distance of first generated point from the start-point-coords. If nil, the first point is the start-point.

  Default Value: nil

- **end-distance**

  Distance of last generated point from the start-point-coords. If nil, the last point is the end-point.

  Default Value: nil

- **method**

  Default Value: :uniform

- **data**

  Default Value: nil

- **distribution-methods?**

  Default Value: nil

```
(generate-distributed-points-between-two-points '(0 0 0) '(10 0 0) 10
:start-distance 1)

;;;Returns
((1.0 0.0 0.0) (2.0 0.0 0.0) (3.0 0.0 0.0) (4.0 0.0 0.0) (5.0 0.0 0.0) (6.0
0.0
0.0) (7.0 0.0 0.0) (8.0 0.0 0.0) (9.0 0.0 0.0) (10.0 0.0 0.0))
```

generate-equally-spaced-points-along-curve [Function]

Given a curve geom and the number of desired points, this function returns a list of coordinate lists of points that are equally spaced along the desired segment of the curve based on the edge orientation, i.e. the first point would be based on the first k-sub-geom-0. This is different from the curve orientation which is based on the point evaluation on the first value in the curve parameter range.

Arguments:

- **curve-geom**

  Geom ID of the curve. Must be of a single curve geometry.

- **n**

Integer specifying the desired number of points to generate.

Keyword Arguments:

- **:tolerance**

  Specifying the tolerance within which the first and last points are generated with respect to the start distance and end distance provided.

  Default Value: 0.00001

- **:start-distance**

  Offset from the start vertex of the curve where the first point will be generated. This value is a fraction (between 0 and 1) or a position (between 0 and the length of the curve) depending on the :fraction keyword.

  Default Value: 0.0

- **:end-distance**

  Offset from the end vertex of the curve where the last point will be generated. This value is a fraction (between 0 and 1) or a position (between 0 and the length of the curve) depending on the :fraction keyword.

  Default Value: 0.0

- **:fraction?**

  When non-nil, the values provided with the :startdistance and :end-distance keywords are a fraction between 0 and 1. Otherwise, these values are a position between 0 and the length of the curve.

  Default Value: t

generate-point-distribution-along-curve [Function]

Given a curve geom and a point distribution, this function returns a list of point coordinates corresponding to the distribution.

Arguments:

- **curve-geom**

  Geom ID of the curve. Must be a single curve geometry.

- **distribution-list**

  List of values between 0 and 1 if using a fractional distribution or values between 0 and the length of the curve otherwise.

Keyword Arguments:

- **:tolerance**

  specifying the value within which each of the returned points corresponds to the corresponding distribution value.

  Default Value: 0.00001

- **:fraction?**

When non-nil, the distribution list is a list of fractional values between 0 and 1 with respect to the curve. When nil, the distribution list is a list of values between 0 and the length of the curve along the curve.

Default Value: t

generate-single-curve-points-from-curves [Function]

This function generates and returns a list of ordered point coordinates from the input curves by generating 'n' points on each curve. These points could then be passed to a curve-frompoints- class for creating a single curve.

Arguments:

- **curves**

  A list of curve geom IDs from which points are to be generated. They must be single curve geoms.

Optional Arguments:

- **n**

  specifying the number of points to be generated on each curve.

  Default Value: 20

group-ordered-items-in-list-from-marker-items [Function]

The function takes a list of ordered ids and a list of marker-ids, where the marker-ids are a subset of ids from the first list, and returns a list of grouped ids based on the markers. **Notes:**

- The marker ids has to be in the same sequence as the input ordered ids, but the markers do not need to be contiguous in the input ordered ids list.
- The order of the markers can be reversed from the original list (See Example 2).
- The original list is assumed to be closed, but the first item must NOT be repeated at the end.

Arguments:

- **elements-list**

  A list of ordered ids describing a loop of ids.

- **mark-elements-list**

  A list of ids from the elements for grouping.

Example 4-73. Group-ordered-items-in-list-from-marker-items

```
Example 1:
(group-ordered-items-in-list-from-marker-items
   '(1 111 2 112 3 113 4 114 5 115 6 116 7 117 8 118)
   '(5 6 1 3))
;;;RETURNS
 ((5 115 6)
   (6 116 7 117 8 118 1)
   (1 111 2 112 3)
   (3 113 4 114 5))
```

```
Example 2:
(group-ordered-items-in-list-from-marker-items
   '(1 111 2 112 3 113 4 114 5 115 6 116 7 117 8 118)
   '(3 1 6 5))
;;;RETURNS
((3 112 2 111 1)
 (1 118 8 117 7 116 6)
 (6 115 5)
 (5 114 4 113 3))


Note that the items need not be numbers. The function uses the test 'equal.
Example 3:
(group-ordered-items-in-list-from-marker-items
  '(v1 e1 v2 e2 v3 e3 v4 e4 v5 e5 v6 e6 v7 e7 v8 e8)
  '(v5 v6 v1 v3))

RETURNS
((v5 e5 v6)
 (v6 e6 v7 e7 v8 e8 v1)
 (v1 e1 v2 e2 v3)
 (v3 e3 v4 e4 v5))
```

mouse-axis-rotate-geom [Function]

This function allows the user to rotate a geometry entity about an axis given its geom ID and the rotation axis direction. The user can left-click and drag the mouse to perform the rotation. The geom provided must be drawn on the current active display. This function does not change the internal orientation data of the geom, i.e. undrawing it and drawing it again will bring it back to its original orientation. The function's is main use is that it returns the angle (in degress) resulting from the mouse rotation.

Arguments:

- **geom**

  Geom ID to rotate.

- **axis**

  XYZ list specifying the rotation axis.

Keyword Arguments:

- **:ref-point**

  Coordinate list of the rotation reference point.

mouse-translate-geom [Function]

This function allows the user to translate a geometry entity by dragging it with the mouse given its geom ID. The geom provided must be drawn on the current active display. This function does not change the internal position data of the geom, i.e. undrawing it and drawing it again will bring it back to its original

position. The function's main use is that it returns the translation coordinate list resulting from the mouse translation. Note: By default the mouse translation allows arbitrary directions. To constrain the translation direction: Hold the [shift] key down while moving to move along the x-axis only. Hold the [control] key down while moving to move along the y-axis only. Hold the [shift] and [control] keys down while moving to move along the z-axis only.

Arguments:

- **geom**

  Geom ID to translate.

point-at-distance-along-curve [Function]

This function returns information about a point on a curve given the curve's geom ID and the point's offset on the curve. The function returns a list of three values in the following order: parameter-space offset of the point (U value), coordinates of the point in global space (x,y,z list), tolerance. The point returned is always within the tolerance (returned) of the desired distance.

Arguments:

- **geom**

  Curve's geom ID.

- **distance**

  Distance from the parameter space origin of the curve. This distance is a relative value [0 -1] if the keyword :fraction? is non-nil, or an absolute value otherwise.

Keyword Arguments:

- **:total-length**

  Specifying the length of the curve. When the length of the curve is known it is more efficient to specify this argument. When nil, it will cause the function to compute the length internally.

  Default Value: nil

- **:tolerance**

  Specifying the tolerance within which the point coordinates returned match the distance provided.

  Default Value: 0.00001

- **:fraction?**

  When non-nil, the distance argument is a relative value between 0 and 1 (inclusive). The distance argument is an absolute value otherwise.

  Default Value: t

point-geom-coords [Function]

This function returns the coordinate list of a point geom given the geom ID.

select-geom [Function]

This function allows the user to interactively select a geom from a list of drawn geoms.

Arguments:

- **geom-list**

  List of geom IDs to be made available for selection.

Keyword Arguments:

- **:color-name**

  String specifying the name of the color used to highlight selectable geoms.

select-point-on-curve [Function]

This function allows the user to select with the mouse a point on a curve given a desired number of candidate points. The candidate points are highlighted on the curve and the user should left click on the desired point. The function returns the coordinate list of the selected point.

Arguments:

- **curve-geom**

  Geom ID of curve in question. Must be a single curve geom.

Keyword Arguments:

- **:nu**

  Integer specifying the number of candidate points to be generated. The candidate points are equally spaced in the param space of the curve.

  Default Value: 10

- **tolerance**

  Optional argument. Distance within which the candidate points coordinates are located relative to the curve.

  Default Value: 0.00001

- **:values?**

  If t, the keyword will return a list of values - the initial return value for mouse button and the coordinates of the point selected by the user.

  Default Value: nil

select-point-on-surface [Function]

Arguments:

- **curve-geom**

  Geom ID of surface in question. Must be a single surface geom.

Keyword Arguments:

- **:nu**

  Integer specifying the number of candidate points to be generated in the u direction of the surface. The candidate points are equally spaced in the u param space.

  Default Value: 10

- **:nv**

Integer specifying the number of candidate points to be generated in the v direction of the surface. The candidate points are equally spaced in the v param space.

Default Value: 10

- **:tolerance**

  Distance within which the candidate points coordinates are located relative to the surface.

  Default Value: 0.00001

- **:values?**

  If t, the keyword will return a list of values - the initial return value for mouse button and the coordinates of the point selected by the user.

  Default Value: nil

undraw-geom [Function]

Undraws a geometry entity given its geom ID. Geom IDs are stored in the geom property of graphic objects or can be generated by direct calls to the virtual geometry layer creation functions (see Appendix D). Working at the level of geom IDs should only by done by advanced AML users.

Arguments:

- **geom-id**

Chapter 5. High Level Objects

Dimension Objects

The dimension objects are used to indicate component dimensions graphically in a manner similar to that typically found on a mechanical drawing. Objects are available to represent linear distance and angular displacement. Three of these objects will typically be used in models. The linear-dimension-object and angular-dimension-object are graphical objects which are displayed as part of the model. The ref-dim-propertiesobject is used to enforce a consistent style across sets of dimension objects. It's behavior is discussed in the section Reference Dimensions.

Common Features

Several components are common to both linear and angular dimension objects. The dimension line is the line (or arc) which indicates the length or angle of the displacement. It may have arrows on either end and contains the text of the dimension object. The extension lines are drawn from thepoints indicating the actual dimension to the dimension line. Dimension lines may be drawn either inside or outside of the extension lines. All dimension objects have a normal property which specifies the normal to the plane which contains the object. This normal and the points being dimensioned determine the location of the dimension object. The property scale-factor governs the appearance of several components of the dimension object. Most size properties (for components such as arrows, text, and extension lines) are multiplied by this value before drawing. This allows component sizes to be specified for their final output format independently of the model size and then scaled to fit the model.

Text

Numerous properties are available to control the appearance of text in a dimension object. Any dimension object has an associated numerical distance (or angle). Within the dimension object graphics, this value will be formatted using the string in the property text-format. Two additional properties, ext-prefix and text-suffix, are provided to place text before and after the numerical value. These three text components are combined into one string using the format contained in the property text-wrap-format. This formatstring has three positions which accept, in order: the text-prefix, the numerical value formatted with text-format, and the text-suffix. The string resulting from text-wrap-format is placed in the text property of the dimension object. The text displayed in the dimension may be overridden by putting any string in the text property.To display no text in the dimension, the property text should be given an empty string, "", and the value of the property text-gap set to 0. The behavior of these text properties is summarized in Table 1.

Figure 5-1. Dimension Object Text Properties

Table 1: Dimension Object Text Properties

distance	text-prefix	text-suffix	text-format	text-wrap-format	text	displayed in dimension
1.2345	""	""	"~a"	"~a~a~a"	"1.2345"	"1.2345"
1.2345	""	" m"	"~,2f"	"~a~a~a"	"1.23 m"	"1.23 m"
1.2345	""	" m"	"~,2f"	"~a~a~a"	"length"	"length"
1.2345	"R"	" in"	"~,1f"	"~a~a~a"	"R1.2 in"	"R1.2 in"
1.2345	""	" mm"	"~,1f"	"~a~a~a"	"(1.2 mm)"	"(1.2 mm)"

The text displayed in the dimension object is affected by the property dimension-scale-factor. Before being used by the text properties, the numerical value of the dimension is multiplied by dimension-scale-factor. This allows the dimensions to be displayed in units other than those in which the model was defined. For example, if a dimension spans a model distance of 10.0 and has a dimension-scale-factor of 25.4, then the value used by the text operations will be 254.0 (representing a conversion from inches to millimeters). Several properties are available to control the positioning of text in the dimension line. The property text-aligned? controls the orientation of the text. If its value is t, then the text will be aligned with the dimension line. If nil, then the text will be displayed in the direction specified by text-direction. Text will be displayed in the plane specified by the property text-normal (which defaults to the dimension object normal property). If the property text-view-normal? has a value of t, then text will always be displayed in the plane of the screen.

Figure 5-2. Dimension Objects: Text Properties

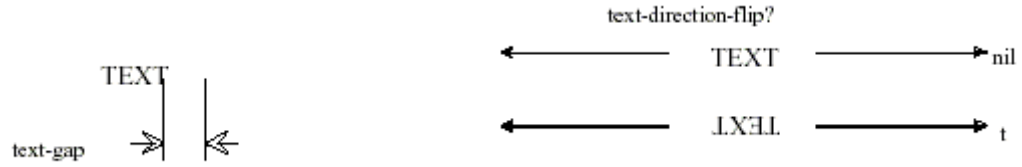


Figure 5-3. Dimension Objects: Text Alignment

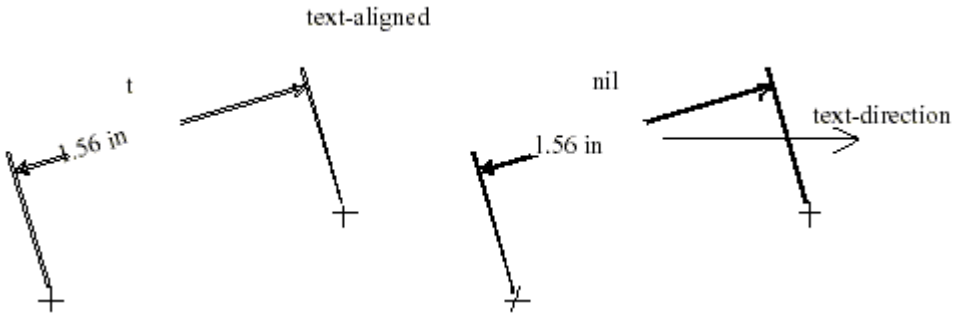
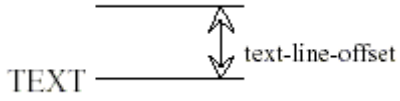


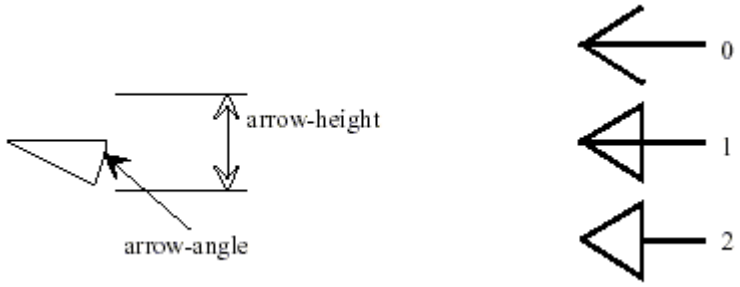
Figure 5-4. Dimension Objects: text-line-offset



Dimension Line

The dimension line is the line (or arc) drawn between the extension lines. It may be split to contain the text and may have arrows on one or both ends.The position of the text is specified with the property text-position. A value from 0 to 1 specifies a position ranging from one extension line to the other. The default value of 0.5 specifies the middle of the dimension line. A gap will be left in the line to fit the dimension text. Additional space will be left on either side of the text equal to the value in the property text-gap. The text may be moved above or below the dimension line by the value in the property text-line-offset. The dimension line can also be drawn outside of the extension lines. If the property dim-outside? has the value t, then the dimension line will be split and drawn from outside the extension lines. The text placement will still default to between the extension lines. Arrows may be drawn on either end of the dimension line. The properties of each arrow may be set independently of the other arrow. Unless otherwise specified, however, the properties of arrow2 will follow arrow1. Four properties govern the appearance of each arrow: arrowN?, arrow-typeN, arrow-heightN, arrow-angleN (where N is either 1 or 2). The following discussion will focus on arrow1. If the property arrow1? is t, then the arrow will be drawn; if nil it will not. The type of arrow (open,closed, etc.) is controlled by the property arrow-type1. The arrow base height and half-angle are specified by the properties arrow-height1 and arrow-angle1.

Figure 5-5. Dimension Objects: Arrows



Extension Lines

Several properties affect the appearance of the extension lines associated with each dimension object. The extension lines are drawn from the points used to specify the dimension toward the dimension line. A gap of length extn-offset is left between the dimension points and the extension lines. The dimension line is placed a minimum of extn-lower from the points. The actual distance depends on the dimension direction property. The extension lines will extend by the additional length extn-upper beyond the dimension line. In addition, two properties, part-extn1 and part-extn2, allow for an additional offset to be applied to each extension line. This behavior allows dimensions to be specified by convenient points which do not necessarily lie on part boundaries. The extension lines can be hidden by setting the properties extn1? and extn2? to nil. If it is not specifically set, extn2? will follow the behavior of extn1?.

Reference Dimension

The linear-dimension-object and angular-dimension-object (described below) inherit from the ref-dim-properties. This class adds the property reference-dimension to these classes. A consistent appearance throughout a group of dimensions may be maintained through this property and the associated behavior built into ref-dim-properties. If an object's reference-dimension is set to point to an object of ref-dim-properties, then many of the object's appearance-related properties will follow the values of the reference-dimension object. These appearance properties include extension line, arrow, and text properties. Any properties which should not follow the reference-dimension object properties may be overridden in the object. By having dimension objects refer to a reference-dimension, drawings may be kept consistent (e.g., all text will be the same size). The reference-dimension object can be an instance of any object which inherits from ref-dim-properties. By maintaining a set of ref-dim-properties objects, a consistent drawing may be made with different groups of dimension objects depending on different (but related) reference-dimension objects. Consider the following example:

```
(in-package :aml)

(define-class ref-dim-example
  :inherit-from (object)
  :subobjects (
               (ref-dim :class 'ref-dim-properties
                        )
               (dim1 :class 'linear-dimension-object
                     reference-dimension ^^ref-dim
                     point1 '(0.0 0.0 0.0)
                     point2 '(5.0 0.0 0.0)
                     )
               (dim2 :class 'linear-dimension-object
                     reference-dimension ^^ref-dim
                     point1 '(7.0 0.0 0.0)
                     point2 '(12.0 0.0 0.0)
                     )
               (dim3 :class 'linear-dimension-object
                     reference-dimension ^^ref-dim
                     point1 '(14.0 0.0 0.0)
                     point2 '(19.0 0.0 0.0)
                     extn-lower 2.0
                     )
               )
  )
```

The subobjects dim1 and dim2 will follow the behavior of ref-dim; the subobject dim3 will follow all behavior except for the extn-lower property (Figure 24). If the text-height of ref-dim is changed from its default value of 0.1 to 0.3, then the text-height of all three linear dimensions will be changed (Figure 25). If the extn-lower of ref-dim is changed from 1.0 to 3.0, only dim1 and dim2 will follow this behavior. Since dim3 has been given a specific value for extn-lower, it will not be affected by changes to ref-dim (Figure 26).

Figure 5-6. Ref-Dim-Properties Example: original objects

Figure 5-7. Ref-Dim-Properties Example: after changing reference dimension text-height

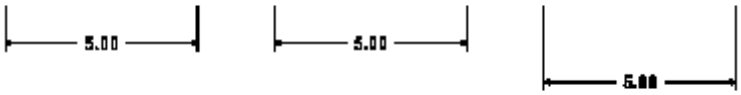
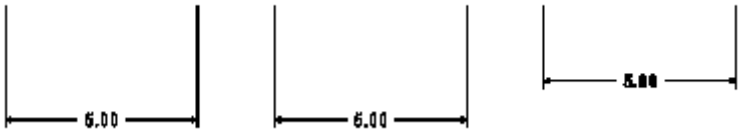


Figure 5-8. Ref-Dim-Properties Example: after changing reference dimension extn-lower



Classes and Methods

new-dimension-class [class]

This class contains basic parameters common to most dimension objects. This class should not be instantiated by the user.

Inheritance: object

Properties:

- **dimension-scale-factor**

  The actual value of the dimension is multiplied by this number before being displayed. This allows for such things as unit conversion (i.e., if the model is defined in inches but the dimensions should be shown in mm, a value of 25.4 can be given for dimension-scale-factor).

  Default Formula: 1.0

- **scale-factor (*)**

  The sizes of various components of the dimensions (text, arrows, extensions) are multiplied by this value. This allows for easy scaling of the dimension relative to the model size.

  Default Formula: 1.0

- **text**

  The text string which will be written in the dimension object. By default, this string is created using the properties text-wrapformat, text-format, text-prefix, and text-suffix. The default string is formed using the format text-wrap-format and contains the string text-prefix, a numerical value formatted

using text-format, and the string text-suffix. Placing any string value in this property will override this behavior.

- **text-prefix (*)**

  A prefix string displayed in the text component of the dimension object (see property text).

  Default Formula: ""

- **text-suffix (*)**

  A suffix string displayed in the text component of the dimension object (see property text).

  Default Formula: ""

- **text-format (*)**

  The string used to format the numerical value displayed in the text component of the dimension object (see property text).

  Default Formula: " -a"

- **text-wrap-format**

  A string used to format the entire text component of the dimension object (see property text).

  Default Formula: " -a -a -a"

- **text-font (*)**

  The font used for the text component of the dimension object.

  Default Formula: "stroked"

- **text-height (*)**

  The height of the text in the text component of the dimension object.

  Default Formula: 0.1

- **text-normal**

  The vector normal to the plane the text component of the dimension object is displayed in.

  Default Formula: dimension normal

- **text-direction**

  The vector indicating the direction in which text will be written for the text component of the dimension object.

  Default Formula: '(1 0 0)

- **text-gap (*)**

  The text component of the dimension object (default '(1 0 0)) text-gap (*) the additionalspace allowed in the dimension line for the string given by the property text.

  Default Formula: 0.05

- **text-direction-flip?**

  If t, negate the text-direction vector automatically calculated by the dimension object.

  Default Formula: nil

- **text-aligned?**

If t, text should be aligned with the dimension line, regardless of the value of text-direction.

Default Formula: t

- **text-view-normal?**

  If t, always view text normal to display.

  Default Formula: nil

- **text-line-offset**

  Distance to offset text normal to dimension line.

  Default Formula: 0.0

(*) These properties will follow the behavior of a reference dimension object, if applicable (see ref-dim-properties). They may be overwritten locally in any dimension object.

ref-dim-properties [class]

This class contains the structure and mechanisms required to allow dimension objects to share common property values. This allows the appearance of multiple dimension objects to follow that of a reference object. Each property (except for reference-dimension) uses the method dim-ref-property to establish this behavior and set default values. This class should not be instantiated by the user.

Inheritance: graphic-object

Properties:

- **reference-dimension**

  Instance of ref-dim-properties specifying the dimension object that this object will mimic. If nil, default values will be used.

  Default Formula: nil

- **value**

  This property is not used directly by any object. It exists to allow other objects to refer to its value. In this way, changing the value of the dimension can change the actual size of the object it refers to. This, in turn, will change the dimension object.

  Default Formula: nil

- **leader-extension-direction**

  A vector indicating the standard direction of leader extensions (used only by leader-dimension-object)

  Default Formula: '(1 0 0)

- **leader-extension-length**

  Allows a common length for all leader extensions (used only by leader-dimension-object)

  Default Formula: 1.0

The following properties are also affected by the properties of the reference-dimension object:

Properties:

- **extn-offset**

  Default Formula: (default 0.05)

- **extn-lower**

  Default Formula: (default 1.00)

- **extn-upper**

  Default Formula: (default 0.20)

- **text-font**

  Default Formula: (default "stroked")

- **text-height**

  Default Formula: (default 0.10)

- **text-gap**

  Default Formula: (default 0.05)

- **arrow-type1**

  Default Formula: (default 1)

- **arrow-height1**

  Default Formula: (default 0.05)

- **arrow-angle1**

  Default Formula: (default 20.0)

- **arrow-type2**

  Default Formula: (default !arrow-type1)

- **arrow-height2**

  Default Formula: (default !arrow-height1)

- **arrow-angle2**

  Default Formula: (default !arrow-angle1)

- **dimension-scale-factor**

  Default Formula: (default 1.0)

- **scale-factor**

  Default Formula: (default 1.0)

- **text-format (default "~,2f")**

  Default Formula: (default "~,2f")

- **text-prefix (default "")**

  Default Formula: (default "")

- **text-suffix (default "")**

  Default Formula: (default "")

- **color**

  Default Formula: (default "white")

dim-ref-property [Method]

When used as the formula in a property, returns the value of the same property in the object specified by reference-dimension, if it exists, or the value specified by default-value if no reference-dimension is specified. This method is only intended for use within the formula of properties in ref-dim-properties objects

Arguments:

- **default-value**

  The value to be returned if the object instance does not have a reference-dimension.

linear-dimension-object [class]

This class creates a simple linear dimension between two points.

Inheritance: points-dimension-class ref-dim-properties simple-geometry-class

Properties:

- **point1**

  The point specifying the first end of the length to be dimensioned.

  Default Formula: '(0 0 0)

- **point2**

  The point specifying the second end of the length to be dimensioned.

  Default Formula: '(0 1 0)

- **flip-dim-side**

  If t, will change the side of the line between point1 and point2 on which the dimension object will be drawn.

  Default Formula: nil

- **normal**

  The normal to the plane in which the dimension object will be drawn.

  Default Formula: If point1 and point2 specify a line roughly aligned with the z axis, then (default '(1 0 0)); otherwise (default '(0 0 1))

- **direction**

  A vector specifying the direction in which the dimension measurement will be made and along which the dimension line will be drawn. If nil, the direction will be along the vector specified by point1 and point2.

  Default Formula: nil

- **location-offset**

  A vector specifying an offset to apply to the position in which the dimension object is drawn.

  Default Formula: '(0 0 0)

- **connect-dim-pts?**

  If t, draw a line between point1 and point2 in addition to the rest of the dimension object components.

Default Formula: nil

- **show-dim-line?**

  If t, draw the dimension line (a line, with arrows, between the extension lines at point1 and point2).

  Default Formula: t

- **text**

  The text to be displayed in the dimension object. The default value measures and scales the distance between point1 and point2 and uses the appropriate formatting, prefix, and suffix information.

- **text-format**

  The format in which the numerical part of the text property is represented.

  Default Formula: "~,2f"

Example 5-1. Linear Dimension object example

```
(define-class linear-dimension-example
  :inherit-from (object)
  :subobjects (
               (polygon :class 'polygon-object
                        vertices '((0 0 0) (8 0 0) (8 4 0) (6 4 0) (0 2 0))
                        )
               (ref-dimension :class 'ref-dim-properties
                              arrow-height1 0.2
                              text-height 0.2
                              arrow-type1 0
                              extn-offset 0.1
                              text-suffix "m"
                              )
               (width-dimension :class 'linear-dimension-object
                                point1 (vertex-of-object ^^polygon 0)
                                point2 (vertex-of-object ^^polygon 1)
                                reference-dimension ^^ref-dimension
                                )
               (height-dimension :class 'linear-dimension-object
                                 point1 (vertex-of-object ^^polygon 1)
                                 point2 (vertex-of-object ^^polygon 2)
                                 reference-dimension ^^ref-dimension
                                 )
               (cut-dimension :class 'linear-dimension-object
                              point1 (vertex-of-object ^^polygon 3)
                              point2 (vertex-of-object ^^polygon 4)
                              reference-dimension ^^ref-dimension
                              flip-dim-side t
                              text-direction-flip? t
                              )
               (horiz-cut-dimension :class 'linear-dimension-object
                                    point1 (vertex-of-object ^^polygon 3)
                                    point2 (vertex-of-object ^^polygon 4)
                                    reference-dimension ^^ref-dimension
                                    flip-dim-side t
                                    text-direction-flip? t
```

```
                                        direction '(1 0 0)
                                        )
                    )
  )
```

Figure 5-9. Linear-Dimension-Object: linear-dimension-example

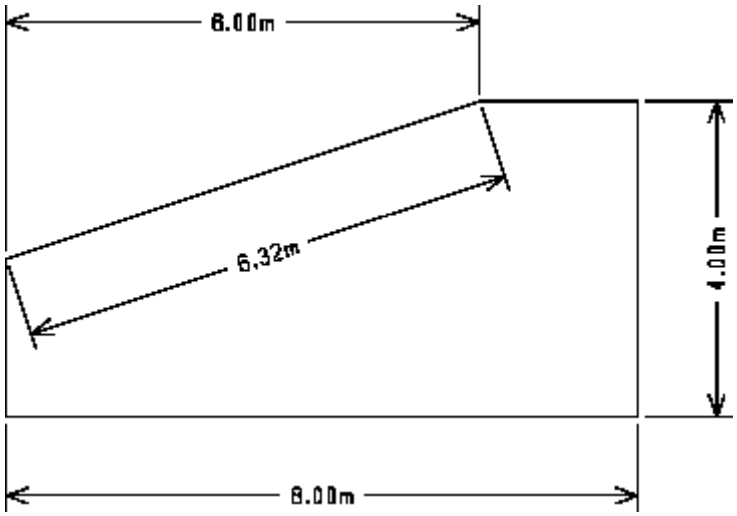


Figure 5-10. Linear-Dimension-Object: General

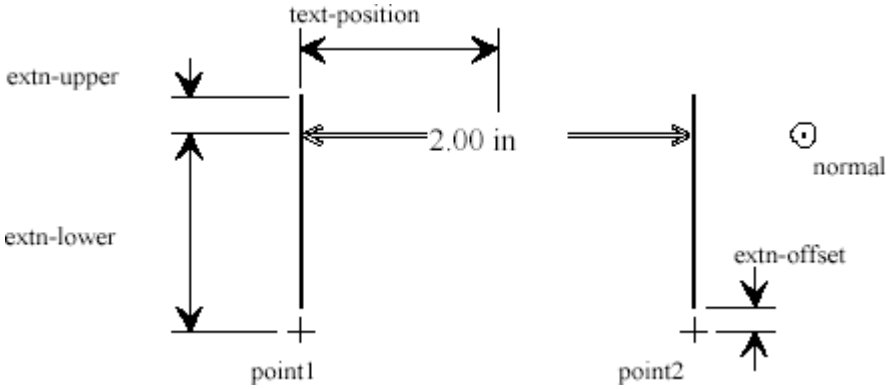


Figure 5-11. Linear-Dimension-Object: Extension Lines

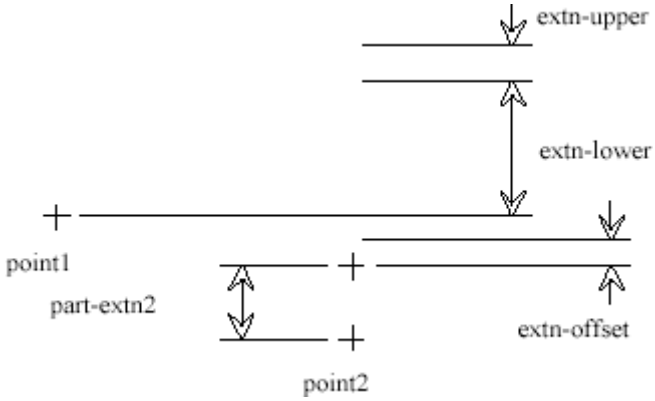


Figure 5-12. Linear-Dimension-Object: Dimension Lines

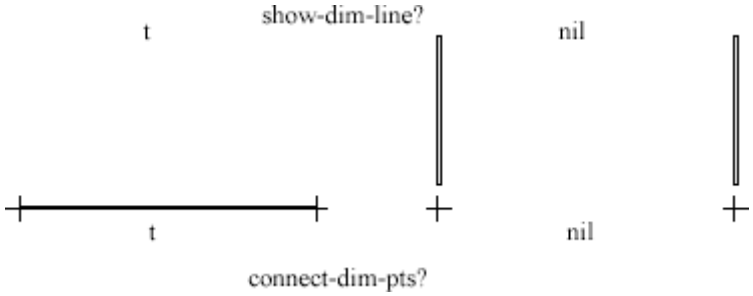


Figure 5-13. Linear-Dimension-Object: Outside Dimension

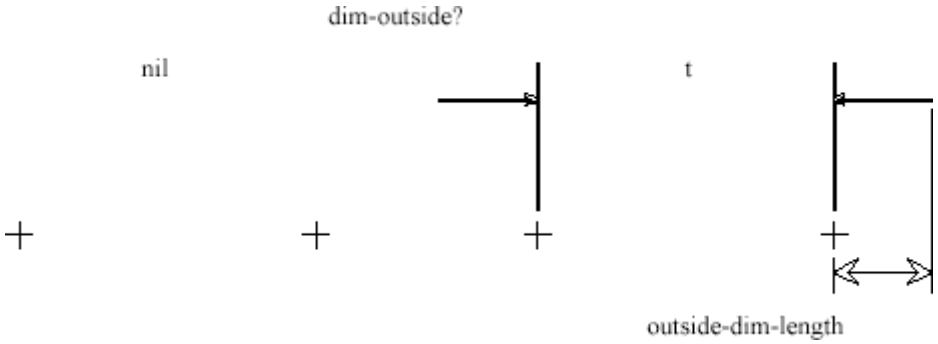


Figure 5-14. Linear-Dimension-Object: flip-dim-side

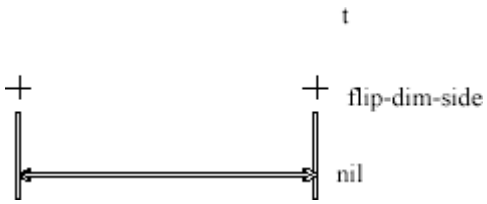
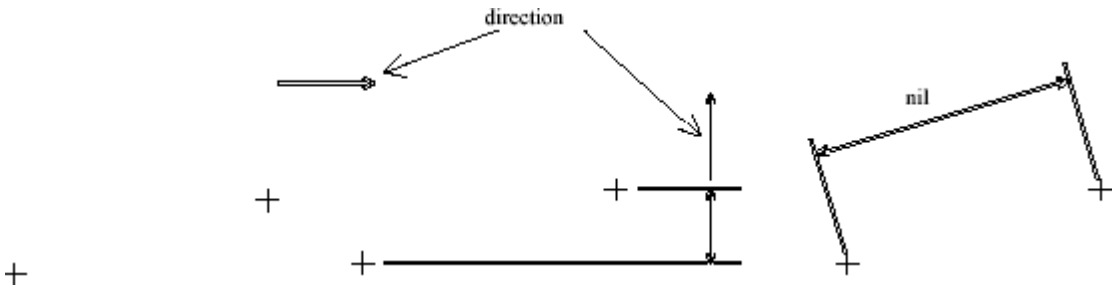


Figure 5-15. Linear-Dimension-Object: direction



angular-dimension-object [class]

This class creates a simple angular dimension based on a center point and two additional points indicating the extent of the angle.

Inheritance: ref-dim-properties points-dimension-class simple-geometry-class

Properties:

- **center**

  A vector specifying the center point of the angle being dimensioned.

  Default Formula: '(0 0 0)

- **point1**

  A vector specifying a point on one of the rays from the center indicating the extent of the angle.

  Default Formula: '(1 0 0)

- **point2**

  A vector specifying a point on one of the rays from the center indicating the extent of the angle.

  Default Formula: '(0 1 0)

- **normal**

  A vector specifying the normal to the plane of the dimension object.

  Default Formula: The normal of the plane containing center, point1, and point2

- **dim-arc?**

  If t, draw the dimension line between the two extension lines.

  Default Formula: t

- **dim-inside?**

  If t, draw the dimension object on the inside (toward the center) of point1 and point2. If nil, draw on outside (away from center).

  Default Formula: nil

- **ctr?**

  If t, draw a simple crosshair object at the point specified by center.

  Default Formula: nil

- **ctr-size**

  The radius of the crosshair object drawn at center, if necessary.

  Default Formula: 0.1

- **ctr-ref-vec**

  A vector specifying the orientation of the center crosshair, if necessary. If nil, the crosshair will align with the vector from center to point1.

  Default Formula: nil

- **reflect-pt1?**

  If t, use the reflection of point1 through the center (instead of point1) when determining the extents of the angle and when drawing the dimension object.

  Default Formula: nil

- **relfect-pt2?**

  If t, use the reflection of point2 through the center (instead of point1) when determining the extents of the angle and when drawing the dimension object.

  Default Formula: nil

- **complement-anlge?**

  Normally, the angle dimensioned is the smaller of the two angles specified by center, point1, and point2. If complement-angle? is t, the larger angle is chosen.

  Default Formula: nil

- **ext1-to-ctr?**

  If t, extend the extension line at point1 to center, rather than stopping at point1.

  Default Formula: nil

- **ext2-to-ctr?**

  If t, extend the extension line at point2 to center, rather than stopping at point1.

  Default Formula: nil

- **text**

  The text to be displayed in the dimension. By default, it uses the standard format of prefix, value, and suffix.

- **text-format**

  The format in which numerical values in the dimension text should be presented.

  Default Formula: "~,2f"

- **angle**

  Contains the value of the angle being dimensioned and is not normally changed by the user.

- **text-auto-pos?**

  Controls automatically the position of the text string according to the value of the angle.

  Default Formula: nil

points-dimension-class [class]

This class contains basic parameters common to dimension objects based on distances characterized by two points.

Inheritance: new-dimension-class

Properties:

- **extn-offset (*)**

  The size of the gap between the extension lines of the dimension object and the points being dimensioned.

  Default Formula: 0.05

- **extn-lower (*)**

  The minimum length of the extension line from the dimension line to the points being dimensioned.

  Default Formula: 1

- **extn-upper (*)**

  The length which the extension line extends beyond the dimension line.

  Default Formula: 0.20

- **part-extn1**

  An additional gap added between the extension line of the dimension object and the first point being dimensioned.

  Default Formula: 0.0

- **part-extn2**

  An additional gap added between the extension line of the dimension object and the second point being dimensioned

  Default Formula: 0.0

- **normal**

  The vector specifying the normal to the plane the dimension will be drawn in

  Default Formula: '(0 0 1)

- **text-normal**

  The vector normal to the plane the text component of the dimension object will be drawn in (default !normal). This overrides the new-dimension-class property.

  Default Formula: !normal

- **text-position**

  Specifies the position of the text within the dimension line (from 0 to 1).

  Default Formula: 0.5

- **text-auto-pos?**

  If t, automatically position the text in the dimension line.

  Default Formula: nil

- **extn1?**

  If t, draw extension line at first point.

  Default Formula: t

- **extn2?**

  If t, draw extension line at second point.

  Default Formula: t

- **dim-outside?**

  If t, draw dimension lines outside of extension lines.

  Default Formula: nil

- **outside-dim-length**

  If dim-outside? is t, lines with this length are drawn as dimension line.

  Default Formula: 0.1

- **arrow1?**

  If t, draw arrow at first end of dimension line.

  Default Formula: t

- **arrow-type1 (*)**

  Specifies type of arrow to draw at first end of dimension line.

  Default Formula: 1

- **arrow-height1 (*)**

  Specifies height of arrow at first end of dimension line.

  Default Formula: 0.1

- **arrow-angle1 (*)**

  Specifies angle of arrow at first end of dimension line.

  Default Formula: 15.0

arrow2?, arrow-type2 (*), arrow-height2 (*), arrow-angle2 (*) are similar to properties for arrow head 1 (defaults follow arrow head 1 properties)

(*) These properties will follow the behavior of a reference dimension object, if applicable (see ref-dim-properties). They may be overwritten locally in any dimension object.

Graphing

The datagraph-object can be used to produce x-y (scattergram) plots of numerical data within a model.

Classes

datagraph-object [class]

The datagraph-object can be used to produce simple x-y (scattergram) data plots. The user has control over the appearance of the data, the graph axes, labelling, etc.

Inheritance: group-object

Properties:

- **location**

  The position of the origin (of the x and y axes) of the plot (or of the lower left corner of the plot border if show-border is t).

  Default Formula: '(0 0 0)

- **x-direction**

  A vector indicating the direction along which the x-axis should lie.

  Default Formula: '(1 0 0)

- **normal**

  The normal to the plane in which the graph should lie.

  Default Formula: '(0.0 0.0 1.0)

- **color**

  The default color for all objects in the graph.

- **title**

  Text which will be displayed at the top of the graph (default is no text).

- **title-size**

  The size of the title text.

  Default Formula: 0.25

- **title-offset**

  The amount of space between the top of the graph and the bottom of the title text.

  Default Formula: title-size

- **title-font**

  The font used for the title text (default "stroked"). If the property text-view-normal? is not t, all fonts will appear as "stroked".

- **title-color**

  The color in which the title should be displayed.

  Default Formula: ^color

- **x-limit-lower**

  The lower limit of the x-axis. If nil, the limit will be determined automatically to fit the data.

Default Formula: nil

- **x-limit-upper**

  The upper limit of the x-axis. If nil, the limit will be determined automatically to fit the data.

  Default Formula: nil

- **x-length**

  The length of the x-axis.

  Default Formula: 4.0

- **x-major-tick-space**

  The increment between labelled tick marks on the x-axis. If nil, the increment will be determined automatically.

  Default Formula: nil

- **x-minor-ticks**

  The number of minor increments (smaller tick marks) between major tick marks. A value of 1 results in one division and no tick marks, a value of 2 results in one minor tick mark, etc.

  Default Formula: 2

- **x-major-length**

  The length of the major tick marks.

  Default Formula: 0.05

- **x-minor-length**

  The length of the minor tick marks.

  Default Formula: 0.025

- **x-show-grid**

  If t, a line will be drawn across the graph at each major tick mark on the x-axis.

- **x-label**

  The label displayed along the x-axis.

  Default Formula: ""

- **x-label-size**

  The text size of the x-axis label.

  Default Formula: 0.10

- **x-label-font**

  The font used to display the x-axis label.

  Default Formula: "stroked"

- **x-label-offset**

  The distance between the x-axis and the center of the x-axis label.

  Default Formula: 0.30

- **x-number-format**

  The format used to display the labels at the major tick marks along the x-axis.

  Default Formula: "~f"

- **x-number-font**

  The font used to display major tick mark labels on the x-axis.

  Default Formula: "stroked"

- **x-number-size**

  The text size of the x-axis major tick mark labels.

  Default Formula: 0.10

- **x-number-offset**

  The distance between the end of the major tick marks and the nearest part of the tick mark labels.

  Default Formula: (/ !x-label-size 10.0)

- **x-number-scale**

  The numbers displayed for the tick mark labels are divided by this value.

  Default Formula: 1.0

- **x-auto-scale**

  If t, the numbers displayed for the tick mark labels are automatically divided by 10^n (where n is a power of 3). x-number-scale takes precedence over this parameter.

  Default Formula: nil

- **x-axis-color**

  The color for the x-axis.

  Default Formula: !color

- **x-label-color**

  The color used for the x-axis label.

  Default Formula: !x-axis-color

- **x-number-color**

  The color used for the x-axis tick mark labels.

  Default Formula: !x-axis-color

- **x-grid-color**

  The color used for the x-axis grid lines.

  Default Formula: !x-axis-color

- **y-limit-lower through y-grid-color are the equivalent to the above properties for the y-axis.**

- **show-border**

  If t, a rectangle is drawn around the entire graph.

  Default Formula: nil

- **border-offset**

  A list of two values specifying the horizontal (x) and vertical (y) margins between the graph proper and the border rectangle.

  Default Formula: '(0.5 0.5)

- **close-graph-box**

  If t, completes the rectangle formed by the x and y axes.

  Default Formula: nil

- **data**

  A list (or list of lists) of x-y paired data (i.e. '(((x1_1 y1_1) (x2_1 y2_1) ...) ((x1_2 y1_2) (x2_2 y2_2) ...) ...)).

  Default Formula: ()

- **line-color**

  A list of color values used to draw the lines connecting data points. Each position in the list corresponds to the equivalent list of x-y points in the data property. If fewer colors than the number of curves specified in data are given, the curves will cycle through the available colors.

  Default Formula: (list !color)

- **line-types**

  A list of the line types used to draw the lines connecting data points. The behavior of this list is similar to that of line-color.

  Default Formula: (list 0)

- **line-widths**

  A list of the line widths used to draw the lines connecting data points. The behavior of this list is similar to that of line-color.

  Default Formula: (list 1)

- **symbol-type**

  A list of the symbol types to be displayed at each point along the data curves (0 - none, 1 - +, 2 - o, 3 - square, 4 - diamond). The behavior of this list is similar to that of line-color.

  Default Formula: 1

- **symbol-size**

  A list of the sizes of the symbols displayed at each point along the data curves. The behavior of this list is similar to that of line-color.

  Default Formula: (list 0.1)

- **symbol-color**

  A list of the colors of the symbols displayed at each point along the data curves. The behavior of this list is similar to that of line-color.

  Default Formula: (list !color)

- **show-line**

A list of values, one for each list in data, for which, if t, lines will be drawn connecting the data points. If nil, no lines will be drawn.

Default Formula: (list nil)

- **drop-line-to-x**

  A list of values, one for each list in data, for which, if t, lines will be drawn from the data point to the x-axis.

  Default Formula: (list nil)

- **drop-line-to-y**

  A list of values, one for each list in data, for which, if t, lines will be drawn from the data point to the y-axis.

  Default Formula: (list nil)

- **drop-line-color**

  A list specifying the colors that drop lines will be drawn in (drop lines to the x- and y-axes will be the same color).

  Default Formula: (list !color)

- **show-legend**

  When t, a legend will be displayed showing the symbol, line type, color, and name for each series of points in data. When nil, no legend is displayed.

  Default Formula: nil

- **legend-location**

  The position of the symbol of the first series in the legend.

  Default Formula: '(4.5 3.0)

- **series-name**

  A list of strings to be displayed labelling the series in the legend.

  Default Formula: (list "series")

- **legend-text-size**

  The size of the text used to display series information in the legend.

  Default Formula: 0.10

- **legend-text-font**

  The font of the text used to display series information in the legend.

  Default Formula: "stroked"

- **legend-text-color**

  The color of the text used to display series information in the legend.

  Default Formula: !color

- **point-labels**

A list of coordinates and corresponding labels to be placed on the graph. The labels are placed at the specified coordinates with a given alignment and offset. The format is a list with elements (list "string" x-coord y-coord) where the x-coord and y-coord are specified in the same frame as the plot data.

Default Formula: ()

- **point-labels-font**

  The font in which the point-labels should be displayed.

  Default Formula: "stroked"

- **point-labels-size**

  The size in which the point-labels should be displayed.

  Default Formula: 0.1

- **point-labels-color**

  The color in which the point-labels should be displayed.

  Default Formula: !color

- **point-labels-alignment**

  The alignment of the point-labels relative to their specified coordinates.

  Default Formula: '(:center :center)

- **point-labels-offset**

  An offset added to the point-labels coordinates before the pointlabels- alignment takes effect.

  Default Formula: '(0.0 0.0)

- **point-labels-direction**

  The direction in which the point-labels should read.

  Default Formula: !x-direction

Example 5-2. Datagraph object example

```
(define-class datagraph-example
  :inherit-from (datagraph-object)
  :properties (
               data '(((0.0 0.0) (1.0 2.0) (2.0 1.0) (3.0 3.0) (4.0 2.0))
                      ((0.0 1.0) (1.0 1.5) (2.0 2.0) (3.0 1.5) (4.0 3.0))
                      ((0.0 0.5) (1.0 1.0) (2.0 0.5) (3.0 1.0) (4.0 1.0))
                      )
               symbol-size '(0.05) ;this will be used for all 3 data series
               symbol-type '(2 3 4)
               line-color '(green)
               title "An Example Graph"
               title-size 0.15
               title-offset 0.25
               x-label "Distance (m)"
               x-minor-ticks 4
               y-label "Load (N)"
               y-limit-upper 4.0
```

```
                show-legend t
                series-name '("Data 1" "Data 2" "Data 3")
                )
)
```

Figure 5-16. Datagraph-Object

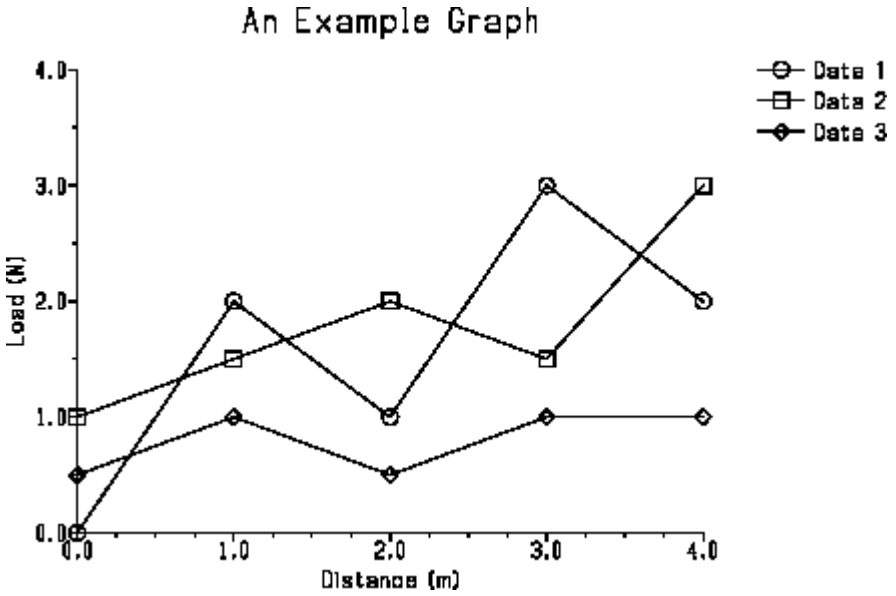


Figure 5-17. Datagraph-Object: location, show-border

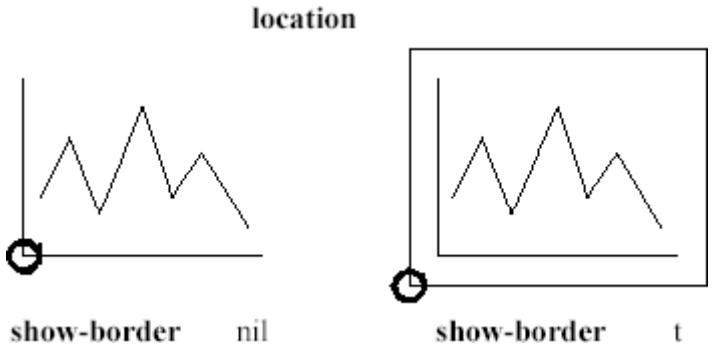


Figure 5-18. Datagraph-Object: normal, x-direction

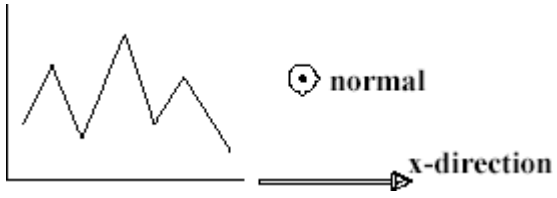


Figure 5-19. Datagraph-Object: title

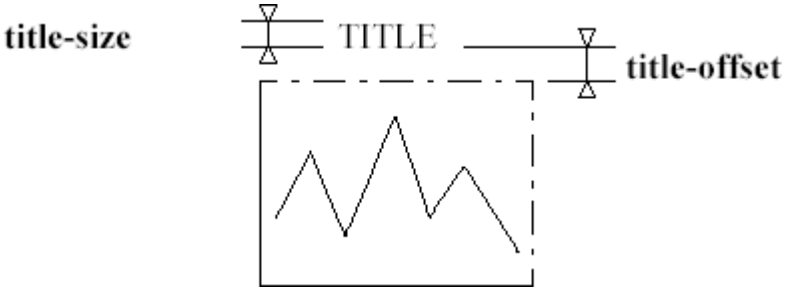


Figure 5-20. Datagraph-Object: axis properties

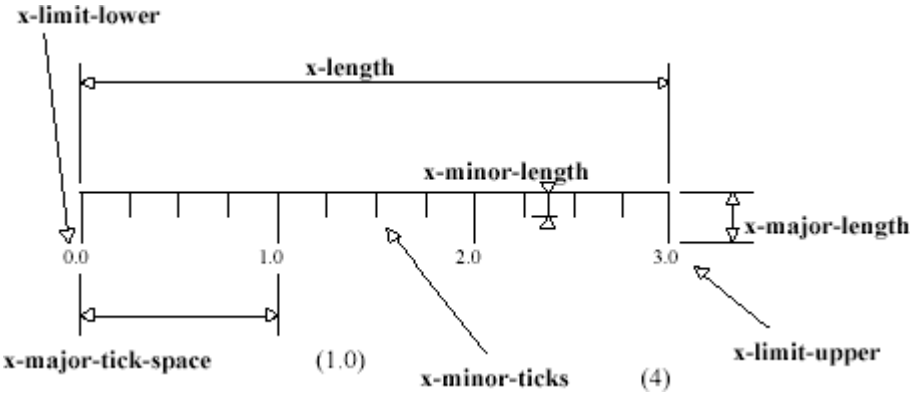


Figure 5-21. Datagraph-Object: axis numbers and labels

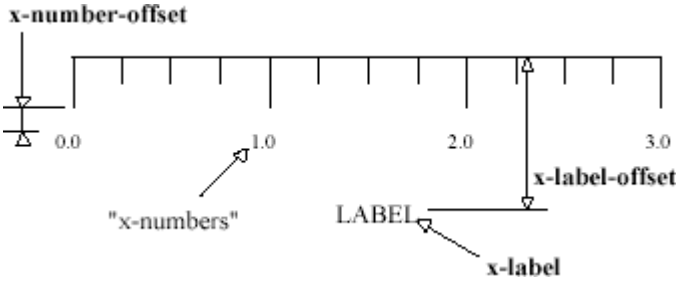


Figure 5-22. Datagraph-Object: show-grid

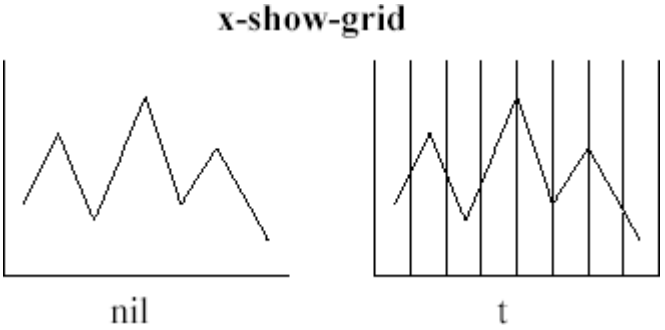


Figure 5-23. Datagraph-Object: show-border

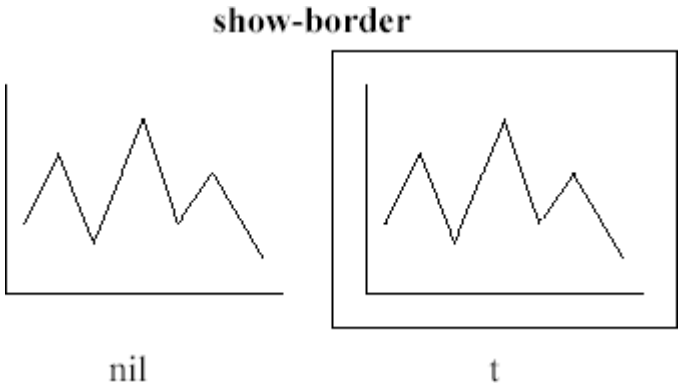


Figure 5-24. Datagraph-Object: close-graph-box

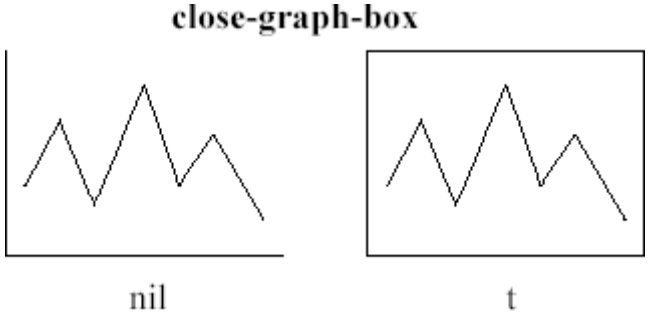


Figure 5-25. Datagraph-Object: show-line

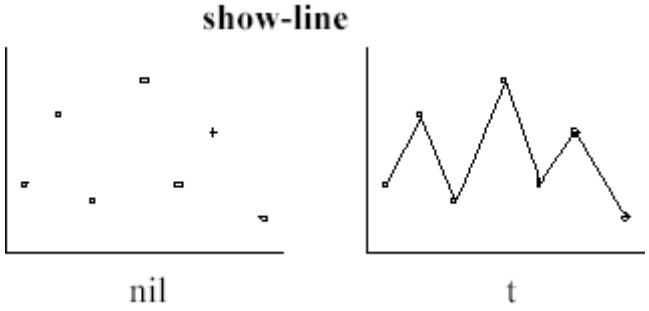


Figure 5-26. Datagraph-Object: drop-line

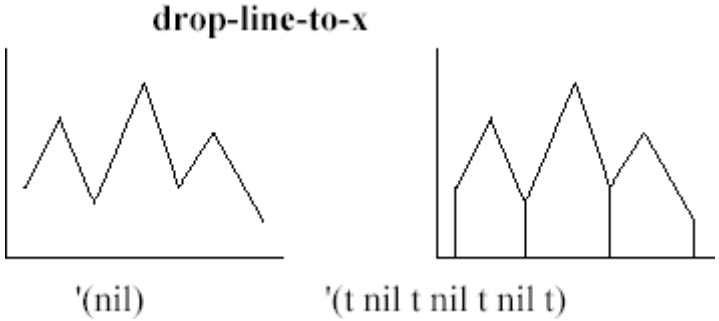


Figure 5-27. Datagraph-Object: legend

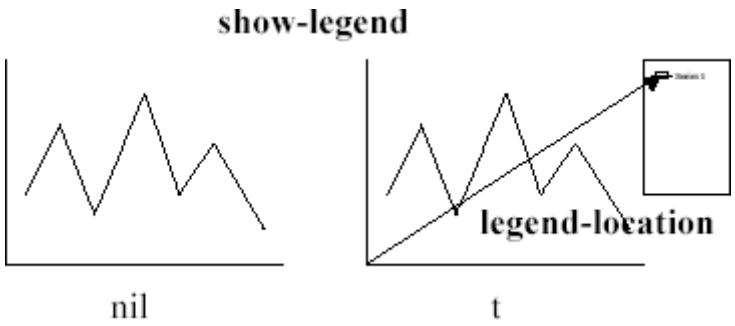
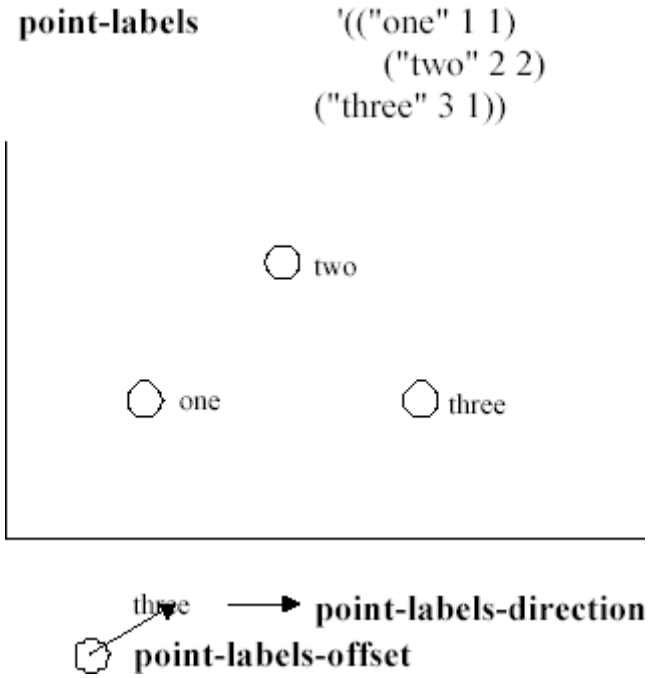


Figure 5-28. Datagraph-Object: point-labels



Drawing Enhancements

The plot-frame and plot-info-box objects can be used to enhance output drawings or create tabular reports within drawings.

Classes and Methods

plot-frame [class]

This object creates a labelled border which can be used to frame a drawing in a manner typical to mechanical drawings. All sizes specified in the object are multiplied by the property scalefactor when the frame is drawn. This allows the size of the frame, text size, and other dimensions to be sized appropriately to fit the final output (e.g., print-out) independent of the model size.

Inheritance: assembly-object

Properties:

- **location**

  The position of lower-left corner of frame if center-frame? is nil or of center of frame if center-frame? is t

  Default Formula: '(0 0 0)

- **height**

  The height of frame

  Default Formula: 8.5

- **width**

  The width of frame (along width-direction)

  Default Formula: 11.0

- **scale-factor**

  Used to scale the other dimensioned properties

  Default Formula: 1.0

- **width-direction**

  Vector defining orientation of the frame width

  Default Formula: '(1 0 0)

- **normal**

  Normal to the plane of the frame

  Default Formula: '(0 0 1)

- **border-width**

  Width of the frame border. If border-width is 0 or nil, no border will be drawn and the plot-frame will consist of a single, rectangular border.

  Default Formula: 0.25

- **partition-line-length**

  The length of the lines separating regions in the border

  Default Formula: 0.20

- **text-height**

  Height of text used to label borders

- **text-font**

  Font used to label borders

- **h-label-type**

  Type of labels to use on vertical borders (height direction) (0 - none, 1 - alphabetic, 2 - numeric)

  Default Formula: 2

- **w-label-type**

Type of labels to use on horizontal borders (width direction) (0 - none, 1 - alphabetic, 2 - numeric)

Default Formula: 1

- **reverse-h-text**

  Describes variation of labels in vertical borders, Boolean property (nil - labels increase from bottom to top, t - labels decrease from bottom to top)

  Default Formula: nil

- **reverse-w-text**

  Describes variation of labels in horizontal borders, Boolean property (nil - labels increase from left to right, t - labels decrease from left to right)

  Default Formula: nil

- **center-frame?**

  If t, the frame is centered about the position given by location. If nil, the frame is located with the its lower left corner at location

- **corner-LL, corner-LR, corner-UR, corner-UL**

  coordinates of inside corners of frame (useful for placing plot-info-box's) (LL - lower left, LR - lower right, UR - upper-right, UL - upper-left)

Example 5-3. Plot frame example

```
(define-class plot-frame-example
  :inherit-from (object)
  :subobjects (
               (box :class 'box-object
                    height 2.0
                    width 2.0
                    depth 2.0
                    orientation (list (translate '(5.0 5.0 0.0)))
                    )
               (frame :class 'plot-frame
                      location (center-of-object (the box))
                      scale-factor 0.5
                      center-frame? t
                      )
               )
  )
```

Figure 5-29. Plot-Frame

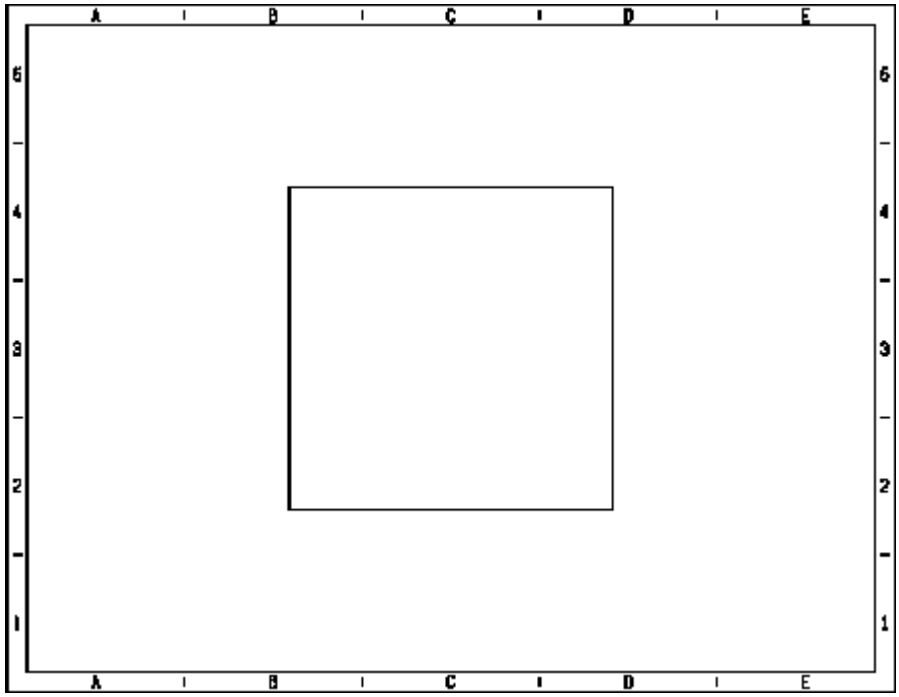


Figure 5-30. Plot-Frame: vertex-of-object indices

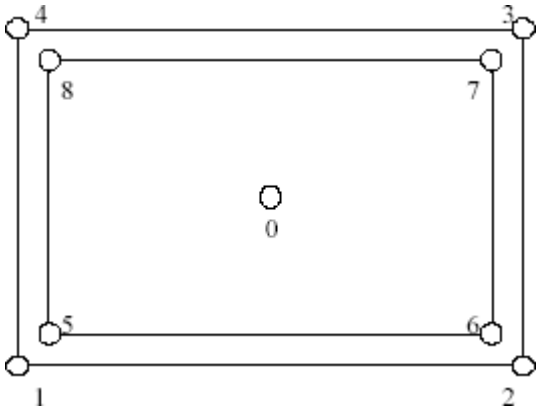
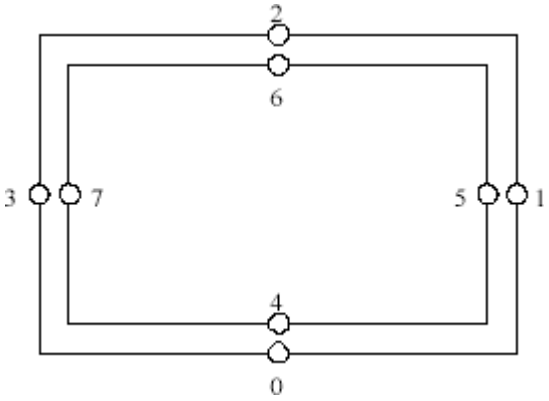


Figure 5-31. Plot-Frame: center-of-edge indices



plot-info-box [class]

This object creates a partitioned box containing text. All sizes in the object are multiplied by the property scale-factor when the box is drawn. This allows the size of the box and text within the box to be sized appropriately for the final output form (e.g., print-out) independent of the model size.

Inheritance: assembly-object

Properties:

- **box-info**

  A list in which each element is a list describing each row in the box. The first element in each list specifies the relative height of that row in the box. The heights for all rows are added together and individual row heights are used to determine the percentage of the overall box height occupied by each row. Therefore, two rows with equal height could be specified as 1 and 1 or as 50 and 50. Two rows, the first twice as high as the second, could be specified as 2 and 1. After the height entry, each list contains lists of information specifying the size and contents of each column in the row. The first entry in each of the column lists is a width specifier which works in a manner similar to the row height value. The list format is: (list column-width text-string text-height text-alignment textfont). Not all elements in the list need to be specified, but no elements can be skipped (i.e., if you want to specify text-alignment, everything to the left of text-alignment must also be specified but text-font does not). If the last entry in a row list is :hl, then no horizontal line will be drawn beneath that row. If the las entry in a column list (within a row list) is :vl, then no vertical line will be drawn after that column (except for the last column, which is always drawn).

  Default Formula:

```
(list (list 1 '(100 "Text" 0.20 :center "stroked"))
(list 1 '(100 "Text" 0.20 :center "stroked"))
(list 1 '(50 "Text" 0.10 :center "stroked")
'(50 "Text" 0.10 :center "stroked")))
```

- **border?**

  When non-nil, borders (external and internal) of the info box will draw when the box is drawn. The borders will not draw otherwise.

- **height**

  Height of box.

  Default Formula: 1.0

- **width**

  Width of box.

  Default Formula: 3.0

- **width-direction**

  Vector defining orientation of the box width.

  Default Formula: '(1 0 0)

- **normal**

  Normal to the plane of the box.

  Default Formula: '(0 0 1)

- **scale-factor**

  Used to scale the other dimensioned properties.

  Default Formula: 1

- **attachment-point**

  Location at which to anchor the box.

  Default Formula: '(0 0 0)

- **attachment-corner**

  Defines the part of the box to place at the attachment-point (0 - center, 1 - lower left, 2 - lower right, 3 - upper right, 4 - upper-left)

  Default Formula: 1

- **text-font**

  Font to be used if not specified in box-info.

  Default Formula: "stroked"

- **text-alignment**

  Alignment to be used for text items if not specified in box-info.

  Default Formula: :center

- **text-size**

  Size of text to be used if not specific in box-info.

  Default Formula: 0.10

- **text-margin**

  Spacing between borders of box partitions and text when text is aligned :left or :right.

  Default Formula: (half !text-size)

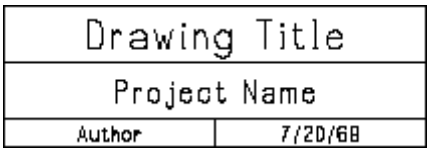
- **corner-LL, corner-LR, corner-UR, corner-UL**

Coordinates of corners of info box.

Example 5-4. Plot info box example 1

```
(define-class plot-info-box-example-1
  :inherit-from (plot-info-box)
  :properties (
               box-info (list (list 1 '(100 "Drawing Title" 0.20 :center))
                              (list 1 '(100 "Project Name" 0.15 :center))
                              (list 0.5 '( 50 "Author" 0.10 :center)
                                    '( 50 "7/20/69" 0.10 :center)
                                    )
                              )
               height 1.0
               width 3.0
               )
  )
```

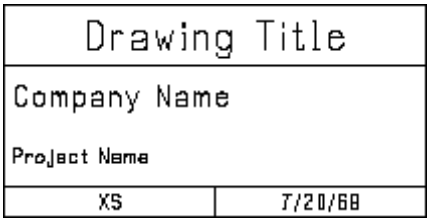
Figure 5-32. Plot-info-box-example1



Example 5-5. Plot info box example 2

```
(define-class plot-info-box-example-2
  :inherit-from (plot-info-box)
  :properties (
               box-info (list (list 1 '(100 "Drawing Title" 0.20 :center))
                              (list 1 '(100 "Company Name" 0.15 :left) :hl)
                              (list 1 '(100 "Project Name" 0.10 :left))
                              (list 0.5 '( 50 "XS" 0.10 :center)
                                    '( 50 "7/20/69" 0.10 :center)
                                    )
                              )
               height 1.5
               width 3.0
               )
  )
```

Figure 5-33. Plot-info-box-example2



Example 5-6. Plot info box example 3

```
(define-class plot-info-box-example-3
```

```
  :inherit-from (object)
  :subobjects (
               (frame :class 'plot-frame)
               (info :class 'plot-info-box
                     box-info (list (list 1 '(100 "Drawing Title" 0.20
:center))
                                    (list 1 '(100 "Project Name" 0.15
:center))
                                    (list 0.5 '( 50 "Author" 0.10 :center)
                                          ( 50 "7/20/69" 0.10 :center)
                                          )
                                    )
                     height 1.0
                     width 3.0
                     attachment-corner 2
                     attachment-point (the frame corner-LR)
                     )
               )
  )
```

Figure 5-34. Plot-info-box-example3

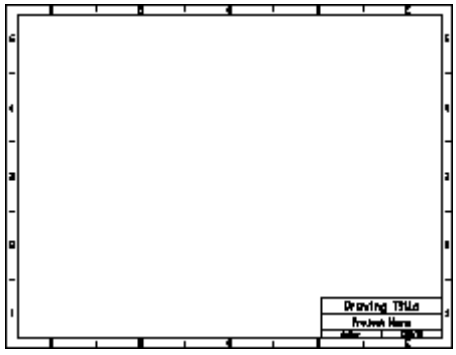


Figure 5-35. Plot-Info-Box: vertex-of-object indices

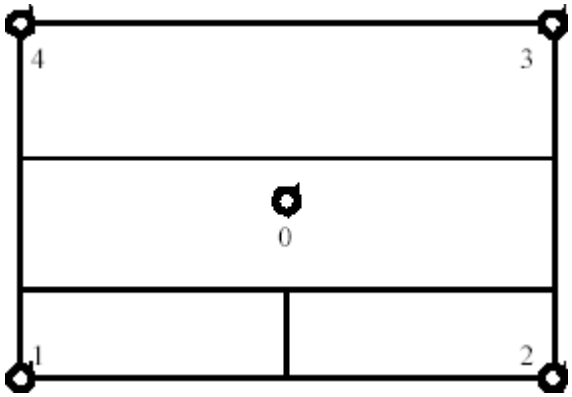
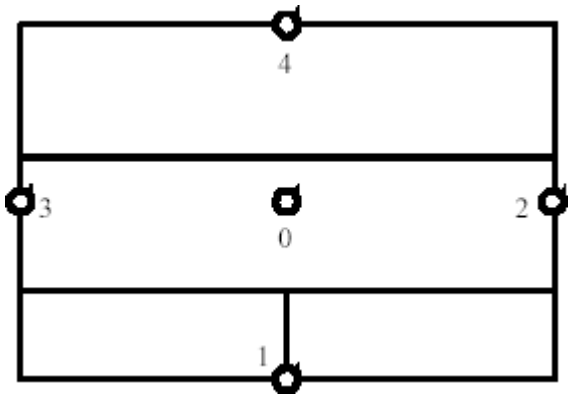


Figure 5-36. Plot-Info-Box: center-of-edge indices



get-info-box-cell-info [Method]

This method returns the coordinates of the specified vertex for the cell at the given row and col position of the info box. In addition, it returns the dimensions of that partition. The results are returned in a list:

```
(coordinates width height scale-factor)
```

The coordinates are returned as a list, (x y z). The values returned for width and height are those specified in the object properties. To determine the true width and height they must be multiplied by the scale factor.

Arguments:

- **instance**

  An instance of type plot-info-box

- **row**

  The row number of the cell to be queried. The first row is 0.

- **col**

  The column number within the specified row of the cell to be queried. The first column in each row is 0. The number of the last column will vary from row to row.

- **corner**

  The index of the desired cell vertex.

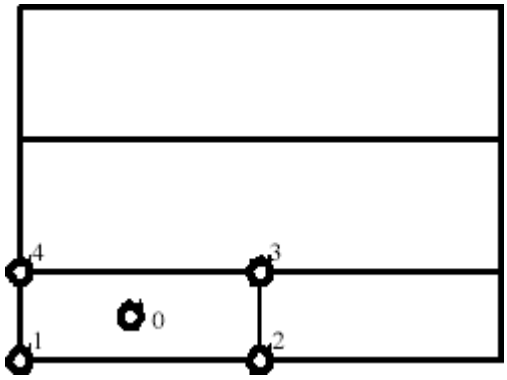
Keyword Arguments:

- **:global?**

  If t, the coordinates will be returned with respect to the global coordinate frame. The default returns the coordinates in the local frame.

  Default Value: nil

Figure 5-37. Plot-Info-Box: Get-Info-Box-Cell-Info: vertex indices



Chapter 6. Orientation

Introduction

All objects which inherit from position-object can be oriented in model space. An object's orientation can consist of any combination of translations and rotations. This orientation may be "built-in" to the object through its class definition or applied to the object after instantiation (creation). The position-object class has a property called orientation. This property can contain a list of orientation functions which describe the object's transformation from its initial position (where it is created) to its final position. The orientation property and the functions it contains are discussed in the section Orientation Functions. A group of methods is available to change the orientation of objects at any time after their creation. These methods add instructions to the list in the object's orientation property. Most of these methods are written on the class position-object. Some methods, which depend on graphic aspects of the object, are written on the class graphic-object. All of these methods are described in the section Transformation Methods.

Coordinate Frames And Reference Coordinate Systems

Each object which inherits from position-object has three coordinate frames associated with it: global, local, and reference. The global frame is the model (or world) coordinate system. This frame remains fixed and all objects are oriented within it (although the orientation operations may be defined in a different coordinate frame). The local frame is the coordinate system in which an object is defined. Before orientation, this system is aligned with the global system. The local coordinate frame follows the object through any orientation operations performed on it. All position objects are defined with respect to their own local coordinate frame. Object are oriented with respect to their reference coordinate frame. By default, the reference coordinate frame of an object is the global coordinate frame. Subobjects do not automatically follow the orientation of their parent object. The orientation behavior may be modified by changing an object's reference coordinate frame. Every object that inherits from position-object has the property reference-coordinate-system. This property can contain a the reference to an object of class coordinate-system-class, which will provide a reference coordinate frame for the object. By default, this property has a value of nil and all orientation operations performed on the object will be with respect to the global coordinate frame. If, instead, the reference-coordinate-system points to a coordinate-system-class object, all orientation operations will be with respect to the local coordinate frame of the coordinate-system-class object. The reference-coordinate-system may be anywhere in the model tree. If a subobject should follow the orientation of its parent or an object above it in the tree, then its reference-coordinate-system should explicitly reference that object. The reference-coordinate-system property can only refer to an object of class coordinate-system-class. References to any objects which do not inherit from coordinate-system-class will be ignored and orientation operations will be with respect to the global system. A group of methods and functions are available to convert coordinates and vectors between the three frames. Two methods, convert-coords and convert-vector, can operate between any two of the frames. Other methods and functions are available to convert between specific frames. These methods are described in the section Coordinate Conversions.

Important note on reference coordinate frames: Every object that inherits from position-object also has the property reference-object. AML 3.1.2 introduced the reference-coordinate-system to replace the

use of reference-object. By default, reference-object has the value nil. The reference-object of an instance affects the orientation similarly to reference-coordinate-system, but reference-object can point to any position-object instance. The methods and functions available that use reference-coordinatesystem behave similarly with reference-object. For backward compatibility purposes, if an object specifies both a reference-object and a reference-coordinate-system, reference-object takes precedence and reference-coordinate-system is ignored. It is recommended to use reference-coordinate-system.

Classes

coordinate-system-mixin [class]

The coordinate-system-mixin class is used to add properties to the position-object class to create the behavior of a coordinate frame. This class should not be instantiated - the class coordinate-system-class should be used in models. The properties of coordinate-system-mixin are used to determine its orientation with respect to its reference coordinate frame. The reference coordinate frame is the global coordinate system by default. The origin, vector-i, and vector-j properties specify the position and orientation of the object.

Inheritance: position-object

Properties:

- **origin**

  The origin property specifies the position of the origin of the coordinate-system-mixin object with respect to the reference coordinate system.

  Default Formula: '(0 0 0).

- **vector-i**

  The direction of the I-axis of the coordinate-system-mixin with respect to the reference coordinate system.

  Default Formula: '(1 0 0).

- **vector-j**

  The direction of the j-axis of the coordinate-system-mixin with respect to the reference coordinate system.

  Default Formula: '(0 1 0).

coordinate-system-class [class]

The coordinate-system-class provides an orthogonal reference frame which can be used to position other objects in a model. It uses the positioning functionality of the coordinate-system-mixin and adds a visible indicator of the coordinate frame position and orientation. The coordinate-system-class is used as a reference frame for other objects (including other coordinate-system-class objects). All objects that inherit from position-object have the property reference-coordinate-system. This property should point to a coordinate-system-class object that will be used as the reference frame for orienting the object. A coordinate-system-class object is drawn as a set of axes in the local x, y, and z directions; a box displaying x, y, and z; and the name of the coordinate-system-class object. These components can be individually turned off.

Inheritance: coordinate-system-mixin simple-geometry-class

Properties:

- **length**

  The length of the lines used to draw the x, y, and z axes.

  Default Formula: 1.0

- **colors**

  Specifies the colors used to draw the x, y, and z axes.

  Default Formula: '(red green blue).

- **draw-axes?**

  When t, lines will be drawn indicating the x, y, and z axes.

  Default Formula: '(t).

- **draw-box?**

  When t, a box indicating positive and negative x, y, and z directions will be drawn. When draw-box? is nil, it draws the "x", "y" and "z" label at the end of the axes.

  Default Formula: '(t).

- **label**

  Label string.

  Default Formula: '(object name).

- **draw-label?**

  When t, the label will be drawn.

  Default Formula: '(t).

- **negative-axes?**

  When t, the axis lines will be drawn in both the positive and negative directions.

  Default Formula: '(nil).

- **render**

  This property defined the graphics rendering method. Values allowed are: boundary (wireframe graphics), shaded (shaded representation), facet (surface facets representation), isoline (grid of lines following the parametric lines of the surfaces), boundaryshaded, facet-shaded, isoline-shaded. Isoline modes do not work on the Parasolid-based solid modeler.

  Default Formula: shaded

coordinate-system-orthonormal-class [class]

The class allows you to specify an x-axis-vector that will be the i-vector and instead of requiring the j-vector to be exactly perpendicular to the i-vector (as needed by the coordinate-system-class), the xy-plane-vector can be specified defining the i-j plane. The normal is computed using a cross-product of the 2 vectors, and then the j-vector is computed through the cross-product of the normal and the i-vector.

Inheritance: coordinate-system-mixin

Properties:

- **x-axis-vector**

  Default Formula: '(1.0 0.0 0.0)

- **xy-plane-vector**

  Default Formula: '(0.0 1.0 0.0)

Orientation Functions

These functions are used to incorporate the orientation of objects within their class definition. All objects that inherit from position-object have a property called orientation. This property is a list of orientation functions that describe the operations to be performed on an object immediately after creation; the functions describe how the object reached its current position. Any orientation operations which are performed on an object after instantiation (through orientation methods) will be added to the orientation property of that instance of the object and will not affect other instances of the same class. All orientations take place with respect to the reference coordinate frame of the object. By default, this frame is the global coordinate frame (the value of the reference-object and the reference-coordinate-system properties are nil). The format of the orientation property is:

```
orientation (list [(operation-1 args) (operation-2 args) ...])
```

The formula for orientation cannot be defined as a quoted (') list because it is not evaluated in the same manner as other properties. The operations in the list come from the functions described in this section. Consider the example of a box-object translated and rotated away from its original position.

```
(define-class example-box-object
    :inherit-from (box-object)
    :properties
    (
    orientation (list (translate '(5.0 0.0 0.0)) (rotate 45.0 :z-axis))
    )
 )
```

The orientation property specifies that the box should first be translated a distance of 5.0 along the x-axis (of the global coordinate frame). After that, it is rotated by 45.0 degrees about the z-axis (again, of the global coordinate frame). The orientation operations are built-into the object. Any instance of this object will be immediately transformed to the new orientation on creation. Note that the order of the operations in orientation is important. Reversal of the operations,

```
orientation (list (rotate 45.0 :z-axis) (translate (5.0 0.0 0.0)))
```

will result in a different final orientation.

The orientation functions can contain expressions which will be evaluated when the object is created. Thus, to move the box-object along the x-axis by a distance equal to twice its width and then rotate it, the following orientation can be used:

```
orientation (list (translate (list (* 2.0 !width) 0.0 0.0)(rotate 45.0 :z-
axis))
```

There are methods which provide transformations equivalent to the orientation functions which may be performed on objects which are already instantiated.

Functions

align [Function]

The align function is used in the orientation property of an object to orient that object with respect to another object or position. Alignment is based on a position and two vectors associated with the object and another set of position and vectors. Several flag arguments (which take values of t or nil) are available to control the behavior of the function (:move?, :align?, :orient?, and :global?). The position and vector arguments must be specified even if they are not required due to the settings of the flag arguments. The function align-object will perform the same operation on a pre-existing object.

Arguments:

- **point1**

  The source point to be moved to point2.

- **align-vector1**

  The vector that will be aligned parallel or colinear with align-vector2.

- **orient-vector1**

  The vector that will be oriented parallel or colinear with orientvector2. This will rotate the instance about the align-vector1 until the orient vectors are parallel.

- **point2**

  The target location for point1.

- **align-vector2**

  The vector with which align-vector1 will be aligned.

- **orient-vector2**

  The vector with which orient-vector1 will be aligned.

Keyword Arguments:

- **:move?**

  When t, it will translate point1 to point2.

  Default Value: t

- **:align?**

  When t, it will position by aligning the align-vectors.

  Default Value: t

- **:orient?**

  When t, the object will be positioned by aligning the orient-vectors.

Default Value: nil

- **:global?**

  A value of nil means that all of the points and vectors supplied are relative to the local coordinate system of the object being oriented. A value of t means that all of the points and vectors supplied are in the global coordinate system.

  Default Value: nil

Example 6-1. Align Example 1

```
(define-class ALIGN-EXAMPLE-1
  :inherit-from (object)
  :properties (
               pipe-diameter 0.75
               pipe-thickness 0.1
               )
  :subobjects (
               (box :class 'box-object
                    )
               (pipe :class 'pipe-object
                     outer-diameter ^^pipe-diameter
                     thickness ^^pipe-thickness
                     orientation (list
                                  (align (center-of-face ^^pipe 0)
                                         (normal-to-face ^^pipe 0)
                                         nil
                                         (center-of-face ^^box 4)
                                         (negate-vector
                                          (normal-to-face ^^box 4))
                                         nil
                                         :move? t :align? t :orient? nil)
                                  )
                     )
               )
  )
```

Example 6-2. Align Example 2

```
(define-class ALIGN-EXAMPLE-2
  :inherit-from (object)
  :properties (
               outer-diameter 3.0
               thickness 0.25
               )
  :subobjects (
               (STRAIGHT :class 'pipe-object
                         height 10.0
                         outer-diameter ^^outer-diameter
                         thickness ^^thickness
                         )
               (BEND :class 'pipe-elbow-object
                     elbow-radius 5.0
                     angle 90.0
```

```
                    outer-diameter ^^outer-diameter
                    thickness ^^thickness
                    orientation (list
                                 (align (center-of-face ^^bend 0)
                                        (normal-to-face ^^bend 0)
                                        '(0 1 0)
                                         (center-of-face ^^straight 1)
                                         (negate-vector
                                          (normal-to-face ^^straight 1))
                                         '(0 1 0)
                                          :move? t :align? t :orient? t)
                                 )
                    )
               )
  )
```

align-object [Function]

The align-object function is used to orient an object with respect to another object or position. Alignment is based on a position and two vectors associated with the object and another set of position and vectors. Several flag arguments (which take values of t or nil) are available to control the behavior of the function (:move?, :align?, :orient?, and :global?). The position and vector arguments must be specified even if they are not required due to the settings of the flag arguments. The function align can be used to perform the same operation in the orientation property of an object.

Arguments:

- **instance**

  An object instance that is to be aligned.

- **point1**

  The source point to be moved to point2.

- **align-vector1**

  The vector that will be aligned parallel or colinear with align-vector2.

- **orient-vector1**

  The vector that will be oriented parallel or colinear with the orientvector2. This will rotate the instance about align-vector1 until the orient-vectors are parallel.

- **point2**

  The target location for point1.

- **align-vector2**

  The vector with which align-vector1 will be aligned.

- **orient-vector2**

  The vector with which orient-vector1 will be aligned.

Keyword Arguments:

- **:move?**

  Translate point1 to point2.

Default Value: t

- **:orient?**

  This keyword controls whether the orient vectors will be aligned.

  Default Value: nil

- **:align?**

  Align the align-vectors.

  Default Value: t

- **:global?**

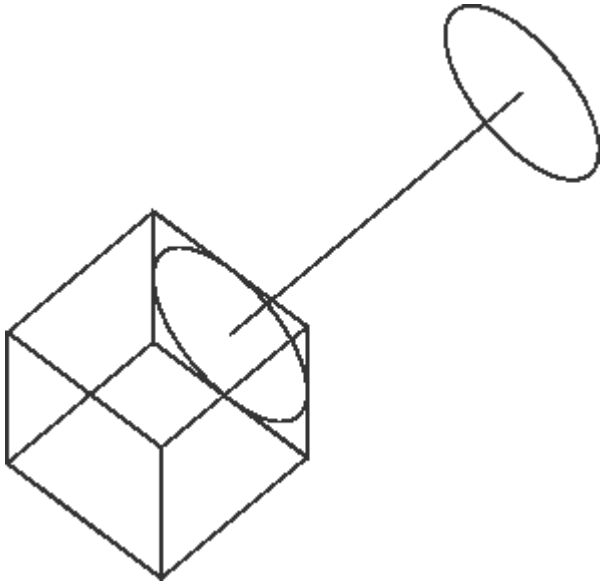
  A value of nil means that all of the points and vectors supplied are relative to the local coordinate system of the object being oriented. A value of t means that all of the points and vectors supplied are in the global coordinate system.

  Default Value: nil

Example 6-3. Align-object Example

```
(define-class ALIGN-OBJECT-EXAMPLE
  :inherit-from (object)
  :subobjects (
        (box :class 'box-object
         )
        (cyl :class 'cylinder-object
        )
       )
  )

(create-model 'align-object-example)
(align-object (the cyl) (center-of-face (the cyl) 0) (normal-to-face (the
cyl) 0) nil
(center-of-face (the box) 1) (negate-vector (normal-to-face (the box) 1))
nil)
(draw (the))
;The box and the cylinder will be drawn end-to-end
```

Figure 6-1. Face 1 of the box is aligned with face 0 of the cylinder.



apply-matrix [Function]

The apply-matrix function is used in the orientation property of an object to apply a combined rotation and translation operation to the object. This operation is specified in a 4x4 matrix. This function is used primarily to apply the effects of combined orientation operations on one object to another object. The method apply-matrix-to-object will perform the same operation on a pre-existing object.

Arguments:

- **matrix**

  A 4x4 matrix used to position the object. The matrix can also be a flat list of 16 values that represent a 4x4 matrix.

Example 6-4. Apply Matrix

```
(define-class APPLY-MATRIX-EXAMPLE
  :inherit-from (object)
  :subobjects (
               (box1 :class 'box-object
                     height 1.0
                     width 1.5
                     depth 2.0
                     )
               (box2 :class 'box-object
                     height 1.0
                     width 1.5
                     depth 2.0
                     orientation (list (apply-matrix
                                        '((-0.07487340109895328
0.856351602076828
                                                                -
0.5119902061584463 0.0)
```

```
                                                    (-0.8105613104305517
0.2473863845172211

0.5325982023589072 0.0)
                                                    ( 0.5827507298630535
0.45487689126119846

0.6756028167743183 0.0)
                                                    ( 1.6549746990203857
1.6125391721725464
                                                                        0.0 1.0))
                                                )
                                               )
                        )
                )
  )
```

move [Function]

The move function is used in the orientation property of an object to translate that object along the vector specified by two points. The object is moved so that the point initially at point1 is located at point2. This is equivalent to using the translate function in the form (translate (subtract-vectors point2 point1)). The arguments point1 and point2 can be the values returned by the geometric query functions (e.g., center-of-object, center-of-face, etc.) or any form which results in a 3-dimensional position. The method move-object will perform the same operation on a pre-existing object.

Arguments:

- **point1**

  3d location to determine the start for the move.

- **point2**

  3d location to determine the move distance from point1.

Example 6-5. Move Example

```
(define-class MOVE-EXAMPLE
  :inherit-from (object)
  :properties (
               height 10.0
               width 10.0
               depth 5.0
               )
  :subobjects (
               (box1 :class 'box-object)
               (box2 :class 'box-object
                     orientation (list (move (vertex-of-object (the box2) 0)
                                             (vertex-of-object (the box1)
6)))
                     )
               )
  )
```

rotate [Function]

The rotate function is used in the orientation property of an object to rotate that object about a specified vector. The vector is specified as a list of x,y,z coordinates. To rotate about a vector-object or line-object, use the function rotate-about-object. The method rotate-object will perform the same operation on a pre-existing object.

Arguments:

- **degrees**

  The number of degrees to rotate the object instance.

- **axis**

  Axis values may be :x-axis :y-axis :z-axis or a vector to rotate about.

Keyword Arguments:

- **:axis-point**

  The base point for the axis of rotation.

- **:global?**

  A value of nil means that all of the points and vectors supplied are relative to the local coordinate system of the object being oriented. A value of t means that all of the points and vectors supplied are in the global coordinate system.

  Default Value: nil

Example 6-6. Rotate Example

```
(define-class ROTATE-EXAMPLE
  :inherit-from (object)
  :subobjects (
               (brick :class 'box-object)
               (Shaft :class 'cylinder-object
                      height 2.0
                      orientation (list (rotate 45.0 :x-axis))
                      )
               )
  )
```

translate [Function]

The translate function is used in the orientation property of an object to translate that object along a vector. The vector is specified as a list of x,y,z coordinates. To translate along a vector-object, use the function translate-along-object(next). To translate the object between two coordinate positions, use the move function. The method translate-object will perform the same operation on a pre-existing object.

Arguments:

- **vector**

  A vector of the x y z translations.

Keyword Arguments:

- **distance**

  If a value is supplied for the distance, the object will be translated that distance along the vector. Otherwise, the object will be translated by the length of vector.

  Default Value: nil

- **:global?**

  A value of nil means that all of the points and vectors supplied are relative to the local coordinate system of the object being oriented. A value of t means that all of the points and vectors supplied are in the global coordinate system.

  Default Value: nil

Example 6-7. Translate Example

```
(define-class TRANSLATE-EXAMPLE
  :inherit-from (object)
  :subobjects (
               (brick-1 :class `box-object)
               (brick-2 :class `box-object
                        orientation (list (translate
                                           (list (the brick-1 width) 0.0
0.0)))
                        )
               )
  )
```

Transformation Methods

These methods are used to change the orientation of objects which have already been instantiated. Any object which inherits from position-object has an orientation which may be changed using these methods. The orientation property of a position-object contains a list of all operations which have been performed to give the object its current orientation. This property initially contains only those orientation functions given in the class definition that the object inherits from. Any additional operations performed by the transformation methods are added to this list. All orientations take place with respect to the reference coordinate frame of the object. By default, this frame is the global coordinate frame (the value of the reference-object and reference-coordinate-system properties are nil).

Consider the example of a box-object which will be rotated and translated. To create the object,

```
(create-model 'name-generator)
;the statement returns an instance of name-generator

(add-object (the) 'box 'box-object)
;the statement creates an instance of box-object

(translate-object (the box) '(5.0 0.0 0.0))
;translates the object a distance of 5.0 along the x-axis

(rotate-object (the-box) 45.0 :z-axis)
;rotates the object by 45.0 degrees about the z-axis
```

After each operation is performed, the current list of operations in the object's orientation property is returned. There are functions which provide operations equivalent to these methods that may be used within the orientation property of a class definition.

Transformation methods written on position-object:

```
apply-matrix-to-object
move-object
rotate-object
translate-object
```

Transformation methods written on graphic-object:

```
interactive-rotate
interactive-rotate-bounding-box
interactive-translate
interactive-translate-bounding-box
```

See Also: position object, graphic object

Methods

apply-matrix-to-object [Method]

Defined on Classes:

position-object

The apply-matrix-to-object method is used to apply a combined rotation and translation operation to an object. This operation is specified as a 4x4 matrix. This method is used primarily to apply the effects of combined orientation operations on one object to another object. The function apply-matrix can be used to perform the same operation in the orientation property of an object.

Arguments:

- **instance**

  An instance of a position-object.

- **matrix**

  A 4x4 matrix to position the object.

Example 6-8. Apply-matrix-to-object Example

```
(define-class MATRIX
  :inherit-from (object)
  :subobjects (
```

```
        (box1 :class 'box-object
         height 1.0
         width 1.5
         depth 2.0
        )
        (box2 :class 'box-object
         height 1.0
         width 1.5
         depth 2.0
        )
      )
 )
(create-model 'matrix)
(apply-matrix-to-object (the box2)
'((-0.07487340109895328 0.856351602076828
-0.5119902061584463 0.0)
(-0.8105613104305517 0.2473863845172211
0.5325982023589072 0.0)
(0.5827507298630535 0.45487689126119846
0.6756028167743183 0.0)
(1.6549746990203857 1.6125391721725464 0.0 1.0)))
;this will change the orientation of the box2
```

interactive-rotate [Method]

Defined on Classes:

graphic-object

The interactive-rotate method is used to change the orientation of an instance after creation by using the mouse. First select an object to rotate, then click and hold down the mouse button to rotate. During rotation, hold down [control] to constrain the rotation to the y axis, hold down [shift] to constrain the rotation to the x axis, and hold down both [control] and [shift] to constrain the rotation to the z axis.

Arguments:

- **instance-object**

  The graphic-object instance that is to be rotated interactively.

Keyword Arguments:

- **reference-point**

  This is the point to rotate the object about. The default will cause the object to rotate about its center.

  Default Value: nil

Example 6-9. Interactive-rotate Example

```
(create-model 'box-object)
(draw (the))
(interactive-rotate (the))
;by using the mouse the box can now be rotated
```

See Also: interactive-rotate-bounding-box, interactive-translate, interactive-translate-bounding-box, rotate-object

interactive-rotate-bounding-box [Method]

Defined on Classes:

graphic-object

This method is used to rotate an object after creation using the mouse. During rotation, the object will be displayed as a bounding box originally oriented along the global axes. The process begins by selecting the object. The object is moved by clicking and holding down the mouse button. While moving, holding down [control] will constrain motion to rotation about the y-axis, holding down [shift] will interactive-rotate-bounding-boxconstrain motion to rotation about the x-axis, and holding both [control] and [shift] will constrain motion to rotation about the z-axis.

Arguments:

- **instance**

  The graphic-object instance that is to be rotated interactively.

Keyword Arguments:

- **ref-point**

  The point about which the object rotates. The default value will produce rotation about the center of the object.

  Default Value: nil

- **line-type**

  The line type used to draw the bounding box. The default value will produce a dashed line.

  Default Value: 1

- **line-width**

  The width of the line used to draw the bounding box. The default value will produce a thin line.

  Default Value: 0

Example 6-10. Interactive-rotate-bounding-box Example

```
(create-model 'box-object)
(draw (the))
(interactive-rotate-bounding-box (the))
;by using the mouse the box can now be rotated
```

See Also: interactive-rotate, interactive-translate, interactive-translate-bounding-box, rotate-object

interactive-translate [Method]

Defined on Classes:

graphic-object

The interactive-translate method is used to change the orientation of an instance after creation by using the mouse. First select an object to move, then click and hold down the mouse button to move. During moving, hold down [control] to constrain the movement to the y direction, [shift] to constrain the movement to the x direction, and both [control] and [shift] to constrain the movement to the z direction.

Arguments:

- **instance-object**

  The graphic-object instance that is to be translated interactively.

Example 6-11. Interactive-translate Example

```
(create-model 'box-object)
(draw (the))
(interactive-translate (the))
;by using the mouse the box can now be moved
```

See Also: interactive-rotate, interactive-rotate-bounding-box, interactive-translate-bounding-box, rotate-object

interactive-translate-bounding-box [Method]

Defined on Classes:

graphic-object

This method is used to translate an object using the mouse. During translation, the object will be displayed as a bounding box originally oriented along the global axes. The process begins by selecting the object. The object is moved by clicking and holding down the mouse button. While moving, holding down [control] will constrain motion to the ydirection, holding down [shift] will constrain motion to the x-direction, and holding both [control] and [shift] will constrain motion to the z-direction.

Arguments:

- **instance**

  The graphic-object instance that is to be translated interactively.

Keyword Arguments:

- **line-type**

  The line type used to draw the bounding box. The default value will produce a dashed line.

  Default Value: 1

- **line-width**

  The width of the line used to draw the bounding box. The default value will produce a thin line.

  Default Value: 0

Example 6-12. Interactive-translate-bounding-box Example

```
(create-model 'box-object)
```

```
(draw (the))
(interactive-translate-bounding-box (the))
;by using the mouse the box can now be moved
```

See Also: interactive-rotate, interactive-rotate-bounding-box, interactive-translate, rotate-object

move-object [Method]

Defined on Classes:

position-object

The move-object method will translate an object instance the relative distance between two points after an object instance has been created. The move function can be used to perform the same operation in the orientation property of an object.

Arguments:

- **instance**

  An object instance that is to be moved.

- **point1**

  3d location to determine the start for the move.

- **point2**

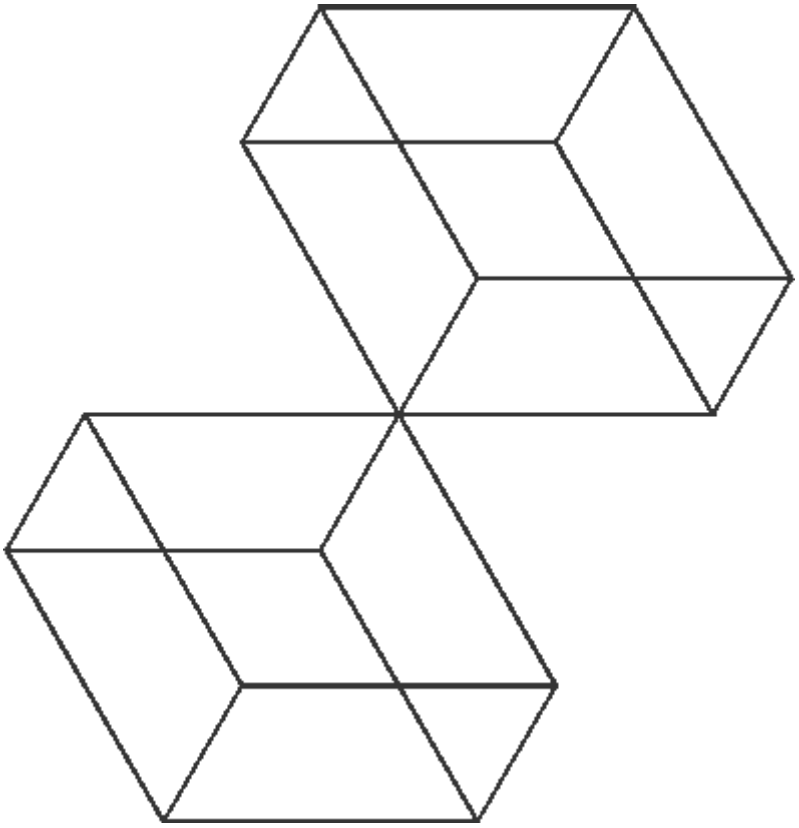
  3d location to determine the move distance from point1.

Note: Point1 and point2 may be the values returned by the methods: center-of-object, center-of-face, center-of-edge, vertex-of-object or any other form that results in a 3-dimensional coordinate.

Example 6-13. Move-object Example

```
(define-class MOVE-EXAMPLE
  :inherit-from (object)
  :properties (
        height 10.0
        width 10.0
        depth 5.0
             )
  :subobjects (
        (box1 :class 'box-object)
        (box2 :class 'box-object)
            )
 )
(create-model 'MOVE-EXAMPLE)
(move-object (the box2) (vertex-of-object (the box2) 0)
                (vertex-of-object (the box1) 6))
(draw (the))
;The two boxes will be drawn touching at one point.
;The front-left-bottom point on box2 is moved to the back-right-top point on
box1.
```

Figure 6-2. Vertex 0 of box2 is moved to vertex 6 of box 1.



rotate-object [Method]

Defined on Classes:

position-object

The rotate-object method is used to rotate an object about a specified vector. The vector is specified as either a list of x, y, z coordinates or a keyword specifying the x, y, or z axes. The function rotate can be used to perform the same operation in the orientation property of an object.

Arguments:

- **instance**

  An object instance that is to be rotated.

- **degrees**

  The number of degrees to rotate the object instance.

- **axis**

  Axis values may be :x-axis :y-axis :z-axis or a vector to rotate about.

Keyword Arguments:

- **:axis-point**

  The start point for the axis of rotation.

Default Value: (0 0 0)

- **:global?**

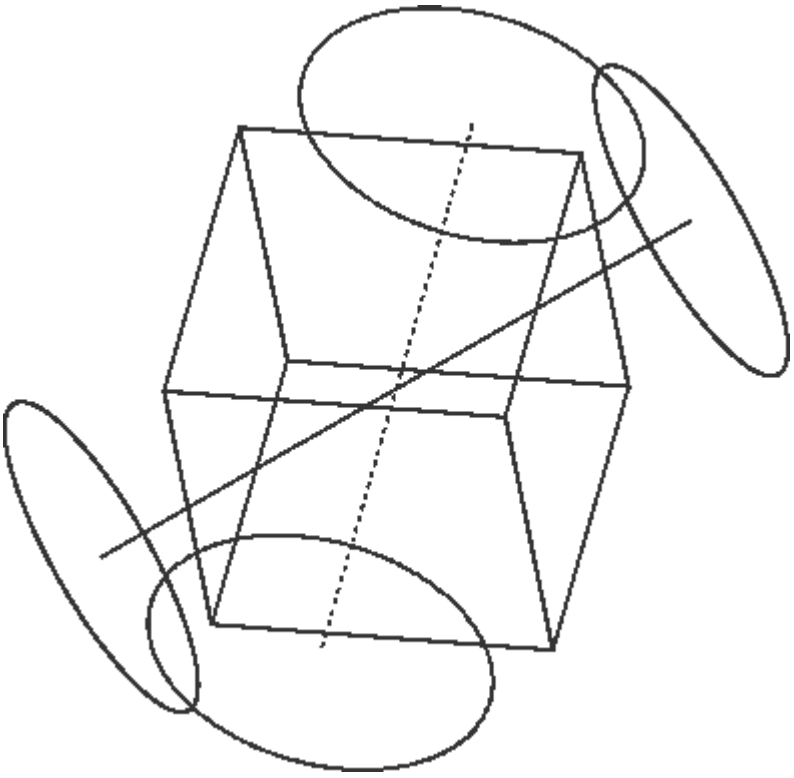
  A value of nil means that all of the points and vectors supplied are relative to the local coordinate system of the object being oriented. A value of t means that all of the points and vectors supplied are in the global coordinate system.

  Default Value: nil

Example 6-14. Rotate-object Example

```
;Assuming an instance of the class above is the root object (or model):
(define-class ROTATE-OBJECT-EXAMPLE
  :inherit-from (object)
  :subobjects (
        (brick :class 'box-object)
        (Shaft :class 'cylinder-object
          height 2.0
           )
        )
  )
(create-model 'rotate-object-example)
(rotate-object (the Shaft) 45.0 :x-axis)
```

Figure 6-3. The cylinder is rotated 45 degrees about the x-axis through the origin.



translate-object [Method]

Defined on Classes:

position-object

The translate-object method is used to translate an object along a vector. The vector is specified as a list of x, y, z coordinates. To translate along a vector-object, use the method translateobject-along-object. To translate the object between two coordinate positions, use the method move-object. The function translate can be used to perform the same operation in the orientation property of an object.

Arguments:

- **instance**

  An object instance that is to be translated.

- **vector**

  A vector of the x y z translations.

Keyword Arguments:

- **:distance**

  If a value is supplied for the distance, the object will be translated that distance along the vector. Otherwise, the object will be translated by the length of the vector.

  Default Value: nil

- **:global?**

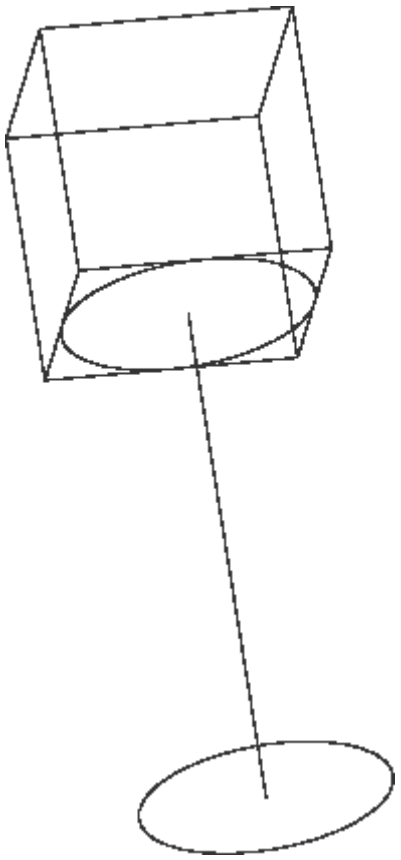
  A value of nil means that all of the points and vectors supplied are relative to the local coordinate system of the object being oriented. A value of t means that all of the points and vectors supplied are in the global coordinate system.

  Default Value: nil

Example 6-15. Translate-object Example

```
;Assuming an instance of the class above is the root object (or model):
(define-class TRANSLATE-OBJECT-EXAMPLE
   :inherit-from (object)
   :subobjects (
        (box :class 'box-object)
        (cyl :class 'cylinder-object)
            )
   )
(create-model 'TRANSLATE-OBJECT-EXAMPLE)
(translate-object (the cyl) (list 0.0 0.0 (half (+ (the cylheight) (the box
depth)))))
```

Figure 6-4. The cylinder is translated such that its face 1 is aligned with face 0 of the box.



Coordinate Conversions

All positions, dimensions, and orientations are specified with respect to one of three coordinate frames: global, local, and reference. Methods and functions are available to convert between these coordinate frames. Two methods, convert-coords and convert-vectors, may be used for any required conversions. Other methods are available to convert between two specific coordinate frames.

Functions and Methods

convert-coords [Method]

Defined on Classes:

position-object

Converts point coordinates from one object's coordinate frame to another.

Arguments:

- **instance**

  The object instance whose coordinate frames are used. The :ref frame will be based on this object's reference-object or referencecoordinate-system.

- **point**

  The coordinates (a list of '(x y z)) of the point in the :from coordinate frame which will be converted and returned in the :to coordinate frame.

Keyword Arguments:

- **:from**

  The coordinate frame in which point is specified. This can be either :local, :global, or :ref. The local coordinate frame (:local) is the frame in which the instance was originally defined. The global coordinate frame (:global) is the model (world) frame. The :ref frame is the local frame of the instance's reference-object or reference-coordinate-system. If there is no reference-object or reference-coordinate-system, the global coordinate frame is used.

  Default Value: :local

- **:to**

  The coordinate frame to which point should be converted. The possible frames are the same as for the :from frame.

  Default Value: :global

Note: When instance has both a reference-object and a reference-coordinate-system defined, reference-coordinate-system is ignored.

Example 6-16. Convert-coords Example

```
(convert-coords (the box-0001) '(1 0 0) :from :local :to :global)
```

See Also: convert-vector, local-point, global-to-local-frame, global-point-to-ref

convert-vector [Method]

Defined on Classes:

position-object

Converts a vector from one object's coordinate frame to another.

Arguments:

- **instance**

  The object instance whose coordinate frames are used. The :ref frame will be based on this object's reference-object or reference-coordinate-system.

- **vector**

  The vector (a list of '(x y z)) in the :from coordinate frame which will be converted and returned in the :to reference frame.

Keyword Arguments:

- **:from**

  The coordinate frame in which point is specified. This can be either :local, :global, or :ref. The local coordinate frame (:local) is the frame in which the instance was originally defined. The global

coordinate frame (:global) is the model (world) frame. The :ref frame is the local frame of the instance's reference-object or reference-coordinate-system. If there is no reference-object or reference-coordinate-system, the global coordinate frame is used.

Default Value: :local

- **:to**

  The coordinate frame to which point should be converted. The possible frames are the same as for the :from frame.

  Default Value: :global

Note: When instance has both a reference-object and a reference-coordinate-system specified, reference-coordinate-system is ignored.

Example 6-17. Convert-vector Example

```
(convert-vector (the box-0001) '(1 1 1) :from :local :to :global)
```

See Also: convert-coords, local-vector, global-to-local-frame, global-vector-to-ref

convert-2d-point-to-3d-point [Function]

This function takes a point in a two-dimensional coordinate system and converts it to a threedimensional system. The orientation of the two-dimensional system is specified by giving its origin and axes in the three-dimensional system.

Arguments:

- **local-point**

  A two-dimensional point in the coordinate system specified by origin, x-vector, and y-vector.

- **origin**

  The origin of the 2D coordinate-system within the 3D system.

- **x-vector**

  A vector defining the x-axis of the 2D coordinate system within the 3D system.

- **y-vector**

  A vector defining the y-axis of the 2D coordinate-system within the 3D system.

Example 6-18. Convert-2d-point-to-3d-point Example

```
(convert-2d-point-to-3d-point '(1.0 2.0) '(0.0 0.0 0.0) '(1.0 0.0 0.0) '(0.0
1.0 0.0))
(1.0 2.0 0.0)
(convert-2d-point-to-3d-point '(1.0 2.0) '(0.0 0.0 0.0) '(0.0 1.0 0.0) '(1.0
0.0 0.0))
(2.0 1.0 0.0)
(convert-2d-point-to-3d-point '(1.0 2.0) '(0.0 0.0 0.0) '(0.0 1.0 0.0) '(0.0
0.0 1.0))
(0.0 1.0 2.0)
```

global-point-to-ref [Method]

Defined on Classes:

position-object

This method translates a point from the global coordinate system to the local coordinate system of the specified object.

Arguments:

- **instance**

  An instance of a position-object.

- **point**

  A point specified as a vector or a list.

Example 6-19. Using global-point-to-ref

```
;Assuming an instance of the class above is the root object (or model):
(define-class EXAMPLE
     :inherit-from (object)
     :subobjects (
                (block-1 :class `box-object
                        color 'green
                    )
                (block-2 :class `box-object
                        orientation (list (rotate 45.0 '(0.0 1.0 0.0))
                        (translate '(2.0 0.0 0.0)))
                        color 'cyan
                    )
                (block-3 :class `box-object
                        reference-object (the block-2)
                        orientation (list (rotate 45.0 '(0.0 1.0 0.0))
                        (translate '(2.0 0.0 0.0)))
                        color 'red
                    )
        )
  )
(create-model 'example)
(global-point-to-ref (the block-1) '(0.0 0.0 0.0))
(0.0 0.0 0.0)
(global-point-to-ref (the block-2) '(0.0 0.0 0.0))
(0.0 0.0 0.0)
(global-point-to-ref (the block-3) '(0.0 0.0 0.0))
(-1.414213562373095 0.0 -1.414213562373095)
```

global-to-local-frame [Function]

Convert a point from the global coordinate system to the specified coordinate-system.

Arguments:

- **point**

  The global point to be translated into the local coordinate-system.

- **origin**

  A point to be used as the origin of the local coordinate-system.

- **i-vector**

  A vector to be used as the x-axis of the local coordinate-system.

- **j-vector**

  A vector to be used as the y-axis of the local coordinate-system.

Example 6-20. Global-to-local-frame Example

```
(global-to-local-frame '(1.0 0.0 0.0) '(0.0 0.0 0.0) '(1.0 0.0 0.0) '(0.0 1.0
0.0))
(1.0 0.0 0.0)
(global-to-local-frame '(1.0 0.0 0.0) '(0.0 0.0 0.0) '(0.0 1.0 0.0) '(1.0 0.0
0.0))
(0.0 1.0 0.0)
(global-to-local-frame '(1.0 0.0 0.0) '(0.0 0.0 0.0) '(1.0 1.0 0.0) '(1.0 -
1.0 0.0))
(0.7071067811865475 0.7071067811865475 0.0)
```

See Also: local-to-global-frame, global-point-to-ref, global-vector-to-ref, local-point, local-point-to-global, local-vector, local-vector-to-global.

global-vector-to-ref [Method]

Defined on Classes:

position-object

This method translates a vector from the global coordinate system to the local coordinate system of the specified object.

Arguments:

- **instance**

  An instance of a position-object.

- **vector**

  A vector specified as a vector or a list.

Example 6-21. Using global-vector-to-ref

```
;Assuming an instance of the class above is the root object (or model):
(define-class EXAMPLE
   :inherit-from (object)
   :subobjects (
```

```
                (block-1 :class `box-object
                        color 'green
                        )
                (block-2 :class `box-object
                        orientation (list (rotate 45.0 '(0.0 1.0 0.0))
                                          (translate '(2.0 0.0 0.0)))
                        color 'cyan
                        )
                (block-3 :class `box-object
                        reference-object (the block-2)
                        orientation (list (rotate 45.0 '(0.0 1.0 0.0))
                                          (translate '(2.0 0.0 0.0)))
                        color 'red
                        )
        )
   )
(create-model 'example)
(global-vector-to-ref (the block-1) '(1.0 0.0 0.0))
(1.0 0.0 0.0)
(global-vector-to-ref (the block-2) '(1.0 0.0 0.0))
(1.0 0.0 0.0)
(global-vector-to-ref (the block-3) '(1.0 0.0 0.0))
(0.7071067811865475 0.0 0.7071067811865475)
```

local-to-global-frame [Function]

This function converts a point's coordinates from a given local coordinates frame to the global coordinates frame. It returns a list of the global X,Y,Z coordinates.

Arguments:

- **point**

  The global point to be translated into the local coordinate-system.

- **origin**

  A point to be used as the origin of the local coordinate frame.

- **i-vector**

  A vector to be used as the x-axis of the local coordinate frame.

- **j-vector**

  A vector to be used as the y-axis of the local coordinate frame.

- **k-vector**

  A vector to be used as the z-axis of the local coordinate frame

See Also: global-to-local-frame, global-point-to-ref, global-vector-to-ref. local-point, local-point-to-global, local-vector, local-vector-to-global.

Chapter 7. Collection Objects

Introduction

This chapter describes the various objects used to create multiple objects (graphic and otherwise) and objects that combine multiple graphic objects into a single graphic object.

Objects producing a single graphic object

Linear-array-object and circular-array-object let the user create linear and circular patterns by specifying a source object and creating multiple copies combined into a single object. To take a list of existing graphic objects and combine them into a single graphic object, use assembly-object or group-object.

Objects creating multiple subobjects

Linear-clonified-object and circular-clonified-object let the user create linear and circular patterns by specifying a source object and creating multiple copies as separate objects. Series-object let the user create multiple subobjects under a single object. Each object in the series can be of an arbitrary class and need not be a graphic object. Further, every object in the series gets an index property that identifies the object's position in the series. Properties can also be specified for each object in the series through an init-form property in the series-object. Sequence-object is similar to series-object and creates a set of objects based on input conditions being satisfied.

Collection Objects

Classes and Methods

array-class [class]

The array-class is the class that the different types of array objects (arrays and clonified) inherit from.

Inheritance: object

Properties:

- **source-object**

  The object instance that will be copied and positioned in an array arrangement.

  Default Formula: nil

- **quantity**

  The number of copies that will appear in the array.

  Default Formula: 1

circular-array-object [class]

The array is created by making copies of the geometry of the source-object and placing them in a circle around the center at the given diameter. The circle is normal to the rotate-axis. The first copy is translated along the translate-axis and then rotated about the rotate-axis counter-clockwise by the start-

angle. Subsequent copies are each placed at a further counterclockwise position incremented by repeat-angle. The circular-array-object creates a single geometric assembly object. The array is similar to the circular-clonified-objects object except that there is only one object with a representation that is the complete array geometry. If individual geometry is required, use the circular-clonified-objects.

Inheritance: array-class simple-geometry-class

Properties:

- **source-object**

  The object to be copied.

  Default Formula: nil

- **diameter**

  The size of the circle of which the copies will be placed about.

  Default Formula: 1.0

- **start-angle**

  The angle at which to place the first array element (with reference to the translate-axis).

  Default Formula: 0.0

- **repeat-angle**

  The angular spacing between two consecutive array elements.

  Default Formula: (/ 360 !quantity)

- **translate-axis**

  The reference vector for positioning the first copy.

  Default Formula: x-axis

- **rotate-axis**

  defines the axis perpendicular to the plane of the circular array.

  Default Formula: y-axis

- **rotate-clones?**

  it decides whether or not the copies will be individually rotated as they are laid out in the circular pattern

  Default Formula: t

- **center**

  The center of the circular pattern, it defaults to the center of the source-object.

  Default Formula: (when (the source-object) (center-of-object (the source-object)))

- **quantity**

  The number of copies to create.

  Default Formula: 1

- **assembly?**

  t for creating an assembly and nil for creating a union.

Default Formula: t

- **ref-point**

  The local point on the source-object (and each copy) that lies on the described circle.

  Default Formula: center of the source-object

Figure 7-1. Circular Array Object (rotate-clones? = t)

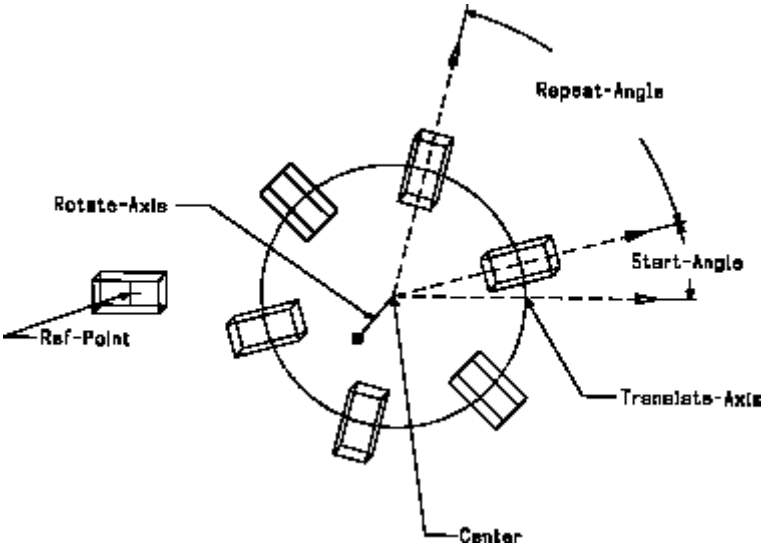
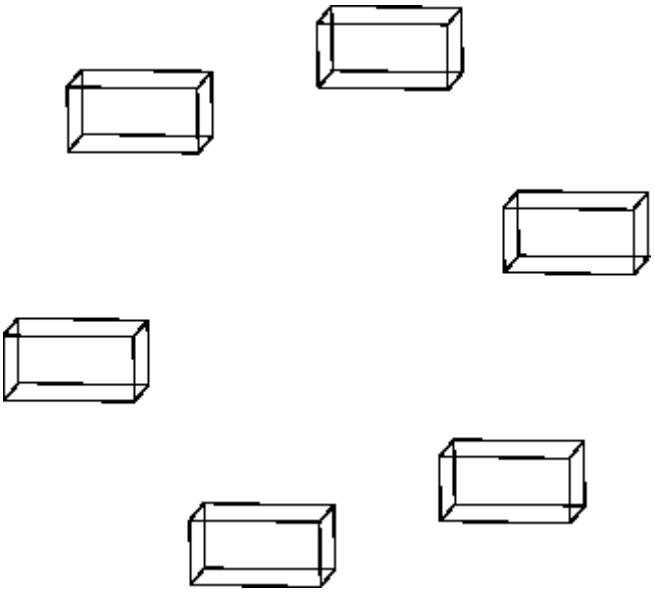


Figure 7-2. Circular Array Object (rotate-clones? = nil)



Example 7-1. Circular array example

```
(define-class circular-array-model
  :inherit-from (circular-array-object)
```

```
  :properties (
               source-object ^model-box
               quantity 6
               diameter 8
               center '(8 0 0)
               ref-point (center-of-object ^source-object)
               start-angle 15.0
               rotate-axis '(0 0 1)
               translate-axis '(1 0 0)
               )
  :subobjects (
               (model-box :class 'box-object
                          width 2
                          height 1.0
                          depth 1.5
                          )
               )
  )
;;Output
(create-model 'circular-array-model)
;; creates new circular-array model
(print-tree (The) :Expand? t)
;;prints the exapanded tree of the model:
CIRCULAR-ARRAY-MODEL
MODEL-BOX
```

Note: This is an equivalent class to the circular-clonified-objects. The difference is that this class will instantiate into only 2 instances whereas the circular-clonified-objects example will create 8 instances.

circular-clonified-objects [class]

The array is created by making copies of the geometry of the source-object and placing them in a circle around the center at the given diameter. The circle is normal to the rotate-axis. The first copy is translated along the translate-axis and then rotated about the rotate-axis counter-clockwise by the start-angle. Subsequent copies are each placed at a further counterclockwise position incremented by repeat-angle. The circular-clonified-objects arranges copies (array elements) of the given object into an array spaced about the circumference of a circle. The object creates a sub-object called clone-parent and the array elements are created as subobjects of the clone-parent.

Inheritance: array-class

Properties:

- **source-object**

  The object to be copied.

  Default Formula: nil

- **diameter**

  The size of the circle of which the copies will be placed about.

  Default Formula: 1.0

- **start-angle**

  The angle at which to place the first array element (with reference to the translate-axis).

  Default Formula: 0.0

- **repeat-angle**

  The angular spacing between two consecutive array elements.

  Default Formula: (/ 360 !quantity)

- **translate-axis**

  The reference vector for positioning the first copy.

  Default Formula: x-axis

- **rotate-axis**

  defines the axis perpendicular to the plane of the circular array.

  Default Formula: y-axis

- **repeat-angle**

  The spacing between two consecutive array elements.

  Default Formula: (/ 360 !quantity)

- **rotate-clones?**

  it decides whether or not the copies will be individually rotated as they are laid out in the circular pattern

  Default Formula: t

- **center**

  The center of the circular pattern, it defaults to the center of the source-object.

  Default Formula: (when (the source-object) (center-of-object (the source-object)))

- **quantity**

  The number of copies to create.

  Default Formula: 1

- **assembly?**

  t for creating an assembly and nil for creating a union.

  Default Formula: t

- **ref-point**

  The local point on the source-object (and each copy) that lies on the described circle.

  Default Formula: center of the source-object

For an example and figures please refer to circular-array-object's example and figures. Note: for the example we'll replace the name of the class with circular-clonified-model and we'll inherit form cicrular-clonified-objects. Then our ouptut would be:

```
(create-model 'circular-clonified-model)
;;creates the new circular-clonified model
(print-tree (The) :Expand? t)
;; and the tree is printed:
CIRCULAR-CLONIFIED-MODEL
MODEL-BOXCLONE-PARENT
"0 CLONE-PARENT"
"1 CLONE-PARENT"
"2 CLONE-PARENT"
"3 CLONE-PARENT"
"4 CLONE-PARENT"
"5 CLONE-PARENT"
```

linear-array-object [class]

The linear-array-object creates a single geometric assembly object. The array is similar to the linear-clonified-objects object except that there is only one object with a representation that is the complete array geometry. If individual geometry is required use the linear-clonified-objects object.

Inheritance: simple-geometry-class

Properties:

- **source-object**

  The object to be copied.

  Default Formula: nil

- **quantity**

  The number of copies to create.

  Default Formula: 1

- **start-distance**

  The distance between the local origin and the first element.

  Default Formula: 0.0

- **spacing**

  The distance between two consecutive element centers.

  Default Formula: 1.0

- **vector**

  A vector that defines the linear direction for placing the elements.

  Default Formula: '(0.0 1.0 0.0)

- **assembly?**

  t for creating an assembly and nil for creating a union.

  Default Formula: t

- **ref-point**

  The local point on the source-object (and each copy) that lies on the described circle.

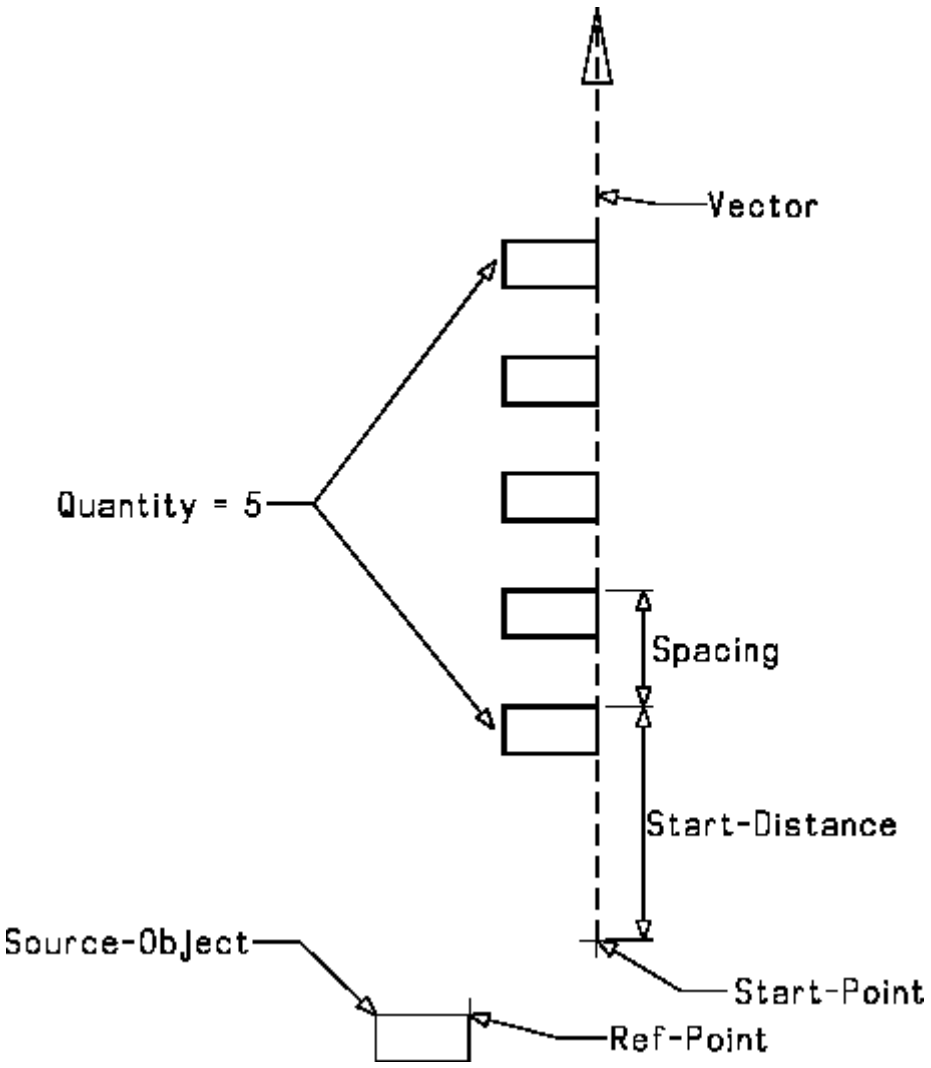
Default Formula: center of the source-object

- **start-point**

  The initial location of the ref-point in the array.

  Default Formula: (when (the source-object) (center-of-object (the source-object)))

Figure 7-3. Linear Array Object



Example 7-2. Linear Array Example

```
(define-class linear-array-model
  :inherit-from (linear-array-object)
  :properties (
               source-object ^model-box
               quantity 5
               spacing 2.5
               start-distance 5.0
               start-point '(3.5 2.0 0.0)
```

```
                ref-point (center-of-edge ^source-object 10)
                vector '(0 1 0)
                )
  :subobjects (
                (model-box :class 'box-object
                           width 2.0
                           height 1.0
                           depth 1.5
                           )
                )
  )


;;; Output
(create-model 'linear-array-model)
;; creates a new linear-array model
(print-tree (The) :Expand? t)
;; prints the tree:
LINEAR-ARRAY-MODEL
MODEL-BOX
```

Note: This is an equivalent class to the linear-clonified-objects object. The difference is that this class will instantiate into only 2 instances whereas the linear-clonified-objects example will create 8 instances.

linear-clonified-objects [class]

The linear-clonified-objects object arranges copies of the geometry (array elements) of the given object into an array spaced in a line. If individual geometric copies of the object are not necessary and speed is desired use the linear-array-object class instead.

Inheritance: simple-geometry-class

Properties:

- **source-object**

  The object to be copied.

  Default Formula: nil

- **quantity**

  The number of copies to create.

  Default Formula: 1

- **start-distance**

  The distance between the local origin and the first element.

  Default Formula: 0.0

- **spacing**

  The distance between two consecutive element centers.

  Default Formula: 1.0

- **vector**

A vector that defines the linear direction for placing the elements.

Default Formula: '(0.0 1.0 0.0)

For an example and figures please refer to linear-array-object example and figures. Note: for the example we'll replace the name of the class with linear-clonified-model and we'll inherit form linear-clonified-objects. Then our ouptut would be:

```
(create-model 'linear-clonified-model)
;; creates new linear-clonified model
(print-tree (The) :Expand? t)
;; prints:
LINEAR-CLONIFIED-MODEL
MODEL-BOX
CLONE-PARENT
"0 CLONE-PARENT"
"1 CLONE-PARENT"
"2 CLONE-PARENT"
"3 CLONE-PARENT"
"4 CLONE-PARENT"
```

Note: Only the geometry is copied from the source-object. If more than the geometry is required see the series-object.

series-object [class]

This class creates an arbitrary number of subobjects. The classes, quantity and properties of the subobjects created are based on the property formulas of class-expression, quantity, and init-form provided. Changing the init-form causes all the formulas of the appropriate properties to be changed to the value specified in the init-form. (Any property specified in the init-form that does not exist on the series subobject will be ignored). Each subobject created has three properties called index, series-previous and series-next automatically added to it. The index starts at 0 and is incremented by 1 for each object being added. The index may be used in the class expression or any of the properties to allow the subobjects to have different characteristics.The properties series-next and series-previous call the functions series-next and series-previous respectively, to return the instance of the next and previous object in the series respectively.

Warning: Currently the subobjects are created immediately after a property is changed or smashed. If a dependency to another series-object exists, circularity will occur.

Inheritance: object

Properties:

- **series-objects**

  The list of series subobjects that were created

  Default Formula: (add-series (superior (current-object)))

- **series-prefix**

  The prefix to be used in naming the series subobjects.

  Default Formula: (princ-to-string (object-name (superior (current-object))))

- **init-form**

  The default formulas for the subobjects.

  Default Formula: nil

- **quantity**

  The number of series subobjects to create. Note: This property inherits from a special class (model-save-property-class), the user should not set the class of this property. (model-save-property-class inherits from property object. Properties of this type will have an entry in the model-save file to guarantee that the dependcies associated will these properties are considered.)

  Default Formula: 0

- **class-expression**

  An expression used to determine which classes to create as subobjects. This expression has to be quoted. Note: This property inherits from a special class (model-save-property-class), the user should not set the class of this property. (model-save-property-class inherits from property object. Properties of this type will have an entry in the model-save file to guarantee that the dependencies associated will these properties are considered.)

  Default Formula: 'object

- **recreate?**

  When this property is non-nil, the series smashes and recreates all the series subobjects whenever the quantity property changes. When nil, only needed subobjects are added/deleted whenever the quantity property changes. This property defaults to t only for backward compatibility with AML versions prior to 3.1.1; however, a series-object behaves more efficiently when recreate? is nil.

  Default Formula: t

Example 7-3. Series Object Example

```
(define-class TABLE-LEGS
  :inherit-from (series-object)
  :properties (
               leg-height 10.0
               leg-diameter 1.0
               leg-vertices '(0 1 4 5)
               series-prefix 'leg
               quantity 4
               class-expression 'cylinder-object
               init-form '(
                           height ^^leg-height
                                  diameter ^^leg-diameter
                                  orientation (list (rotate 90 :x)
                                                    (move (center-of-face
(the cylinder-object) 1)
                                                          (vertex-of-object
(the table)

(nth !index !leg-vertices)))))
               )
  )
```

```
(define-class TABLE
  :inherit-from(box-object)
  :properties(
              height 1.0
              width 10.0
              depth 10.0
              )
  :subobjects(
              (legs :class 'table-legs
                    )
              )
  )
```

```
Example for series-object where the class-name can depend on the index:
```

```
(define-class test-series
  :inherit-from (series-object)
  :properties (
               class-names-list '(box-object cylinder-object sheet-object)
               quantity (length ^class-names-list)
               init-form '(
                           ;<properties-for-each-object>
                           )
               class-expression '(nth !index !class-names-list)
               )
  )
```

```
Note that the class-expression is evaluated at the level of the series-object
so
the index and the class-names-list are referenced using (the index) and (the
class-names-list) and not using (the superior index) and (the superior class-
names-list)
```

create-series [Method]

Defined on Classes:

series-object

This is the mechanism that creates the series subobjects of a series object. It is intended for internal use only.

Arguments:

- **instance**

  An object of type series-object.

series-first [Method]

Defined on Classes:

series-object

Returns the first series subobject from an object that inherits from series-object or series-parent-object.

Arguments:

- **instance**

  An instance of a series-object or a series-object.

Example 7-4. Series-first example

```
> (create-model 'table)
;; creates a new table model
> (series-first (the legs))
;; returns an instance of the legs of the table
> (find-tree (series-first (the legs)))
;; returns the first leg in the tree with root legs:
(THE TABLE LEGS LEG-0000)
```

series-last [Method]

Defined on Classes:

series-object

Returns the last series subobject from an object that inherits from series-object or series-parent-object.

Arguments:

- **instance**

  An instance of a series-object or a series-object.

Example 7-5. Series-last example

```
> (create-model 'table)
;; creates a new table model
> (series-last (the legs))
;; returns an instance of the last leg
> (find-tree (series-last (the legs)))
;; returns the last leg in the list :
(THE TABLE LEGS LEG-0003)
```

series-members [Method]

Returns a list of the member subobjects of the series.

sequence-object [class]

The sequence-object creates a set of subobjects based on the contents of the "sequence-list" and "condition-list" properties. The condition-list is a list of expressions each of which evaluates to t or nil (one of which is expected to be true at any time). The elements of sequence-list are corresponding lists of object and class names. The elements in the "condition-list" property are evaluated in order and the position of the satisfied condition is used to select the element within the sequence-list to create the subobjects.

Inheritance: series-object

Properties:

- **condition-list**

  A list of conditions, one of which will be true at any time and its index will be used to determine the objects to be created using the sequence-list.

  Default Formula: nil

- **sequence-list**

  A list of lists that specify possible subobject combinations. Each element of the list has the format: (obj-name-11 class-name-11 obj-name-12 class-name-12 ...). The elements of the list do not all need to have the same number of objects. The element of the list used to create the subobjects depends on the index of the first satisfied condition from the condition-list.

  Default Formula: nil

- **init-form-list**

  A list of lists that specify possible subobject initializations based on the condition-list. This list corresponds to the sequence-list and the condition-list and is used to initialize the objects created using the index of the satisfied condition in the condition-list.

  Default Formula: nil

Example 7-6. Sequence-object example

```
(define-class SEQUENCE-EXAMPLE
  :inherit-from (sequence-object)
  :properties (
               init-form-list (default nil)
               i 0
               condition-list '(('< !i 0)
                                (= !i 0)
                                ('< !i 5)
                                t)
               sequence-list '((p1 point-object s1 sheet-object)
                               (b1 box-object c1 cylinder-object)
                               (e1 elbow-object)
                               (a1 arc-object))
               )
  )


(create-model 'sequence-example)
;; creates a new sequence model
(print-tree (the) :expand? t)
;; prints the expanded tree :
SEQUENCE-EXAMPLE
B1
C1
;;Here, based on the value of i, the objects
;; p1,s1 for i '< 0
```

```
;; b1,c1 for i = 0
;; e1 for i '< 5
;; a1 for all other cases
```

See Also: simple-sequence-class, which is more user friendy for this type of usage.

simple-sequence-class [class]

Sequences are series objects that create subobjects based on the specifications in the object-class-name-list. The object-class-name-list is expected to be a list of a single list, where the inner list contains name class pairs to be created. Init-form works like the init-form in a series-object.

Inheritance: sequence-object

Properties:

- **object-class-name-list**

  A list of a single list where the inner list is a name, class pair.

  Default Formula: nil

- **init-form**

  The default formula for the subobjects.

  Default Formula: nil

Example 7-7. Simple-sequence-class Example

```
(define-class simple-sequence-test-class
 :inherit-from (simple-sequence-class)
 :properties (
              object-class-name-list (list (list 'box-1 'box-object
                                                 'cyl-1 'cylinder-object
                                                 'cone-1 'cone-object
                                                 'box-2  'box-object
                                                 )
                                           )
              init-form '(
                          height 2.0
                          diameter 4.0
                          width 3
                          depth 4
                          )
              )
 )
```

Chapter 8. Graphics and Visualization

Graphic Functions

Functions

add-light [Function]

Function to add a light object to the active graphics canvas (also referred to as the current-display). This affects the display of shaded images. If any of the arguments is not supplied, a form will be displayed for the user to enter the information.

Keyword Arguments:

- **name**

  The object name (quoted) of the light-object to be added.

- **color**

  The color for the light source.

- **x**

  The x direction for the light source.

- **y**

  The y direction for the light source.

- **z**

  The z direction for the light source.

Example 8-1. Add-light example

```
;; Assume the AML graphics server has been started (see aml)
;;and an active graphics display exist (see the method activate-display).
(create-model 'cylinder-object)
;; creates a new cylinder model
(draw (the))
;; Draws the cylinder
(zoom :all)
;; Zooms in for best view
(change-view :iso)
;; Rotates the cylinder - isometric
(shade (the))
;; Shades the cylinder
(add-light :name 'light1 :color 'white :x 0.2 :y 0.6 :z 0.4)
;; adds white light from the given angle
(add-light :name 'light2 :color 'white :x -0.2 :y -0.6 :z -0.4)
;; adds white light from the given angle
```

adjust-camera [Function]

This function adjusts the camera location to avoid unexpected clipping of lines in the current graphic display.

angle-between-2-vectors-in-pixel [Function]

Returns the angle between two vectors in screen pixels.

Arguments:

- **vector1**

  A three dimensional vector (in list format) with a z value of 0.

- **vector2**

  A three dimensional vector (in list format) with a z value of 0.

Example 8-2. Angle-between-2-vectors-in-pixel example

```
> (angle-between-2-vectors-in-pixel (vector 10 0 0) (vector 10 10 0))
;; returns 45.00000125223908
```

change-view [Function]

Change the view of the display to the supplied view-name.

Arguments:

- **view-name**

  Valid view names are :front :back :top :bottom :right :left :isometric. If no view is supplied, a menu of available views will be displayed and the view will be changed to the selected view.

Example 8-3. Change-view Example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (create-model 'cylinder-object)
;; creates a new cylinder model
> (draw (the))
;; draws the cylinder
> (change-view :iso)
;; changes the view to isometric
> (change-view :front)
;; changes the view to front
```

change-view-vectors [Function]

Sets the view of the current display according to the eye-vector and up-vector specified.

Arguments:

- **eye-vector**

List representing the X,Y,Z global components of the normal vector to the desired view plane. The eye-vector is always directed towards the "eyes" of the user.

- **up-vector**

  List representing the X,Y,Z global components of the vector lying on the desired view plane defining the up direction.

Keyword Arguments:

- **:update?**

  Optional keyword. Defaults to true refresh the current display window. When nil, the current display window is not refreshed.

  Default Value: t

See Also: current-eye-vector, current-up-vector

clear [Function]

Erase the graphics in the current display window.

Example 8-4. Clear example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (create-model 'cylinder-object)
;; creates a new cylinder model
> (draw (the))
;; draws the cylinder
> (clear)
;; clears the screen
```

current-camera-field [Function]

Returns a list of the width and height (respectively) of the camera field (view rectangle) of the current display. This function does not take any arguments.

See Also: set-camera-field

current-camera-target-position [Function]

Returns a list of the x,y,z global coordinates of the camera target position of the current display. The camera target position is the point in global coordinates that defines the center of the viewing area. This function does not take any arguments.

current-display [Function]

Returns the current active display.

See Also: current-display-id

current-display-id [Function]

Returns the window id for the current active display.

See Also: current-display

current-eye-vector [Function]

Current-eye-vector returns a list of the X,Y,Z components of the eye-vector of the current display. The eye-vector is the normal vector to the view plane. The eye-vector is always directed towards the "eyes" of the user. This function does not take any arguments.

See Also: change-view-vectors, current-up-vector

current-up-vector [Function]

Current-up-vector returns a list of the X,Y,Z components of the up-vector of the current display. The up-vector is a vector inside the view plane that defines the up direction of the view. This function does not take any arguments.

See Also: change-view-vectors, current-eye-vector

delete-all-lights [Function]

This function deletes all the lights on all available canvas or display object instances of the current AML session

delete-current-display-lights [Function]

This function deletes all the lights on all available canvas or display object instances of the current AML session

drag-line-with-angle [Function]

Allows the user to drag a line at a specified angle to select the proper distance. Returns the mouse button selected and the end point.

Arguments:

- **start-point**

  The starting point of the line.

- **angle**

  The angle the line will extend from the start point.

Keyword Arguments:

- **:values?**

  If t, the keyword will return a list of values - the initial return value for mouse button and the coordinates of the point selected by the user.

  Default Value: nil

Example 8-5. Drag-line-with-angle example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (create-model 'name-generator)
```

```
;; creates a new name-generator model
> (add-object (the) 'grid-object 'grid-object)
;; adds a grid to the object
> (draw (the grid-object))
;; draws the grid
> (change-view :front)
;; changes the view to a front xy plane view
> (drag-line-with-angle '(0.0 0.0 0.0) 45.0)
;; draws a line with a start point at origin and
;; angle of 45 degr. and allows user to select end point
```

drag-line-with-distance [Function]

Allows the user to drag a line of a specified fixed length to select the proper angle. Returns the mouse button selected and the end point.

Arguments:

- **start-point**

  The starting point of the line.

- **distance**

  The length of the line.

Keyword Arguments:

- **:values?**

  If t, the keyword will return a list of values - the initial return value for mouse button and the coordinates of the point selected by the user.

  Default Value: nil

Example 8-6. Drag-line-with-distance

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
;;Assume the same setup as the example above
> (drag-line-with-distance '(0.0 0.0 0.0) 5.0)
;; creates a line from the origin with distance 5
;; and allows the user to change the anlge
```

draw-3d-line [Function]

Used to draw a line in the display pane.

Arguments:

- **point1**

  The starting point of the line.

- **point2**

  The ending point of the line.

Keyword Arguments:

- **:color-id**

  The id of the color to use for drawing.

- **:display-id**

  The id of the display pane to draw the line in.

- **:draw-type**

  Valid draw types are :flip, :draw, and :erase.

  Default Value: :draw

Example 8-7. Draw-3d-line example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
(draw-3d-line '(0.0 0.0 0.0) '( 5.0 5.0 5.0) :color-id (get-color 'green)
:draw-type :flip)
```

draw-3d-point [Function]

The draw-3d-point function will draw a point at a given 3d coordinate.

Arguments:

- **point**

  The coordinates of the point to be drawn.

Keyword Arguments:

- **:thickness**

  The number of pixels wide the point will be.

- **:draw-type**

  Valid draw types are :flip, :draw, and :erase.

  Default Value: :draw

Example 8-8. Draw-3d-point example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
(draw-3d-point '(0.0 0.0 0.0) :thickness 3)
```

draw-3d-point-center-2d-circle [Function]

Used to draw a circle on the screen by specifying a point on the circumference of the circle and the center of the circle.

Arguments:

- **point**

  A point on the circumference of the circle.

- **center**

  The center of the circle.

Keyword Arguments:

- **:display-id**

  The id number of the display pane to draw the circle in.

- **:draw-type**

  Valid draw types are :flip, :draw, and :erase.

  Default Value: :flip

- **:filled?**

  A non nil value will cause the pixels enclosed in the arc to be drawn.

  Default Value: nil

Example 8-9. Draw-3d-point-center-2d-circle example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
(draw-3d-point-center-2d-circle '(3.0 3.0 0.0) '(5.0 5.0 0.0))
;; just line
(draw-3d-point-center-2d-circle '(4.0 4.0 0.0) '(5.0 5.0 0.0) :filled? t)
;; solid
```

draw-arc [Function]

Used to draw an elliptical arc segment in the display pane in the specified pixel area.

Arguments:

- **x-center**

  The x pixel location for the center of the arc.

- **y-center**

  The y pixel location for the center of the arc.

- **x-radius**

  The radius of the arc along the horizontal axis of the display.

- **y-radius**

  The radius of the arc along the vertical axis of the display.

- **start-angle**

  The start angle with respect to the horizontal of the display.

- **sweep-angle**

  The angle the arc is swept from the start angle.

Keyword Arguments:

- **:display-id**

  The id number of the display pane to draw the arc in.

- **:draw-type**

  Valid draw types are :flip, :draw, and :erase.

  Default Value: :flip

- **:filled?**

  A non nil value will cause the pixels enclosed in the arc to be drawn.

  Default Value: nil

Example 8-10. Draw-arc example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
>(draw-arc 200 200 50 100 0.0 180.0 :filled? nil)
>(draw-arc 400 200 50 100 0.0 180.0 :filled? t :draw-type :flip)
```

draw-circle-segment [Function]

Used to draw a circle segment in the display pane in the specified pixel area.

Arguments:

- **center-x**

  The x pixel location for the center of the circle segment.

- **center-y**

  The y pixel location for the center of the circle segment.

- **radius**

  The radius of the circle.

Keyword Arguments:

- **:start-angle**

  The start angle with respect to the horizontal of the display.

- **:angle**

  The angle the arc is swept from the start angle.

- **:display-id**

  The id number of the display pane to draw the circle in.

- **:draw-type**

  Valid draw types are :flip, :draw, and :erase.

  Default Value: :flip

- **:filled?**

  A non nil value will cause the pixels enclosed in the arc to be drawn.

Default Value: nil

Example 8-11. Draw-circle-segment example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
(draw-circle-segment 200 200 50 :start-angle 0.0 :angle 180.0 :draw-type
:flip)
(draw-circle-segment 400 200 50 :start-angle 0.0 :angle 360.0 :draw-type
:flip :filled? t)
```

draw-line [Function]

Used to draw a line in the display pane in the specified pixel location.

Arguments:

- **x1**

  The horizontal location of the line start point from the left edge of the display. The value must be an INTEGER.

- **y1**

  The vertical location of the line start point from the top of the display. The value must be an INTEGER.

- **x2**

  The horizontal location of the line end point from the left edge of the display. The value must be an INTEGER.

- **y2**

  The vertical location of the line end point from the top of the display. The value must be an INTEGER.

Keyword Arguments:

- **:display-id**

  The display-id for the display in which to draw. The value must be an INTEGER.

- **:draw-type**

  Valid draw types are :flip, :draw, and :erase.

  Default Value: :draw

Example 8-12. Draw-line example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
(draw-line 0 500 200 0 :draw-type :flip)
(draw-line 10 10 50 50 :draw-type :draw)
(draw-line 10 10 50 50 :draw-type :erase)
```

draw-nurb-curve [Function]

This function behaves the same as draw-line and draw-arc functions. It draws a given nurb curve in the current graphics window (the curve disappears when the graphics window updates).

Arguments:

- **degree**
- **knots-list**
- **control-points-list**

Keyword Arguments:

- **:display-id**

  The id number of the display pane to draw the nurb-curve in.

  Default Value: (current-display-id)

- **:draw-type**

  Valid draw types are :flip, :draw, and :erase.

  Default Value: :draw

- **:num**

  Default Value: 50

draw-point [Function]

The draw-point function will draw a single pixel on the display in pixel space starting from the upper left corner as 0 0.

Arguments:

- **x**

  The horizontal location of the point from the left edge of the display. The value must be an INTEGER.

- **y**

  The vertical location of the point from the top of the display. The value must be an INTEGER.

Keyword Arguments:

- **:display-id**

  The display-id for the display in which to draw. The value must be an INTEGER.

- **:draw-type**

  Valid draw types are :flip, :draw, and :erase.

  Default Value: :draw

Example 8-13. Draw-point example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
```

```
(draw-point 10 10 :draw-type :draw)
```

draw-point-center-circle [Function]

The draw-point-center-circle function draws a circle that goes through the pixel location x1, y1 and is centered at the pixel location x2, y2.

Arguments:

- **x1**

  The x pixel location for a point on the circumference of the circle.

- **y1**

  The y pixel location for a point on the circumference of the circle.

- **x2**

  The x pixel location for the center of the circle.

- **y2**

  The y pixel location for the center of the circle.

Keyword Arguments:

- **:display-id**

  The id number of the display pane to draw the circle in.

- **:draw-type**

  Valid draw types are :flip, :draw, and :erase.

  Default Value: :flip

- **:start-angle**

  The start angle of the arc with respect to horizontal.

- **:angle**

  The sweep angle of the arc.

- **:filled?**

  A non nil value will cause the pixels enclosed in the arc to be drawn.

  Default Value: nil

Example 8-14. Draw-point-center-circle

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
(draw-point-center-circle 200 200 300 300 :draw-type :flip :start-angle 0.0
:angle 90.0)
```

draw-string [Function]

The draw-string function will draw the supplied string to on the display at a given pixel location. The text is not rotatable with views, it will always appear as if it were a front view.

Arguments:

- **x**

  The horizontal location of the text start point from the left edge of the display. The value must be an INTEGER.

- **y**

  The vertical location of the text start point from the top of the display. The value must be an INTEGER.

- **string**

  A to be displayed.

Keyword Arguments:

- **:display-id**

  The display-id for the display in which to draw. The value must be an INTEGER.

- **:draw-type**

  The type of draw to perform. The options are :draw, :erase, :flip.

  Default Value: :flip

Example 8-15. Draw-string example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
(draw-string 100 100 "Hello World")
(draw-string 20 20 "Hello World" :draw-type :flip)
```

draw-vector [Function]

Draw-vector is used to display a vector in the display pane.

Arguments:

- **start-point**

  The location to start drawing the vector at.

- **vector**

  The vector to draw.

Keyword Arguments:

- **:thickness**

  The thickness in pixels of the start point of the vector.

Example 8-16. Draw-vector example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
(draw-vector '(0.0 0.0 0.0) '(1.0 0.0 0.0) :thickness 2 :color 'green)
```

drawn-objects [Function]

Returns an array containing the information of the objects that are drawn in the current display. The returned information gives (object color render geom-id line-type).

Example 8-17. Drawn-objects example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (drawn-objects)
;;returns () meaning no objects are drawn
> (create-model 'box-object)
;; creates a new box-object model
> (draw (the))
;; draws the box
> (drawn-objects)
;; returns an instance of the box
> (add-object (the) 'cylinder-object 'cylinder-object)
;; adds a new cylinder object
> (drawn-objects)
;; still retursn only box's instance
> (draw (the cylinder-object))
;; draws the cylinder
> (drawn-objects)
;; returns an instance of the box and the cylinder
```

get-cycled-color [Function]

This function enables the user to cycle through a list of colors to assign color values to multiple objects. It takes an index (integer) and a list of colors such as those accepted by the color property of a graphic-object and returns the desired color. The index is used to pick a color from the color-list. The acceptable values of index are integers from 0 upwards. If the index is greater than the "length of color-list - 1", then it cycles through to the beginning, and so on.

Example 8-18. Get-cycled-color example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (get-cycled-color 2 '(red green blue cyan magenta yellow))
;; returns BLUE
> (get-cycled-color 6 '(red green blue cyan magenta yellow))
;; returns RED
```

get-gray-color [Function]

Arguments: (value min max &key (gray-low 0.1) (gray-high 1.0)) This function maps a value within a range to a gray colorscale. The function returns a grayscale color for the input value as a shade of gray that corresponds to a grayscale intensity between gray-low and gray-high. It returns a color as a list of RGB values that can be passed to the color property of an object. The color is in the form of '(x y z) where 0.0 <= x <= 1.0 and x = y = z; RGB values being equal identifies a shade of gray between black (0 0 0) and white (1 1 1).

Arguments:

- **value**

  Desired value (real number) to be mapped. (min <= value <= max)

- **min**

  Real number defining the lower boundary of the value range.

- **max**

  Real number defining the upper boundary of the value range.

Keyword Arguments:

- **:gray-low**

  Value between 0 and 1 specifying the lower boundary of the gray range to be mapped to.

- **:gray-high**

  Value between 0 and 1 specifying the upper boundary of the gray range to be mapped to.

Example 8-19. Get-gray-color example

```
(get-gray-color 0.5 0.0 1.0 :gray-low 0.0 :gray-high 1.0)
;; returns (0.5 0.5 0.5)
(get-gray-color 0.0 0.1 1.0)
;; returns (0.0 0.0 0.0)
```

get-mouse-last-selection-location [Function]

This function returns the mouse location of the last selection from get-object.

get-random-color [Function]

This function returns a random color from AML's internal color list.

Keyword Arguments:

- **colors-to-exclude**

  List of colors to exclude from the functions' internal list.

Example 8-20. get-random-color example

```
(get-random-color)
;;; returns "dark green"
```

```
 (get-random-color :colors-to-exclude '(black grey))
;;; returns "orange"
```

get-ranked-color [Function]

Maps an integer i between 0 and n (n > 0) to a color on the HSV scale from. It returns a list of RGB values between 0 and 1 that can be passed to the color property of an object. (Color scale: Blue -> Cyan -> Green -> Yellow -> Red). This function sets s = 0.8 and v = 0.7 .

Arguments:

- **i**

  Integer to map

- **n**

  Maximum Integer

get-rgb-color [Function]

The get-rgb-color function returns a list of the red, green, and blue used to define the specified color. The values are in a range from 0 (no color) to 1.0 (full color).

Arguments:

- **name**

  The name of the desired color as a string or symbol.

Example 8-21. Get-rgb-color example

```
(GET-RGB-COLOR 'red)
;; returns (1.0 0.0 0.0)
(GET-RGB-COLOR 'green)
;; returns (0.0 1.0 0.0)
(GET-RGB-COLOR 'blue)
;; returns (0.0 0.0 1.0)
(GET-RGB-COLOR 'white)
;; returns (1.0 1.0 1.0)
(GET-RGB-COLOR 'black)
;; returns (0.0 0.0 0.0)
```

get-rgb-color-from-rainbow-colormap [Function]

The get-rgb-color-from-rainbow-colormap function returns a rgb color for a given value min and max to match the color shows in the color bar.

Arguments:

- **value**

  The value of the color.

- **min**

  The minimum value of the color.

- **max**

  The maximum value of the color.

get-scaled-color [Function]

This function takes a value within a range and returns a color within a "color scale" as a list of RGB values (between 0 and 1) that can be passed to the color property of an object. A color scale is defined as a continuous set of colors such that two values can be distinguished (and identified as smaller or larger) based on the colors they are mapped to.

Arguments:

- **value**

  Value (Real number) to be mapped. (min <= value <= max)

- **min**

  Real number defining the lower boundary of the value range.

- **max**

  Real number defining the upper boundary of the value range.

Keyword Arguments:

- **:compliment?**

  When nil, the function maps a value between min and max to a colorscale such that blue corresponds to the smallest value and red to the largest value. The transitions occur on the HSV color scale such that values vary as follow: Blue -> Cyan -> Green -> Yellow - > Red. If compliment? is non-nil, the values vary as follow: Red -> Magenta -> Blue.

- **:s**

  Color saturation (between 0 and 1) corresponding to the HSV color scale.

  Default Value: 0.7

- **:v**

  Color value (between 0 and 1) corresponding to the HSV color scale.

  Default Value: 0.8

get-view [Function]

Returns a list of the current view parameters of the current display in the following order: eyevector, up-vector, camera target position, and camera field. All parameters returned are a list of x,y,z components/coordinates with the exception of the camera field which is a list of the width and height (respectively) of the current view rectangle. This function does not take any arguments.

See Also: view-set, change-view-vectors, current-eye-vector, current-up-vector, set-camera-target, current-camera-target-position, set-camera-field, current-camera-field

lower-display [Function]

This method causes the current display object to be lowered or pushed to the back of the screen.

See Also: raise-display

magnify [Function]

Change the viewing magnification of the current display window by the specified factor.

Arguments:

- **factor**

  Positive number. Factors greater than 1 will reduce the viewing area making objects appear closer, and factors less than 1 will augment the viewing area making objects appear farther.

Keyword Arguments:

- **update?**

  Default t updates the graphics window.

  Default Value: t

Example 8-22. Magnify example

```
> (magnify 0.5)
;; Make the graphics viewing area half size.
> (magnify 2.0)
;; Double the graphics viewing area,.
```

map-3d-to-pixel [Function]

Converts a 3d point to the pixel coordinates of the current active display.

Arguments:

- **x**

  The x coordinate of the point.

- **y**

  The y coordinate of the point.

- **z**

  The z coordinate of the point.

Example 8-23. Map-3d-to-pixel example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (create-model 'name-generator)
;; creates a new name-generator model
> (add-object (the) 'grid-object 'grid-object)
;; adds a grid
> (draw (the))
;; draws the grid
> (map-3d-to-pixel 0.0 0.0 0.0)
;; returns the following values :
184
491
```

```
> (draw-point 184 491 :draw-type :flip)
;; draws the new point
```

mouse-pan [Function]

This function allows the graphics in the current display window to be interactively moved by moving the camera. Click and hold the left mouse button in the display window while moving the mouse to pan. Constraining pan: Hold the [shift] key down while moving to move along the x-axis only. Hold the [control] key down while moving to move along the y-axis only. Hold the [shift] and [control] keys down while moving to move along the z-axis only. Note: The axis referred to above are with respect to the view where x is horizontal, the y vertical, and z is perpendicular to the screen.

mouse-rotate [Function]

This function allows the graphics in the current display window to be interactively rotated by rotating the camera. Click and hold the left mouse button in the display window while moving the mouse. Constraining rotation: Hold the [shift] key down while moving to rotate about the x-axis only. Hold the [control] key down while moving to rotate about the y-axis only. Hold the [shift] and [control] keys down while moving to rotate about the z-axis only. Note: The axis referred to above are with respect to the screen where the x is horizontal, the y is vertical, and the z is perpendicular to the screen.

mouse-select-point-from-display [Function]

This function accepts a list of point coordinate list from which the user can interactively select in the display pane. It returns the position in the list of the selected point as well as the point coordinates in the form of multiple-values.

Arguments:

- **point-list**

  A list of points to allow the user to select from.

- **window**

  The window id of the display to allow the user to select the point from.

Example 8-24. Mouse-select-point-from-display example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (create-model 'name-generator)
;; creates a new name-generator model
> (add-object (the) 'grid-object 'grid-object)
;; adds a grid
> (draw (the))
;; draws the grid
> (zoom :all)
;; fits the grid to screen
> (mouse-select-point-from-display '((1.0 0.0 0.0) (0.0 1.0 0.0) (0.0 0.0
0.0)))
;; Move mouse around in display pane. Each point will highlight
```

mouse-zoom [Function]

Allows the user to zoom in on a section of the screen by specifying a box enclosing the region to be displayed. The box is specified by clicking on two opposite corners.

Arguments:

- **window**

  The window id of the current active display to allow the user to zoom in on the region to be displayed.

Example 8-25. Mouse-zoom example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (create-model 'box-object)
;; creates a new box-object model
> (change-view :iso)
;; changes the view to isometric
> (draw (the))
;; draws the box
> (mouse-zoom)
;;Press left mouse and drag to create zoom box.
```

output-display [Function]

This function prints or saves the contents of the current display window in a format suitable for printing or inclusion in documents. Options allow for the contents to be written to a file or sent directly to a printer in different formats.

Keyword Arguments:

- **:file-path**

  The complete path and filename of the file to be written by this function. The default is a file named "display.out" in the user's temp directory.

- **:lpr?**

  If t, the output will be sent directly to the printer. The :file-path will be ignored.

- **:back-color**

  If this is not nil, the background color of the display window will be changed to :back-color before the output is created. The default is 'white which is appropriate for sending the output to a printer in order to prevent filling the page with a black background.

  Default Value: 'white

- **:fore-color**

  Color that some objects are forced to change to before printing. Refer to the colors-to-change keyword below. This is useful for creating single-color files for devices which do not present gray-scale images well.

- **:outlined?**

  If t, the output file will have a border around the image. If nil, there will be no border.

- **:format**

  Specifies the format of the output file. The options allowed are: :ps postscript format (the default): :bmp bitmap format. :eps encapsulated postscript :hpgl hpgl plotter language :pict Macintosh(tm) pict format :tiff for TIFF image format :gif for GIF image format :jpeg for JPG image format :png for PNG format

- **:print-command**

  String representing the platform dependent print command line. This varies from computer platform to the other. This keyword is ignored when lpr? is nil.

- **:colors-to-change**

  List of color name. Defaults to '(white). Colors specified in this list are the colors that are changed to fore-color before printing. When nil, all colors are forced to change to fore-color.

- **:window-id**

  Specifies the window to be printed. It takes a native window id (integer). A native window id is a platform dependent pointer/integer that is a handle to a window (In WINDOWS it is equal to the "hwnd" handle of the window. In X Windows (UNIX/LINUX) it is the "Window" handle of a window). To get the window id of the currently active canvas, call the function "current-display-id".

  Default Value: current-display-id

- **:image-width**

  Bitmap image width

  Default Value: window width

- **:image-height**

  Bitmap image height

  Default Value: window height

- **:min-line-width**

  Minimum line-width

  Default Value: 0

- **:line-width-scale**

  Scale of line width

  Default Value: 1

- **:nColorBits**

  This keyword is only used when the output is in bitmap format. Its valid values are 0 and 4. When it is 0, it creates a bitmap with default color bits (24 bits); when it is 4, it creates a bitmap with 4-bit colors.

  Default Value: 0

position-axis [Function]

Used to position the axis satellite in the display pane.

Arguments:

- **position**

  The position to place the axis. The valid positions are :up-left, :upright, :low-left, :low-right.

raise-display [Function]

This function is used to raise (bring to the foreground) the current display.

Keyword Arguments:

- **refresh?**

  This determines whether or not to refresh the display after it is brought to the foreground. The default value causes the display to be refreshed.

  Default Value: t

refresh [Function]

Redraw the valid graphics that are currently in the display window. The graphics are not regenerated.

regen [Function]

Recreate the graphics from the model in the display. If changes are made to the model that will affect the graphics this function must be used to update the graphics.

Example 8-26. Regen example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (create-model 'box-object)
;; creates a new box-object model
> (draw (the))
;; draws the box
> (zoom :all)
;; fits to screen
> (change-value (the height) 5.0)
;; changes the value of box's height to 5.0
> (regen)
;; The graphics are regenerated and refreshed - new box is shown
> (change-value (the height) 1.0)
;; changes the height agian to 1
> (refresh)
;; Graphics disapear.
> (regen)
;; The graphics are regenerated.
```

save-view-to-file [Function]

This function saves user-defined or current graphics view parameters to a text file. The format of the text file is readable by the method retrieve-views-from-file (add href here). Retrieving views from a file is also accessible from different AML GUI forms like the graphic control toolbar.

Arguments:

- **view-name**

  Name of view, a string is expected.

Keyword Arguments:

- **:up-vector**

  A list of x,y,z components specifying the "UP" vector of the view. When nil, the "UP" vector of the current view in the active graphics display is used.

  Default Value: nil

- **:eye-vector**

  A list of x,y,z components specifying the "EYE" vector of the view. When nil, the "EYE" vector of the current view in the active graphics display is used.

  Default Value: nil

- **:camera-target-position**

  A list of x,y,z coordinates specifying the camera target position of the view. When nil, the camera target position of the current view in the active graphics display is used.

  Default Value: nil

- **:camera-field**

  A list of 2 components (width and height) specifying the camera field of the view. When nil, the camera field of the current view in the active graphics display is used.

  Default Value: nil

See Also: get-view, view-set, view-manager-class

set-camera-field [Function]

Sets the width and height of the camera field (view rectangle) in the current display.

Arguments:

- **width**

  Width of the camera field.

- **height**

  Height of the camera field.

See Also: current-camera-field

set-camera-rotation-center [Function]

This functions sets the camera rotation center point of the current display. This affects the interactive mouse rotation of the graphics. This function is only effective on platforms running OpengGL graphics.

Arguments:

- **point**

  List of x,y,z coordinates of desired rotation center.

set-camera-target [Function]

Arguments:

- **position**

  List of global x,y,z coordinates of the desired camera target position.

See Also: current-camera-target-position

set-faceting [Function]

Sets the faceting method/value for graphics rendering. A smaller faceting value represents smoother graphics display.

Arguments:

- **faceting-value**

  Can be 'smooth or 'rough. This argument can also be a real value. As a reference, a value of 0.25 corresponds to smooth faceting, while a value of 0.8 corresponds to rough faceting. The lower the value is, the smoother the graphics rendering. This value must be positive.

set-interactive-mode [Function]

This function allows to specify a graphics interactive mode for the current display. A graphics display (or canvas) window can be in translate mode or rotate mode. If an interactive mode is set, dragging the left button in the current display window performs the action corresponding to that mode. This allows the rotation/translate of the objects displayed without having to use the graphic palette's buttons.

Arguments:

- **mode**

  Can take a value of 1 for translate mode, 2 for rotate mode, 0 to disable any exsiting interactive mode and interactive mode 10. When the interactive mode is 10, the following events happened within the graphics window are posted to the AML thread, the user can use register-message-handler function to define callbacks for these events: SERVER_MOUSEMOVE 11000 SERVER_LBUTTONDOWN 11001 SERVER_LBUTTONUP 11002 SERVER_LBUTTONDBLCLK 11003 SERVER_RBUTTONDOWN 11004 SERVER_RBUTTONUP 11005 SERVER_RBUTTONDBLCLK 11006 SERVER_MBUTTONDOWN 11007 SERVER_MBUTTONUP 11008 SERVER_MBUTTONDBLCLK 11009 SERVER_KEYDOWN 11010 SERVER_KEYUP 11011

set-shading [Function]

Sets the shading method used for shaded graphics rendering. The methods available are flat, facet, gouraud and phong.

Arguments:

- **method**

  Shading method desired. Values allowed are : 'flat , 'facet, 'gouraud, 'phong.

ts-vil-clipboard-copy [Function]

This function copies the text-string to the clipboard. The (aml) function must be called before using this funtion.

Arguments:

- **text-string**

  The string to be copied to the clipboard.

Keyword Arguments:

- **socket**

  Default Value: *default-socket*

ts-vil-clipboard-paste [Function]

This function pastes the content of the clipboard to the widget which can be found in the 'tsi-entity property of a widget. The text is placed at the current caret position of the widget.

Arguments:

- **widget**

  The widget to copy the clipboard content to.

Keyword Arguments:

- **socket**

  Default Value: *default-socket*

view-set [Function]

Sets the view parameters of the current display including up-vector, eye-vector, camera target position and camera field.

Arguments:

- **up-vector**

  List of the X,Y,Z components of the desired eye-vector. Refer to change-view-vectors.

- **eye-vector**

  List of the X,Y,Z components of the desired up-vector. Refer to change-view-vectors.

- **target-position**

  List of the x,y,z coordinates of the camera target position. Refer to set-camera-target.

- **camera-field**

  List of the width and height of the camera field (view rectangle). Refer to set-camera-field.

See Also: get-view, change-view-vectors, current-eye-vector, current-up-vector, set-camera-target, current-camera-target-position, set-camera-field, current-camera-field.

zoom [Function]

Changes the camera field (view rectangle) of the current display (active display). Supply :all, :window, :in, or :out and the displayed region will change. The :window option allows the selection of a rectangular area on the screen. The :in and :out options operate at a 25% increase and decrease respectively. The :all option conforms to all graphics currently being displayed; it fits the camera field to make all drawn objects visible. The function can also be passed a list of objects, and the viewing area will fit to the drawn objects specified.

Arguments:

- **all/window/in/out/object**

  The type of zoom to be performed.When this argument is a list of objects, the viewing area will fit to the drawn objects specified.

Optional Arguments:

- **update?**

  When non-nil, refreshes the graphics after changing the camera field.

  Default Value: nil

- **fitview?**

  When fitview? is t, it fits the drawn objects to the current view.

  Default Value: nil

Example 8-27. Zoom example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (create-model 'box-object)
;; creates a new box-object model
> (draw (the))
;; draws the box
> (zoom :all)
;; Graphics Fit in Screen
> (zoom :in)
;; Graphics are increased in size.
> (zoom :out)
;; Graphics are decreased in size.
> (zoom :window)
;; Equivalent to mouse-zoom.
> (zoom (list (the box1) (the cylinder1))
;; Fits the viewing area to the box1 and the cylinder1 if drawn.
```

zoom-box [Function]

Changes the camera field and target position (view rectangle) of the current active display according to two point coordinate lists in model space. The camera target position is set to the middle point of the two points provided. The height and width of the camera field are set to the distance between the two points provided. This function does not affect the eye-vector and up-vector of the view.

Arguments:

- **min-point**

  Coordinate list of first point.

- **max-point**

  Coordinate list of second point.

Keyword Arguments:

- **:update?**

  When non-nil, refreshes the graphics after changing the camera field.

  Default Value: t

Example 8-28. Zoom-box example

```
;; If we are in front view and we need to make the camera field and target
;; match exactly a non oriented cube of size 1 and centered at '(0 0 0)
> (zoom-box '(-0.5 0.0 0.0) '(0.5 0.0 0.0))
;; returns -1
```

Grid Classes

Classes and Methods

grid-class [class]

This is the common class for all of updated/new grid classes. This class should not be instantiated by itself. Eventually all of the functionality available in the grid-object will be available for the classes inheriting from grid-class.

Inheritance: graphic-object

Properties:

- **snap?**

  When t, this property causes the mouse to snap to the grid points when the user is interactively selecting a point from the grid. When nil, the user can select any point.

  Default Formula: nil

get-point-coordinates-from-grid-object [Method]

Defined on Classes:

grid-class

Allows the interactive selection of a point (in the current graphics display) based on a grid-class instance properties. The method takes a grid-class instance argument. It returns the list of x,y,z coordinates of the point selected with a left-click. The method returns nil when it is aborted with a right-click.

get-points-coordinates-list-from-grid-object [Method]

Defined on Classes:

grid-class

Allows the interactive selection of a list of points (in the current graphics display) based on a grid-class instance properties. The method takes a grid-class instance argument. It returns a list of lists of the x,y,z coordinates of the points selected with a left mouse click. The selection of points is aborted with a right mouse click. It returns nil if no point was selected.

grid-from-points-coordinates-class [class]

This class constructs a grid from the points provided in the points-coordinates-list property.

Inheritance: grid-class

Properties:

- **points-coordinates-list**

  This is a list of X,Y,Z coordinate-lists which the grid will be constructed from.

  Default Formula: nil

grid-on-plane-class [class]

This class constructs a grid that lies on the global XY plane. It can be oriented using standard AML orientation capabilities.

Inheritance: grid-class

Properties:

- **center-point**

  List of the x,y.z coordinates of the grid center point.

  Default Formula: '(0 0 0)

- **grid-spacing**

  Spacing between the grid points.

  Default Formula: 1

- **x-range**

  Range (in global space) in the x direction.

  Default Formula: 20

- **y-range**

  Range (in global space) in the y direction.

  Default Formula: 20

grid-on-surface-class [class]

This class constructs a grid that lies on any AML surface (dimension = 2) graphic object.

Inheritance: grid-class

Properties:

- **source-object**

  Surface to generate the grid on. It should be a graphic-object instance with its dimension = 2.

- **number-of-points-u**

  Number of grid points to generate on the surface in the U direction.

  Default Formula: 20

- **number-of-points-v**

  Number of grid points to generate on the surface in the V direction.

  Default Formula: 20

grid-object [class]

The grid-object is used to create and display a grid. Unless the user is interested in any of the several interactive methods defined on grid-object (see below), it is recommended to use grid-on-plane-class instead of grid-object.

Inheritance: primitive-class

Properties:

- **draw-type**

  This defines the type of grid to be drawn. The valid options are: 'point, 'line, 'line-dashed, 'line-dotted, 'line-dash-dot, 'line-dash-dbldot, 'line-dash-tridot, 'line-long-dash, and 'face.

  Default Formula: 'point

- **snap**

  This property controls whether or not to snap. The default value, nil, does not snap.

  Default Formula: nil

- **spacing**

  The spacing between grid points.

  Default Formula: 1.0

- **snap-spacing**

  The spacing between snap points.

  Default Formula: 1.0

- **start-point**

  The location to start drawing the grid at.

  Default Formula: '(0.0 0.0 0.0)

- **x-dir**

  The x-axis of the grid.

  Default Formula: '(1.0 0.0 0.0)

- **y-dir**

The y-axis of the grid.

Default Formula: '(0.0 1.0 0.0)

- **z-dir**

  The z-axis of the grid.

  Default Formula: '(0.0 0.0 1.0)

- **x-limit**

  The distance along the x-axis to draw the grid.

  Default Formula: 10

- **y-limit**

  The distance along the y-axis to draw the grid.

  Default Formula: 10

- **z-limit**

  The distance along the z-axis to draw the grid.

  Default Formula: 0

2-point-circle [Method]

Defined on Classes:

grid-object

This method is used to interactively obtain two points to specify a circle. It will return two lists in the form of multiple values. The first list will contain the first point and the method for obtaining that point and the second list will return the second point and the method for obtaining that point.

Arguments:

- **instance**

  An instance of a grid-object.

Optional Arguments:

- **point-1**

  The first point of the circle. If the point is not specified, the user will interactively select it from the display.

2-point-line [Method]

Defined on Classes:

grid-object

This method is used to interactively obtain the endpoints of a line from the display pane. It will return two lists in the form of multiple-values. The first list will contain the first point and the method used for obtaining that point. The second list will contain the second point and the method for obtaining that point.

Arguments:

- **instance**

  An instance of a grid-object.

Optional Arguments:

- **point-1**

  The first point of the line. If the line is not specified, the user will interactively select the point from the display.

add-a-arc-object [Method]

Defined on Classes:

grid-object

Interactive method for grid-object used to add arcs interactively. The return value for the method is the new instance.

Arguments:

- **grid-instance**

  An instance of a grid-object the new point will be initially attached to for placement.

Keyword Arguments:

- **:add-to**

  This is the object that will be the parent for the object that is to be added. If :add-to is not supplied and a property named construction exists the value of that property will be used as the new location. The error condition is to add the new object to the grid object instance.

- **:method**

  method keyword can take 4 values: 1. '3-point - Define the arc by selecting: Start Point, Point on Arc, and End-point. 2. 'start-center-angle - Define the arc by selecting: Start Point, Center Point, Angle, and Segment. 3. 'center-radius-angles - Define the arc by selecting: Center point, Radius, Start Angle, End Angle, and Arc Segment. 4. 'center-radius-points - Define the arc by selecting: Center point, Start Point, End Point, and Arc Segment.

Note : for example refer to add-a-vector-object method.

add-a-circle-object [Method]

Defined on Classes:

grid-object

Interactive method for grid-object used to add circle interactively. The return value for the method is the new instance.

Arguments:

- **grid-instance**

  An instance of a grid-object the new point will be initially attached to for placement.

Keyword Arguments:

- **:add-to**

This is the object that will be the parent for the object that is to be added. If :add-to is not supplied and a property named construction exists the value of that property will be used as the new location. The error condition is to add the new object to the grid object instance.

- **:method**

  method keyword can take 3 values: 1. '2-point - Define the circle by interactively selecting to diameter points 2.'center-radius - Define the circle by interactively selecting the center of an arc then the center of the circle. 3.'center-point - Define the circle by interactively selecting the center of the circle then a point defining the radius.

  Default Value: 'center-redius

Note : for example refer to add-a-vector-object method.

add-a-line-object [Method]

Defined on Classes:

grid-object

Interactive method for grid-object used to add lines interactively. The return value for the method is the new instance.

Arguments:

- **grid-instance**

  An instance of a grid-object the new point will be initially attached to for placement.

Keyword Arguments:

- **:add-to**

  This is the object that will be the parent for the object that is to be added. If :add-to is not supplied and a property named construction exists the value of that property will be used as the new location. The error condition is to add the new object to the grid object instance.

- **:method**

  The method keyword can take 3 values : 1. 'point-point - Define the line by interactively selecting 2 points. 2. 'point-vector - Define the line by interactively selecting a point, a direction and a distance. 3. 'tangent-to - Define the line by interactively selecting a point and an object the line should be tangent to.

  Default Value: 'point-point

- **:from-point**

  Allows the user to give the first point of the line. It is a list of the 3 coordinates of the point. If ":from-point" is nil, the user is required to interactively select the first point.

  Default Value: nil

Note : for example refer to add-a-vector-object method.

add-a-point-object [Method]

Defined on Classes:

grid-object

Interactive method for grid-object used to add points interactively. The return value for the method is the new instance.

Arguments:

- **grid-instance**

  An instance of a grid-object the new point will be initially attached to for placement.

Keyword Arguments:

- **:add-to**

  This is the object that will be the parent for the object that is to be added. If :add-to is not supplied and a property named construction exists the value of that property will be used as the new location. The error condition is to add the new object to the grid object instance.

Note : for example refer to add-a-vector-object method.

add-a-vector-object [Method]

Defined on Classes:

grid-object

This method adds a vector at the specified location.

Arguments:

- **instance**

  An object of type grid-object to interactively sketch the vector on.

Keyword Arguments:

- **:add-to**

  The object to add the vector-object to. If the add-to is nil, the vector will be added to the instance.

Example 8-29. Add-a-vector example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (create-model 'name-generator)
;; creates a new name-generator model
> (add-object (the) 'grid-object 'grid-object)
;; adds a grid instance to the name-generator
> (add-a-arc-object !grid-object)
;; the user has to specify with the mouse, the start point, middle point
;; and the end point of the arc, which will be drawn after that
> (add-a-circle-object !grid-object)
;; by first click on the screen, user selects the center of circle
;; with second, the user decides on the radius of the circle which will be
drawn
> (add-a-line-object !grid-object)
;; by pointing on the screen user can select the start and
;; end points for the line, which will be drawn after the points are set
> (add-a-point-object !grid-object)
;; by using left mouse button points can be drawn
```

```
> (add-a-vector-object (the grid-object))
;; by pointing on the screen user can select the start and
;; end points for the vector, which will be drawn after the points are set
```

center-radius-angle-arc [Method]

Defined on Classes:

grid-object

This method is used to interactively define an arc by sketching it on the screen. The user will pick the center, radius, a point to define the start vector, a point to define the end vector, and the segment of the arc to keep.

Arguments:

- **instance**

  An instance of type grid-object to add the arc to.

- **point1**

  The center of the arc.

- **radius**

  The radius of the arc.

- **point2**

  A point to define the start vector of the arc.

center-radius-circle [Method]

Defined on Classes:

grid-object

This method is used to interactively define a circle by selecting the center and the radius of the circle.

Arguments:

- **instance**

  An instance of type grid-object to add the circle to.

- **point1**

  The center of the circle.

drag-3-point-arc-on-grid [Method]

Defined on Classes:

grid-object

Used to display a three point arc on the screen while the user picks the third point. Returns the mouse button and the point selected in the form of multiple values.

Arguments:

- **instance**

An instance of a grid-object to drag the arc on

- **point1**

  the starting point for the arc.

- **point2**

  An arbitrary second point on the arc.

Keyword Arguments:

- **:values?**

  If t, the keyword will return a list of values - the initial return value for mouse button and the coordinates of the point selected by the user.

  Default Value: nil

Example 8-30. Drag-3-point-arc-on-grid example

```
;; Assume the AML graphics server has been started (see aml)
;; and an active graphics display exist (see the method activate-display).
> (create-model 'name-generator)
;; creates a new name-generator model
> (add-object (the) 'grid-object 'grid-object)
;; adds a grid instance to the name-generator
> (draw (the))
;; draws the grid
> (zoom :all)
;; fits to screen
> (drag-3-point-arc-on-grid (the grid-object) '(0.0 0.0 0.0) '(1.0 0.0 0.0))
;; An arc drawn/dragged as the mouse cursor moves. The user has to select a
point to define the arc.
;; The point can be selected anywhere. Draws the circle if left mouse button
was pushed and returns 1
;; if right button was selected, doesn't draw the circle, returns 0
> (change-value (the snap (:from (the grid-object))) t)
;; Now the user can only select points that are part of the grid
> (drag-3-point-arc-on-grid (the grid-object) '(0.0 0.0 0.0) '(1.0 0.0 0.0)
:values? t)
;; User can only select a point that is part of the grid. Because the
"values?" keyword is t, the method
;; will also return the point coordinates as the second element of the list
returned.
```

drag-box-on-grid [Method]

Defined on Classes:

grid-object

Used to display a rectangle on the screen while the user picks the second point. Returns the mouse button and the point selected in the form of multiple values.

Arguments:

- **instance**

  An instance of a grid-object to drag the arc on

- **point1**

  The starting point for the box.

Keyword Arguments:

- **:values?**

  If t, the keyword will return a list of values - the initial return value for mouse button and the coordinates of the point selected by the user.

  Default Value: nil

Example 8-31. Drag-box-on-grid example

```
;; Assume the same setup as the example above in the drag-3-point-arc-on-grid
example.
> (drag-box-on-grid (the grid-object) '(0.0 0.0 0.0))
;; A box drawn/dragged as the mouse cursor moves. The user has to select a
point to define the rectangle.
;; The point can be selected anywhere. Draws the rectanlge and if left mouse
button was pushed and returns 1
;; if right button was selected, doesn't draw the circle, returns 0
> (change-value (the snap (:from (the grid-object))) t)
;; Now the user can only select points that are part of the grid
> (drag-box-on-grid (the grid-object) '(0.0 0.0 0.0) :values? t)
;; User can only select a point that is part of the grid. Because the
"values?" keyword is t, the method
;; will also return the point coordinates as the second element of the list
returned.
```

drag-circle-on-grid [Method]

Defined on Classes:

grid-object

Used to display a circle on the screen to select a point on the circumference. Returns the mouse button pressed and the point select in the form of multiple values.

Arguments:

- **instance**

  An instance of type grid-object.

- **center-point**

  The center point of the circle.

Keyword Arguments:

- **:values?**

If t, the keyword will return a list of values - the initial return value for mouse button and the coordinates of the point selected by the user.

Default Value: nil

Example 8-32. Drag-circle-on-grid example

```
;; Assume the same setup as the example above in the drag-3-point-arc-on-grid
example.
> (drag-circle-on-grid (the grid-object) '(5.0 5.0 0.0))
;; A circle drawn/dragged as the mouse cursor moves. The user has to select a
point to define the circle.
;; The point can be selected anywhere. Draws the circle and if left mouse
button was pushed and returns 1
;; if right button was selected, doesn't draw the circle, returns 0
> (change-value (the snap (:from (the grid-object))) t)
;; Now the user can only select points that are part of the grid
> (drag-circle-on-grid (the grid-object) '(5.0 5.0 0.0) :values? t)
;; User can only select a point that is part of the grid. Because the
"values?" keyword is t, the
;; method will also return the point coordinates as the second element of the
list returned.
```

drag-circle-on-grid-by-center [Method]

Defined on Classes:

grid-object

Allows the user to drag a circle defined by a point on the circumference with the mouse location being the center of the circle.

Arguments:

- **instance**

  An instance of a grid-object.

- **point1**

  A point on the chord of the circle.

Keyword Arguments:

- **:values?**

  If t, the keyword will return a list of values - the initial return value for mouse button and the coordinates of the point selected by the user.

  Default Value: nil

Example 8-33. Drag-circle-on-grid-by-center example

```
;; Assume the same setup as the example above in the drag-3-point-arc-on-grid
example.
> (drag-circle-on-grid-by-center (the grid-object) '(5.0 5.0 0.0))
```

```
;; A circle drawn/dragged as the mouse cursor moves. The user has to select a
center to define the circle.
;; The center can be selected anywhere. Draws the circle and if left mouse
button was pushed and returns 1
;; if right button was selected, doesn't draw the circle, returns 0
> (change-value (the snap (:from (the grid-object))) t)
;; Now the user can only select points that are part of the grid
> (drag-circle-on-grid-by-center (the grid-object) '(5.0 5.0 0.0) :values? t)
;; User can only select a point that is part of the grid. Because the
"values?" keyword is t, the
;; method will also return the point coordinates as the second element of the
list returned.
```

profile-3-point-arc [Method]

Defined on Classes:

grid-object

Used to sketch a 3 point arc in the interface.

Arguments:

- **instance**

  A grid-object.

Optional Arguments:

- **point1**

  The start point of the arc. If nil, the user will be asked to select one. If supplied, it must be a list of the point and the method for getting the point.

  Default Value: nil

- **point2**

  A point on the arc. If nil, the user will be asked to specify one. If supplied, it must be a list of the point and the method for getting the point.

  Default Value: nil

Keyword Arguments:

- **:values?**

  If t, the keyword will return a list of values - the initial return value for mouse button and the coordinates of the point selected by the user.

  Default Value: nil

Example 8-34. Profile-3-point-arc example

```
;; Assume the same setup as the example above in the drag-3-point-arc-on-grid
example.
> (profile-3-point-arc (the grid-object))
;; Allows user to select two points and to drag/draw the circle with the
mouse
;; returns the coordinates of the two points selected
```

```
> (profile-3-point-arc (the grid-object) '((2.0 2.0 0.0) (2.0 2.0 0.0)))
;; The two point are specified, so the user has to drag/draw the mouse to
finish the circle.
```

Other Classes

Classes and Methods

bitmap-class [class]

This class retrieves a bitmap from a file (.bmp) and displays the image in the graphics window when the method draw is called on it. This class only supports uncompressed bitmap file format (.bmp). The start-point must be within the graphic window's viewing area, the bitmap will not show if the start-point falls outside the window. The graphics' display "fit" functionality does not work on this class.

Inheritance: object

Properties:

- **file-name**

  The bitmap file.

  Default Formula: nil

- **start-point**

  The bitmap low left corner coordinates.

  Default Formula: nil

- **bitmap-width**

  The image width (in pixels) when drawn in the window. When nil, the width as defined in the bitmap file is used.

  Default Formula: nil

- **bitmap-height**

  The image height (in pixels) when drawn in the window. When nil, the height as defined in the bitmap file is used.

  Default Formula: nil

surface-isolines-object [class]

This object takes a surface and creates geometry that renders it in isoline mode. This class exists because the AML PARASOLID-based solid modeler does not accept 'isoline as a valid value for the render property of graphic objects.

Inheritance: geom-object

Properties:

- **nu**

  Number of lines in the u param direction.

Default Formula: 10

- **nv**

  Number of lines in the v param direction.

  Default Formula: 10

- **surface-object**

  Object instance of a surface geometry.

  Default Formula: nil

view-manager-class [class]

This class allows the management of graphics view parameters during an AML session. The user can instantiate it anywhere in the model tree.

Inheritance: object

Properties:

- **This class defines three internal properties: views, view-names, and view-menu. The user should interact with this class only via the methods defined below.**

add-view [Method]

Defined on Classes:

view-manager-class

This method saves a view record to a view manager instance.

Arguments:

- **view-manager**

  Instance of view-manager-class.

Keyword Arguments:

- **:name**

  Optional keyword specifying a name (string) for the view to add. When nil, it will cause the method to prompt the user for a view name.

  Default Value: nil

- **:view**

  Optional keyword specifying the parameters of the view to add. This list should be in the same format accepted by the function view-set. When nil, it will use the current view parameters of the current active display.

  Default Value: nil

- **:replace?**

  Optional argument. When non-nil and the view name provided already exists under view-manager, the existing view parameters are overwritten.

  Default Value: nil

change-view-from-manager [Method]

Defined on Classes:

view-manager-class

Sets the view of the current active display to the view parameters stored in a view-managerclass instance given the view name. A call to the method is ignored if the view name is not found under the view manager provided.

Arguments:

- **view-manager**

  Instance of view-manager-class.

- **view-name**

  View name (string) as stored in view-manager.

remove-view [Method]

Defined on Classes:

view-manager-class

Removes a view record from a view manager.

Arguments:

- **view-manager**

  Instance of view-manager-class.

- **view-name**

  View name (string) to remove.

select-view-from-manager [Method]

Defined on Classes:

view-manager-class

Sets the view of the current active display by allowing the user to select a view from a menu of available views stored in the view manager provided.

Arguments:

- **view-manager**

  Instance of view-manager-class.

Chapter 9. Functions

Mathematical Functions

Scalar Functions

+ [Function]

Returns the result of adding numbers together.

Arguments:

- **numbers**

  Numbers to be added. There is no limit to the quantity of numbers which can be added together.

Example 9-1. Addition example

```
(+ 1 2)
;; returns 3
(+ 1 2 3 4 5.0)
;; returns 15.0
(+ (+ 3 4) (+ 1 -3))
;; returns 5
```

1+ [Function]

Increments a number by 1. This function is equivalent to (+ 1 number).

Arguments:

- **number**

  Any expression that returns a number.

Example 9-2. 1+ example

```
(1+ 7)
;; returns 8
(1+ (+ 2 3 4 5.0)
;; returns 15.0
```

- [Function]

The - function performs the subtraction of numbers. The order of operation is from left to right. If the calculation involves only integers then an integer will be returned.

Arguments:

- **numbers**

Numbers to be subtracted. There is no limit to the quantity of numbers which can be subtracted.

Example 9-3. Substraction example

```
(- 5 4)
;; returns 1
(- (+ 3.0 6) (+ 10 2))
;; returns -3.0
```

1- [Function]

Decrements a number by 1. This function is equivalent to (- 1 number).

Arguments:

- **number**

  Any expression that returns a number.

Example 9-4. 1- example

```
(1-  4)
;; returns 3
(1- (+ 3.0 1) (+ 10 2))
;; returns 15.0
```

* [Function]

The * function performs the multiplication of numbers. The order of operation is from left to right. If the calculation involves only integers then an integer will be returned.

Arguments:

- **numbers**

  Numbers to be multiplied. There is no limit to the quantity of numbers which can be multiplied.

Example 9-5. Multiply example

```
(* 2 4)
;; returns 8
(* (+ 2 4) (- 3 5))
;; returns -12
```

/ [Function]

The / function performs the division of numbers. The order of operation is from left to right.

Arguments:

- **numbers**

  Numbers to be divided. There is no limit to the quantity of numbers which can be divided.

Example 9-6. Division example

```
(/ 3 4)
;; returns 3/4
(/ (/ 1 2.0) (/ 4 3))
;; returns 0.375
```

abs [Function]

The abs function returns the absolute value for a given value.

Arguments:

- **number**

  Any expression that returns a number.

Example 9-7. Absolute value example

```
> (abs -4)
;; returns 4
> (abs (- 4 8))
;; returns 4
> (abs (- 8 4))
;; returns 4
```

average [Function]

The average function returns a floating number that is the average of the given numbers.

Arguments:

- **numbers**

  May be integers and/or floating point numbers.

Example 9-8. Average example

```
> (average 2 4 6 8)
;; returns 5.0f0
> (average (twice (+ 5 10)) (half (- 40 25)))
;; returns 18.75f0
```

ceiling and fceiling [Function]

The ceiling and fceiling functions round their results to the nearest integer in the positive infinity direction. If an optional divisor is supplied the division is performed and the result is rounded in the positive direction. Ceiling returns an integer and fceiling returns a floating point number.

Arguments:

- **number**

Any expression that returns a number.

Optional Arguments:

- **divisor**

  Any optional expression that returns a number.

Example 9-9. Ceiling example

```
> (ceiling 4.4)
;; returns 5
> (fceiling 4.4)
;; returns 5.0
> (ceiling -5 2)
;; returns -2
> (fceiling -5 2)
;; returns -2.0
```

See Also: floor, round, truncate.

degrees-to-radians [Function]

Converts degrees to radians.

Arguments:

- **angle**

  An angle in degrees

Example 9-10. Degrees-to-radians example

```
> (degrees-to-radians 180.0)
;; returns 3.141592653589793
> (degrees-to-radians 90.0)
;; returns 1.5707963267948966
```

See Also: radians-to-degrees

double-float [Function]

Converts a number to a double float type.

Arguments:

- **number**

  A number to convert to double float.

Example 9-11. Double-float example

```
> (double-float (/ 1 3))
;; returns 0.3333333333333333
```

expt [Function]

The expt function raises a number to a given power. If the calculation involves only integers and the result has an exact solution, an integer will be returned.

Arguments:

- **base-number**

  Any expression that returns a number to be raised to the power of power-number.

- **power-number**

  Any expression that returns a number to be used the exponent.

Example 9-12. Exponent(power) example

```
> (expt 2 3)
;; returns 8
> (expt (+ 2 3) (/ 3 5))
;; returns 2.6265278f0
```

exp [Function]

The exp function returns e (the base of the natural logarithms) raised to the power number.

Arguments:

- **number**

  The exponent for e.

Example 9-13. Exp(e) example

```
> (exp 2.0)
;; returns 7.38905609893065
```

float [Function]

This function converts a number into a single float type.

Arguments:

- **number**

  The number to convert to a single float type.

Example 9-14. Float example

```
>(float (/ 1 3))
;; returns 0.33333334f0
```

floor and ffloor [Function]

The floor and ffloor functions round their results to the nearest integer in the negative inifinity direction. If an optional divisor is supplied the division is performed and the result is rounded in the negative direction. Floor returns an integer and fflour returns a floating point number.

Arguments:

- **number**

  Any expression that returns a number.

Optional Arguments:

- **divisor**

  Optional, any expression that returns a number.

Example 9-15. Floor and ffloor example

```
> (floor 4.4)
;; returns 4
> (ffloor 4.4)
;; returns 4.0
> (floor -5 2)
;; returns -3
> (ffloor -5 2)
;; returns -3.0
```

See Also: ceiling, round, truncate.

half [Function]

The half function returns half of the given number.

Arguments:

- **number**

  May be a integer or floating point number.

Example 9-16. Half example

```
> (half 5.0)
;; returns 2.5
> (half (+ 5 4))
;; returns 4.5
```

See Also: twice

insure-between [Function]

Given a value, this function makes sure that the number passed falls in the range of the min and max values. If the number is higher than the range, the max value will be returned. If the number is bellow

the range, the min value will be returned. If the number is in the range, the number itself will be returned.

Arguments:

- **number**

  The number to check.

- **min**

  The lower limit of the range.

- **max**

  The upper limit of the range.

Example 9-17. Insure-between example

```
> (insure-between 0.5 1.0 10.0)
;; returns 1.0
> (insure-between 5 (- 5 4) (* 2 5))
;; returns 5
```

log [Function]

The log function returns the logarithm of a given number in a given base.

Arguments:

- **number**

  Any expression that returns a number.

Optional Arguments:

- **base**

  Any optional expression that returns a number to determine what base to use for the logarithm. The default is to return the natural logarithm (base e).

Example 9-18. Log example

```
> (log 100)
;; returns 4.6051702
> (log 100 10)
;; returns 2.0
> (log 8 2)
;; returns 3.0
```

max [Function]

The max function returns the number with the highest value from a given group of numbers. If the values are in a list the apply function can easily return the greatest value.

Arguments:

- **numbers**

  Any quantity of numbers to be searched for the greatest value.

Example 9-19. Max example

```
> (max 4 2 9 5 3)
;; returns 9
> (apply 'max (list 4 2 9 5 3))
;; returns 9
```

See Also: min

min [Function]

The min function returns the smallest number from a given group of numbers. If the values are in a list the apply function can easily return the lowest value.

Arguments:

- **numbers**

  Any quantity of numbers to be searched for the smallest value.

Example 9-20. Min example

```
> (min 4 2 9 5 3)
;; returns 2
> (apply 'min (list 4 2 9 5 3))
;; returns 2
```

See Also: max

mod [Function]

The mod function performs modulus arithmetic on a number and its divisor.

Arguments:

- **number**

  Any expression that returns a number.

- **divisor**

  Any expression that returns a number.

Example 9-21. Mod example

```
> (mod 10 3)
;; returns 1
> (mod -10 3)
;; returns 2
```

See Also: rem

random [Function]

The random function returns a random number between 0 and the given number. The random number generated is of the same type as the number provided, i.e. if the number given is an integer, the random number generated is an integer, if it is a real number then the random number is real.

Arguments:

- **number**

  Any expression that returns a number.

Example 9-22. Random example

```
> (random (+ 1 3.0 5 2.3))
;; returns 1.7567125461592457
> (random 4.50)
;; returns 3.2957243568224124
> (random 100)
;; returns 67
```

radians-to-degrees [Function]

Converts radian measurements to degrees.

Arguments:

- **angle**

  An angle in radians

Example 9-23. Radians-to-degrees example

```
> (radians-to-degrees pi)
;; returns 180.0
> (radians-to-degrees (/ pi 2.0))
;; retunrs 90.0
> (radians-to-degrees (/ pi 3.0))
;; returns 59.99999999999999
```

See Also: degrees-to-radians

rem [Function]

The rem function calculates the remainder for a number and its divisor.

Arguments:

- **number**

  Any expression that returns a number.

- **divisor**

Any expression that returns a number.

Example 9-24. Rem example

```
> (rem 10 3)
;; returns 1
> (rem -10.5 3)
;; returns -1.5
```

See Also: mod, floor

roughly-equal [Function]

The roughly-equal function returns a t or nil value to indicate if two supplied numbers are within a given tolerance.

Arguments:

- **number1**

  Any number

- **number2**

  Any number

Optional Arguments:

- **tolerance**

  Optional allowable deviation.

  Default Value: 0.00001

Example 9-25. Roughly-equal example

```
> (roughly-equal 5.1 5.2)
;; returns NIL
> (roughly-equal 5.1 5.2 0.125)
;; returns t
```

round and fround [Function]

The round and fround functions round their results to the nearest integer. If an optional divisor is supplied the division is performed and the result is rounded to the nearest integer. Round returns an integer and fround returns a floating point number.

Arguments:

- **number**

  Any expression that returns a number.

Optional Arguments:

- **divisor**

Any optional expression that returns a number.

Example 9-26. Round and fround example

```
> (round 4.4)
;; returns 4
> (round 4.50)
;; returns 4
> (fround 4.50)
;; returns 4.0
> (round 4.51)
;; returns 5
> (fround 4.51)
;; returns 5.0
> (round 5 2)
;; returns 2
```

See Also: floor, ceiling, round-real, truncate

round-real [Function]

The round-real function will return a floating point number that is the result of rounding the given number to the specified number of decimal places.

Arguments:

- **number**

  Number to be rounded.

- **decimal-places**

  Should be an integer value to indicate the number of digits which should be retained after the decimal point.

Example 9-27. Round-real example

```
> (round real 5.33333 2)
;; returns 5.33f0
> (round-real 5.555 1)
;; returns 5.6f0
```

signum [Function]

The signum function returns a -1 when the value is less than zero, 0 when it equals zero, and 1 when it is greater than zero.

Arguments:

- **number**

  Any expression that returns a number.

Example 9-28. Signum example

```
> (signum 4)
1
> (signum (- 4 8))
-1
```

sqrt [Function]

The sqrt function returns the square root of a given number. If the number is negative the result will not be a real number, so it will return NaN - not a number.

Arguments:

- **number**

  Any expression that returns a number.

Example 9-29. Sqrt example

```
> (sqrt 9)
;; returns 3.0
> (sqrt -9)
;; returns NaN
```

truncate and ftruncate [Function]

The truncate and ftruncate functions round their results to the nearest integer in the direction of zero. If an optional divisor is supplied the division is performed and the result is rounded toward zero. Truncate returns an integer and ftruncate returns a floating point number.

Arguments:

- **number**

  Any expression that returns a number.

Optional Arguments:

- **divisor**

  Any optional expression that returns a number.

Example 9-30. Truncate example

```
> (truncate 4.4)
;; returns 4
> (ftruncate 4.4)
;; returns 4.0
> (truncate 5 2)
;; returns 2
> (ftruncate -5 2)
;; returns -2.0
```

See Also: ceiling, floor, round

twice [Function]

The twice function returns a floating number that is twice the given number.

Arguments:

- **number**

  May be a integer or floating point number.

Example 9-31. Twice example

```
> (twice (+ 5 10))
;; returns 30.0
```

See Also: half

Trigonometric Functions

acos [Function]

Returns the arc cosine of value in radians.

Arguments:

- **value**

  A floating point number.

Example 9-32. Acos example

```
> (acos 0.8)
;; returns: 0.6435011087932844
> (acos 0)
;; returns: 1.5707963267948966
```

See Also: acosd

acosd [Function]

Function to return the angle in degrees for the arc cosine of the specified value.

Arguments:

- **value**

  A floating point number.

Example 9-33. Acosd example

```
> (acosd 0)
;; returns: 90.0
```

```
> (acosd 0.8)
;; returns: 36.86989764584402
```

See Also: acosd

asin [Function]

Returns the arc sine of value in radians.

Arguments:

- **value**

  Any number.

Example 9-34. Asin example

```
> (asin 0.5)
;; returns: 0.5235987755982989
> (asin 0)
;; returns: 0.0
```

See Also: asind

asind [Function]

Returns the arc sine of value in degrees.

Arguments:

- **value**

  Any number.

Example 9-35. Asind example

```
> (asind 0.5)
;; returns: 30.000000000000004
> (asind 0)
;; returns: 0.0
```

See Also: asind

atan [Function]

Returns the arc tangent of value in radians. Returns the arc tangent of value / denominator in radians if denominator is supplied. Handles the case of a zero denominator.

Arguments:

- **value**

  Any number.

Optional Arguments:

- **denominator**

  Optional denominator for value.

  Default Value: 1.0

Example 9-36. Atan example

```
> (atan 1.0)
;; returns: 0.7853981633974483
> (atan 0.0 0.0)
;; returns: 0.0
```

See Also: atand

atand [Function]

Returns the arc tangent of value in degrees. Returns the arc tangent of value / denominator in degrees if denominator is supplied. Handles the case of a zero denominator.

Arguments:

- **value**

  Any number.

Optional Arguments:

- **denominator**

  Optional denominator for value.

  Default Value: 1.0

Example 9-37. Atand example

```
> (atand 0.0)
;; returns: 0.0
> (atand 1.0)
;; returns: 45.0
> (atand 0.0 0.0)
;; returns: 0.0
```

See Also: atan

cos [Function]

Returns the cosine of angle in radians.

Arguments:

- **angle**

  An angle in radians.

Example 9-38. Cos example

```
> (cos (/ pi 2.0))
;; returns: 0.0
> (cos pi)
;; returns: -1.0
```

See Also: cosd

cosd [Function]

Returns the cosine of angle in degrees.

Arguments:

- **angle**

  Any angle in degrees.

Example 9-39. Cosd example

```
> (cosd 90.0)
;; returns: 0.0
> (cosd 180.0)
;; returns: -1.0
```

See Also: cos

cosh [Function]

Returns the hyperbolic cosine of an angle.

Arguments:

- **angle**

  Angle.

See Also: cosd cos

sin [Function]

Returns the sine of angle in radians.

Arguments:

- **angle**

  Function to return the the sine of an angle specified in radians.

Example 9-40. Sin example

```
> (sin pi)
;; returns: 0.0
> (sin (/ pi 2))
```

```
;; returns: 1.0
```

See Also: sind

sind [Function]

Returns the sine of angle in degrees.

Arguments:

- **angle**

  An angle in degrees.

Example 9-41. Sind example

```
(sind 30.0)
;; returns: 0.5
(sind 180)
;; returns: 0.0
```

See Also: sin

sinh [Function]

Returns the hyperbolic sine of an angle.

Arguments:

- **angle**

  Angle.

See Also: sin sind

tan [Function]

Returns the tangent of angle in radians.

Arguments:

- **angle**

  An angle in radians.

Example 9-42. Tan example

```
> (tan pi)
;; returns: 0.0
> (tan (/ pi 3))
;; returns: 1.7320508075688767
```

See Also: tand

tand [Function]

Function to return the tangent of an angle specified in degrees.

Arguments:

- **angle**

  An angle in degrees.

Example 9-43. Tand example

```
> (tand 30.0)
;; returns 0.5773502691896257
> (tand 180.0)
;; returns 0.0
> (tand 60.0)
;; returns 1.7320508075688767
```

See Also: tan

tanh [Function]

Returns the hyperbolic tangent of an angle.

Arguments:

- **angle**

  Angle.

See Also: tan tand

Vector and Matrix Functions

add-vectors [Function]

This function performs the vector addition of an arbitrary number of vectors.

Arguments:

- **vectors**

  A list of vectors represented as a list of X,Y,Z coordinates.

Example 9-44. Add-vectors example

```
> (add-vectors (list 1.0 1.0 1.0) (list 1.0 2.0 3.0))
;; returns the new point (2.0 3.0 4.0)
> (add-vectors (list 1.0 1.0 1.0) (list -1.0 -2.0 -3.0))
;; returns the new point (0.0 -1.0 -2.0)
```

angle-between-2-vectors [Function]

Function which returns the minimum angle between two vectors.

Arguments:

- **vector1**

  A vector represented as a list of X,Y,Z coordinates.

- **vector2**

  A vector represented as a list of X,Y,Z coordinates.

Example 9-45. Angle-between-2-vectors example

```
> (angle-between-2-vectors (list 1.0 0.0 0.0) (list 0.0 1.0 0.0))
;; returns 90.0
> (angle-between-2-vectors (list 1.0 1.0 1.0) (list 0.0 1.0 0.0))
;; returns 54.735610317245346
```

cross-product [Function]

This function takes the cross product of two vectors. The arguments are two lists of X,Y,Z coordinates. The value returned is the same type as the first argument. If the two vectors are parallel, the value returned will be '(0 0 0).

Arguments:

- **vector1**

  A vector represented as a list of X,Y,Z coordinates.

- **vector2**

  A vector represented as a list of X,Y,Z coordinates.

Example 9-46. Cross-product example

```
> (cross-product (list 1.0 0.0 0.0) (list 0.0 1.0 0.0))
;; returns : (0.0 0.0 1.0)
> (cross-product (list 1.0 0.0 0.0) (list 1.0 1.0 1.0))
;; returns : (0.0 -1.0 1.0)
```

dot-product [Function]

This function computes the dot product of two vectors. The arguments are two lists of X,Y,Z coordinates. The value returned is a floating point number.

Arguments:

- **vector1**

  A vector represented as a list of X,Y,Z coordinates.

- **vector2**

  A vector represented as a list of X,Y,Z coordinates.

Example 9-47. Dot-product example

```
> (dot-product (list 1.0 0.0 0.0) (list 0.0 0.0 0.0))
;; returns 0.0
> (dot-product (list 1.0 0.0 0.0) (list -2.0 0.0 0.0))
;; returns -2.0
> (dot-product (list 1.0 1.0 0.0) (list -2.0 1.0 0.0))
;; returns -1.0
```

mid-point [Function]

Given an arbitrary number of points, this function will return the midpoint of the points supplied.

Arguments:

- **points**

  An arbitrary number of points specified as lists.

Example 9-48. Mid-point example

```
> (mid-point (list 0.0 0.0 0.0) (list 10.0 10.0 10.0))
;; returns (5.0 5.0 5.0)
> (mid-point (list 0.0 0.0 0.0) (list 10.0 10.0 10.0) (list 15.0 10.0 10.0))
;; returns (8.333333333333334 6.666666666666667 6.666666666666667)
```

multiply-matrices [Function]

Given two matrices, the result of matrix multiplication is returned. If resultant-matrix is supplied that matrix will be destructively modified. The number of columns of matrix-a must equal the number of rows in matrix-b.

Note: The dimensions of the resultant matrix will be the rows of matrix-a and the columns of matrix-b. Therefore resultant-matrix must be at least that large.

Arguments:

- **matrix-a**

  Must be a matrix of type array.

- **matrix-b**

  Must be a matrix of type array.

Optional Arguments:

- **resultant-matrix**

  Must be a matrix of type array.

Example 9-49. Multiply-matrices example

```
> (setf *print-array* t)
;; returns : T
```

```
> (setf a (IDENTITY-MATRIX))
;; sets a matrix 'a' to ((1.0 0.0 0.0 0.0) (0.0 1.0 0.0 0.0) (0.0 0.0 1.0
0.0) (0.0 0.0 0.0 1.0))
> (setf b (make-array '(4 1) :element-type 'float :initial-element 0.0))
;; sets matrix 'b' to be 4x1 with all 0s : ((0.0) (0.0) (0.0) (0.0))
> (setf (aref b 0 0) 2.0)
;; sets the first column in b to be 2
> (setf (aref b 3 0) 1.0)
;; sets the fourth row if b to be 1
> b
;; prints b : ((2.0) (0.0) (0.0) (1.0))
> (multiply a b)
;; miltiplies the both matrices and gets : ((2.0) (0.0) (0.0) (1.0))
```

multiply-vector-by-scalar [Function]

This function performs the multiplication of a vector by a scalar.

Arguments:

- **vector**

  A vector represented as a list of X,Y,Z coordinates.

- **scalar**

  A scalar value.

Example 9-50. Multiply-vector-by-scalar example

```
> (multiply-vector-by-scalar (list 1.0 2.0 3.0) 2.0)
;; returns new vector : (2.0 4.0 6.0)
> (multiply-vector-by-scalar (list 1.0 2.0 3.0) -2.0)
;; returns new vector : (-2.0 -4.0 -6.0)
```

See Also: negate-vector

negate-vector [Function]

Returns the vector in the opposite direction of the vector supplied.

Arguments:

- **vector**

  The vector to negate represented as a list of X,Y,Z coordinates.

Example 9-51. Negate-vector example

```
> (negate-vector (list 3.0 -2.0 4.0))
;; returns the new vector : (-3.0 2.0 -4.0)
```

normalize [Function]

The normalize function returns a unit vector with the same orientation as the specified vector. The vector is represented as a list of X,Y,Z coordinates.

Arguments:

- **list-or-vector**

  A vector to normalize represented as a list of X,Y,Z coordinates.

Optional Arguments:

- **tolerance**

  The tolerance value is how close each element of the vector has to be to zero before the vector is considered a zero vector.

Example 9-52. Normalize example

```
> (normalize (list 1.0 1.0 1.0))
;; returns : (0.5773502691896258 0.5773502691896258 0.5773502691896258)
```

roughly-same-vector [Function]

Roughly-same-vector returns a t or nil to indicate if two given vectors are within a given tolerance. The input vector is represented as a list of X,Y,Z coordinates.

Arguments:

- **vector1**

  A vector represented as a list of X,Y,Z coordinates.

- **vector2**

  A vector represented as a list of X,Y,Z coordinates.

Optional Arguments:

- **tolerance**

  Optional allowable deviation.

  Default Value: 0.00001

Example 9-53. Roughly-same-vector example

```
> (roughly-same-vector (list 1.0 1.0 1.0) (list 1.0 1.0 1.01))
;; returns NIL
> (roughly-same-vector (list 1.0 1.0 1.0) (list 1.0 1.0 1.01) 0.02)
;; returns T
```

round-point [Function]

The round-point method is defined to perform a round-real on each element in a list.

Arguments:

- **list**

  A point represented as a list of X,Y,Z coordinates.

- **decimal-places**

  Should be an integer value to indicate the number of digits following the decimal point.

Example 9-54. Round-point example

```
> (round-point (list 1.33333 1.33333 1.33333) 2)
;; returns (1.33f0 1.33f0 1.33f0)
> (round-point (list 1.6666 1.6666 1.6666) 2)
;; returns (1.67f0 1.67f0 1.67f0)
```

subtract-vectors [Function]

This function performs the vector subtraction of an arbitrary number of vectors.

Arguments:

- **vectors**

  A list of vectors represented as a list of X,Y,Z coordinates. Two or more vectors need to be supplied as arguments.

Example 9-55. Substract-vectors example

```
> (subtract-vectors (list 1.0 1.0 1.0) (list -1.0 -2.0 -3.0))
;; returns (2.0 3.0 4.0)
> (subtract-vectors (list 1.0 1.0 1.0) (list 1.0 2.0 3.0))
;; returns (0.0 -1.0 -2.0)
```

vector-length [Function]

This function is used to calculate the length of a given vector.

Arguments:

- **vector**

  The vector whose length is to be found, represented as a list of X,Y,Z coordinates.

Example 9-56. Vector-length example

```
> (vector-length (list 1.0 2.0 0.0))
;; returns 2.23606797749979
> (vector-length (list 1.0 0.0 0.0))
;; returns 1.0
```

Geometric Functions

3-point-arc [Function]

This function returns information for arcs defined in an x-y plane. Given the coordinates for three points on the arc, this will return a list containing the center x and y coordinates, the radius, the start angle and the end angle.

Arguments:

- **x1**

  The x-coordinate for the start point of the arc.

- **y1**

  The y-coordinate for the start point of the arc.

- **x2**

  The x-coordinate for a point on the chord of the arc.

- **y2**

  The y-coordinate for a point on the chord of the arc.

- **x3**

  The x-coordinate for the end point of the arc.

- **y3**

  The y-coordinate for the end point of the arc.

Optional Arguments:

- **pixels?**

  Whether to return the arc definition in pixels or standard coordinates. The major difference being the direction of the arc.

Example 9-57. 3-point-arc example

```
> (3-point-arc 10.0 0.0 0.0 10.0 -10.0 0.0)
;; returns : (0.0 0.0 10.0 0.0 180.0)
> (3-point-arc 1.0 0.0 5.0 10.0 10.0 0.0)
;; returns : (5.5 4.0 6.020797289396148 -41.63353933657021 221.6335393365702)
```

See Also: 3-point-arc-info

3-point-arc-info [Function]

Given three points on an arc, this function will return (if :values? is true) a list containing the center, angle, start-vector, and normal of the arc being defined in the form of multiple values. By default it only returns the center of the arc.

Arguments:

- **point1**

The start point of the arc.

- **point2**

  A point on the chord of the arc.

- **point3**

  The end point of the arc.

Keyword Arguments:

- **:values?**

  If it is t, it will return the center, the angle, the start-vector and the normal of the arc, otherwise by default it will just return the arc center.

  Default Value: nil

Example 9-58. 3-point-arc-info example

```
> (3-point-arc-info (list 10.0 0.0 0.0) (list 0.0 10.0 0.0) (list -10.0 0.0
0.0) :values?)
;; returns :
(1.7763568394002505e-15 1.7763568394002505e-15 0.0)
180.0
(9.999999999999998 -1.7763568394002505e-15 0.0)
(0.0 0.0 1.0)
```

See Also: 3-point-arc

3-point-circle-info [Function]

Given three points on the circumference of the desired circle, this function will return a list containing the center and normal(only if :values? is t) of the circle. If the three points are colinear, nil will be returned.

Arguments:

- **point1**

  An arbitrary point on the circle.

- **point2**

  An arbitrary point on the circle.

- **point3**

  An arbitrary point on the circle.

Keyword Arguments:

- **:values?**

  If it is t, it will return the center and the normal of the circle, otherwise by default it will just return the center

  Default Value: nil

Example 9-59. 3-point-circle-info example

```
> (3-point-circle-info (list 10.0 0.0 0.0) (list 0.0 10.0 0.0) (list -10.0
0.0 0.0) :values? t)
;; returns :
(1.7763568394002505e-15 1.7763568394002505e-15 0.0)
(0.0 0.0 1.0)
```

3-point-on-same-line [Function]

Arguments:

- **point1**

  An arbitrary point.

- **point2**

  An arbitrary point.

- **point3**

  An arbitrary point.

Example 9-60. 3-point-on-same-line example

```
> (3-point-on-same-line (list 10.0 12.0 1.0) (list 3.0 4.0 5.0) (list 0.0 0.0
0.0))
;; returns NIL
> (3-point-on-same-line (list 1.0 2.0 3.0) (list 3.0 4.0 5.0) (list 6.0 7.0
8.0))
;; returns T
```

add-points [Function]

This function performs the addition of two points that must be in the XYZ coords and that makes them slightly more efficient when called repeatedly.

Arguments:

- **p1**

  First Point represented as a list of X,Y,Z coordinates.

- **p2**

  Second Point represented as a list of X,Y,Z coordinates.

Example 9-61. Add-points example

```
>(add-points '(1 2 3) '(6 5 4))
;;returns (7 7 7)
```

arc-mid-point [Function]

Based on the arc information provided, this will return the mid-point of the arc.

Arguments:

- **arc-info**

  A list consisting of the arc center, radius, normal, start-vector, and angle.

Example 9-62. Arc-mid-point example

```
> (arc-mid-point (list (list 0.0 0.0 0.0) 2.0 (list 0.0 0.0 1.0) (list 1.0
0.0 0.0) 180.0))
;; returns : (0.0 2.0 0.0)
```

circles-are-in-same-plane [Function]

If two specified circles are coplanar return t (TRUE), otherwise nil (FALSE).

Arguments:

- **center1**

  Center of circle 1: X Y Z point coordinates in list format.

- **normal1**

  Normal to circle 1: X Y Z direction coordinates in list format.

- **center2**

  Center of circle 2: X Y Z point coordinates in list format.

- **normal2**

  Normal to circle 2: X Y Z direction coordinates in list format.

Example 9-63. Circles-are-in-same-plane Example

```
> (circles-are-in-same-plane '(1.0 2.0 0.0) '(0.0 0.0 1.0) '(1.0 0.0 0.0)
'(0.0 0.0 1.0))
;; returns T
> (circles-are-in-same-plane '(1.0 2.0 3.0) '(0.0 0.0 1.0) '(1.0 0.0 0.0)
'(0.0 0.0 1.0))
;; returns NIL
```

distance-from-point-to-plane [Function]

Returns the minimum distance from a point to a plane.

Arguments:

- **point-to-measure-from**

  A point represented as a list of X,Y,Z coordinatesa .

- **point-on-plane**

A point on the plane represented as a list of X,Y,Z coordinates.

- **plane-normal**

  A normal to the plane represented as a list of X,Y,Z coordinates.

Example 9-64. Distance-from-point-to-plane example

```
> (distance-from-point-to-plane (list 10.0 0.0 0.0) (list 0.0 0.0 0.0) (list
1.0 0.0 0.0))
;; returns -10.0
> (distance-from-point-to-plane (list 10.0 0.0 0.0) (list 0.0 0.0 0.0) (list
-1.0 0.0 0.0))
;; returns 10.0
```

get-arc-start/end-angle [Function]

For an arc defined in an x-y plane, this function returns the angle of the first point with respect to the x-axis.

Arguments:

- **x**

  The x coordinate of the arc start or end.

- **y**

  The y coordinate of the arc start or end.

- **center-x**

  The x coordinate of the center of the arc.

- **center-y**

  The y coordinate of the center of the arc.

Optional Arguments:

- **pixel?**

  Tells whether the input values are in pixels or not.

  Default Value: nil

Example 9-65. Get-arc-start/end-angle example

```
(get-arc-start/end-angle 1.0 1.0 0.0 0.0)
;; returns : 45.0
(get-arc-start/end-angle 1 1 0 0 t)
;; returns : 314.9999987477609
```

get-connected-components [Function]

This function takes a list of "edges", arranges them into disjoint loops or polylines and returns the vertices of the resultant loops or polylines. This function will not work correctly if there are more than

two edges meeting at the same vertex. Each edge is defined as a list (vertex1 info vertex2) where info may or may not be present. Vertex1 and vertex2 can be points or geom-ids that are compared using a vertex-test function to determine which edges are adjacent. The function returns a list of connected components, with each component is a list of: vertex-info - A list of vertices in the order that defined the polyline or loop (clockwise or counter-clockwise cannot be determined ahead of time, the first vertex will be the first in the list) The list will also contain the info from the edges as provided in the edges-list between the corresponding vertices. flag - t or nil identifying whether the component is a closed loop or an open polyline. flip-info - A list whose length is equal to number-of-edges-in-loop - 1. Each item is t or nil based on whether the edges (starting with the second one) are used in the order of input vertices or if it is reversed.

Arguments:

- **edges-list**

  List of edges. Each edge is defined as a list (vertex1 info vertex2) where info may or may not be present. Vertex1 and vertex2 can be points, geom IDs... that are compared using the vertex-test function to determine which edges are adjacent.

Keyword Arguments:

- **:vertext-test**

  Name of the function used for vertex comparison. The vertex1 and vertex2 provided with edges-list must follow the type/format accepted by the vertext-test function. For example, if vertex-test is a function that expects two geom IDs, vertex1 and vertex2 must be geoms IDs for all edges in the edgeslist. The available vertext-test functions are: 'same-point which expects a vertex1 and vertex2 to be defined as an XYZ coordinate list, and 'same-vertex which expects vertex1 and vertex2 to be vertex geom IDs.

  Default Value: 'same-vertex

- **:same-point-tol**

  Distance within which two points are considered overlapping.

  Default Value: 0.00001

Example 9-66. Get-connected-components example

```
(get-connected-components '(
        ((0 0 0) E1 (1 0 0))
        ((1 1 0) E2 (0 1 0))
        ((1 1 0) E3 (1 0 0))
        ((0 1 0) E4 (0 0 0))
        ((20 20 0) E7 (20 0 0))
        ((10 0 0) E5 (20 0 0))
        ((20 20 0) E6 (0 20 0))
        )
        :Vertex-test 'same-point
        )
;; returns:
((((0 0 0) E1 (1 0 0) E3 (1 1 0) E2 (0 1 0) E4 (0 0 0)) T (T NIL NIL))
(((0 20 0) E6 (20 20 0) E7 (20 0 0) E5 (10 0 0)) NIL (NIL T)))
```

get-connected-nodes-lists-from-connectivity-list [Function]

The function takes node and edge data as input and returns the nodes separated into components, where each component has the set of nodes that can be reached directly or indirectly from each other by moving along edges. It returns a list of lists, where each list is the complete list of nodes that can be reached directly or indirectly from each other. There is no specified order in the output data for either the order of components, or the order of nodes within a component.

Arguments:

- **connectivity-list**

  A list of lists where each list is a list of node and connectivity data. The connectivity data for a node is the list of nodes that the node is adjacent to, i.e. it identifies a set of edges out from the node.

Example 9-67. Get-connected-nodes-lists-from-connectivity-list example

```
(get-connected-nodes-lists-from-connectivity-list
   '(
     (a (b d))
     (b (a e c))
     (c (b f))
     (d (a e g))
     (e (b d f h))
     (f (e c f))
     (g (d h))
     (h (g e f))
     (p (q r))
     (q (p r))
     (r (p q s))
     (s (r))
     )
   )
;;; The return value ((g d a b e f c h) (p q r s))
```

line-is-in-plane [Function]

This function determines whether a line (a base vector and direction vector) lies within a given plane (specified by a point and a normal vector). If the line starting at b and in direction r is in the plane through point at c with normal n then the return value is t (TRUE), otherwise nil (FALSE).

Arguments:

- **b**

  Starting point of line represented as a list of X,Y,Z coordinates.

- **r**

  Direction vector of the line represented as a list of X,Y,Z coordinates.

- **c**

  Point on the plane represented as a list of X,Y,Z coordinates.

- **n**

Normal to the plane represented as a list of X,Y,Z coordinates.

Example 9-68. Line-is-in-plane example

```
> (line-is-in-plane '(1.0 0.0 0.0) '(0.0 1.0 0.0) '(1.0 0.0 0.0) '(0.0 0.0
1.0))
;; returns T
> (line-is-in-plane '(1.0 0.0 0.0) '(0.0 1.0 0.0) '(1.0 1.0 0.0) '(0.0 0.0
1.0))
;; returns T
> (line-is-in-plane '(1.0 0.0 0.0) '(0.0 1.0 0.0) '(1.0 0.0 1.0) '(0.0 0.0
1.0))
;; returns NIL
```

line-parallel? [Function]

Returns t if the two lines specified are parallel, nil if they are not.

Arguments:

- **line1**

  A list of two points specifying the end points of the line.

- **line2**

  A list of two points specifying the end points of the line.

Example 9-69. Line-parallel? example

```
(line-parallel? '((0.0 0.0 0.0) (1.0 1.0 1.0)) '((2.0 1.0 1.0) (3.0 2.0
2.0)))
;; returns T
(line-parallel? '((0.0 0.0 0.0) (1.0 1.0 1.0)) '((2.0 1.0 1.0) (3.0 2.0
1.0)))
;; returns NIL
```

offset-line [Function]

This function will return the end points of a line that is offset the specified distance and direction from the original line.

Arguments:

- **line-info**

  A list containing the end points of the line.

- **vector**

  The direction to offset the line.

- **distance**

  The distance to offset the line.

Example 9-70. Offset-line example

```
> (offset-line '((0.0 0.0 0.0) (1.0 0.0 0.0)) '(0.0 1.0 0.0) 1.0)
;; returns : ((0.0 1.0 0.0) (1.0 1.0 0.0))
> (offset-line '((1.0 1.0 0.0) (1.0 2.0 3.0)) '(1.0 1.0 1.0) 2.0)
;; returns : ((2.1547005383792515 2.1547005383792515 1.1547005383792517)
(2.1547005383792515 3.1547005383792515 4.1547005383792515))
```

point-along-vector [Function]

Projects a point a given distance along a vector.

Arguments:

- **start-point**

  The point to project along the vector.

- **vector**

  The direction the point will be projected along represented as a list of X,Y,Z coordinates.

- **distance**

  The distance to project the point along the vector.

Example 9-71. Point-along-vector example

```
> (point-along-vector (list 0.0 0.0 0.0) (list 1.0 0.0 0.0) 2.5)
;; returns : (2.5 0.0 0.0)
> (point-along-vector (list 0.0 0.0 0.0) (list 1.0 1.0 1.0) 2.0)
;; returns : (1.1547005383792517 1.1547005383792517 1.1547005383792517)
```

points-distance [Function]

Returns the distance between two points.

Arguments:

- **point1**

  A point represented as a list of X,Y,Z coordinates.

- **point2**

  A point represented as a list of X,Y,Z coordinates.

Example 9-72. Points-distance example

```
> (points-distance (list 1.0 1.0 1.0) (list 1.0 0.0 0.0))
;; returns : 1.4142135623730951
```

point-in-triangle [Function]

This function checks if the point is within the triangle.

Arguments:

- **point**

  A point represented as a list of X,Y,Z coordinates to test.

- **tri-coords**

  Coordinates of the triangle vertices.

Keyword Arguments:

- **project-to-plane?**

  When t, the projection of the point to the plane of the triangle is checked.

  Default Value: t

point-is-in-plane [Function]

If the point specified is in the specified plane then return t (TRUE), otherwise nil (FALSE).

Arguments:

- **point**

  A point represented as a list of X,Y,Z coordinates.

- **plane-point**

  A point (represented as a list of X,Y,Z coordinates) defining the plane location.

- **plane-normal**

  A vector (represented as a list of X,Y,Z coordinates) defining the plane normal.

Example 9-73. Point-is-in-plane example

```
(point-is-in-plane '(1.0 0.0 0.0) '(0.0 0.0 0.0) '(0.0 0.0 1.0))
;; returns T
(point-is-in-plane '(1.0 0.0 0.0) '(0.0 0.0 0.0) '(1.0 1.0 1.0))
;; returns NIL
(point-is-in-plane '(1.0 2.0 -3.0) '(0.0 0.0 0.0) '(1.0 1.0 1.0))
;; returns T
```

point-line-distance [Function]

This function returns the perpendicular (closest) distance between a point and a line. If :values? is set to t, it also returns the location of the point on the line closest to the given point.

Arguments:

- **b**

  Base point of line (represented as a list of X,Y,Z coordinates).

- **d**

  Direction vector of line (represented as a list of X,Y,Z coordinates).

- **point**

Point (represented as a list of X,Y,Z coordinates) to project onto line.

Keyword Arguments:

- **:values?**

  If it is t, it will return the perpendicular distance, the location of the point on the line closest to the given point, otherwise by default it will just return the perpendicular distance.

  Default Value: nil

Return Values: D - Perpendicular distance between point and line. p - Point on line closest to specified point (in list format). flags - Variable containing information on results of operation : 1. Point p does not lie within line segment. 2. Point p lies before beginning of line segment. 3. Specified line has zero length.

Example 9-74. Point-line-distance example

```
(point-line-distance '(0.0 0.0 0.0) '(1.0 1.0 1.0) '(1.0 0.0 0.0))
;; returns : 0.816496580927726
(point-line-distance '(0.0 0.0 0.0) '(1.0 1.0 1.0) '(2.0 2.0 2.0) :values? t)
;; returns :
   (0.0 (2.0 2.0 2.0) 1) ; See flag 1
(point-line-distance '(0.0 0.0 0.0) '(1.0 1.0 1.0) '(1.0 0.0 1.0) :values? t)
;; returns :
  (0.816496580927726 (0.6666666666666666 0.6666666666666666
0.6666666666666666) 0)
(point-line-distance '(0.0 0.0 0.0) '(1.0 1.0 1.0) '(-2.0 -2.0 -2.0) :values?
t)
;; returns :
  ( 0.0   (-2.0 -2.0 -2.0)   3 ); See flags 1 and 2
```

See Also: project-point-to-line

point-on-arc? [Function]

Used to determine if the point is on the specified arc.

Arguments:

- **point**

  A point represented as a list of X,Y,Z coordinate.

- **arc-info**

  A list containing the center, radius, normal, start-vector, and angle of the arc.

Example 9-75. Point-on-arc? example

```
(point-on-arc? '(1.0 0.0 0.0) '((0.0 0.0 0.0) 1.0 (0.0 0.0 1.0) (1.0 0.0 0.0)
90.0))
;; returns : (1.0 0.0 0.0)
(point-on-arc? '(0.0 1.0 0.0) '((0.0 0.0 0.0) 1.0 (0.0 0.0 1.0) (1.0 0.0 0.0)
90.0))
```

```
;; returns : (0.0 1.0 0.0)
(point-on-arc? '(0.0 -1.0 0.0) '((0.0 0.0 0.0) 1.0 (0.0 0.0 1.0) (1.0 0.0
0.0) 90.0))
;; returns : NIL
(point-on-arc? '(0.0 10.0 0.0) '((0.0 0.0 0.0) 1.0 (0.0 0.0 1.0) (1.0 0.0
0.0) 90.0))
;; returns : NIL
```

point-on-line-segment? [Function]

Used to determine if a point is on the specified line segment.

Arguments:

- **point**

  A point represented as a list of X,Y,Z coordinates.

- **line-info**

  A list of the end points of the line.

Example 9-76. Point-on-line-segment? example

```
(point-on-line-segment? '(1.0 1.0 1.0) '((0.0 0.0 0.0) (5.0 5.0 5.0)))
;; returns : (1.0 1.0 1.0)
(point-on-line-segment? '(10.0 10.0 10.0) '((0.0 0.0 0.0) (5.0 5.0 5.0)))
;; returns : NIL
(point-on-line-segment? '(1.0 2.0 3.0) '((1.0 2.0 3.0) (5.0 5.0 5.0)))
;; returns : (1.0 2.0 3.0)
```

project-point-to-face [Function]

Projects a point onto a plane.

Arguments:

- **point**

  A point represented as a list of X,Y,Z coordinates.

- **face-point**

  A point on the face represented as a list of X,Y,Z coordinates.

- **normal**

  The normal to the face represented as a normalized list of X,Y,Z coordinates.

Example 9-77. Project-point-to-face example

```
> (project-point-to-face '(1.0 1.0 1.0) '(0.0 0.0 0.0) '(0.0 1.0 0.0))
;; returns : (1.0 0.0 1.0)
> (project-point-to-face '(1.0 2.0 3.0) '(1.0 1.0 1.0) '(1.0 1.0 1.0))
;; returns : (0.0 1.0 2.0)
> (project-point-to-face '(1.0 2.0 3.0) '(1.0 1.0 1.0) '(1.0 2.0 3.0))
```

```
;; returns : (0.4285714285714286 0.8571428571428572 1.2857142857142858)
```

project-point-to-line [Function]

Projects a point onto a line.

Arguments:

- **point**

  A point represented as a list of X,Y,Z coordinates.

- **line-end-1**

  The start point of the line represented as a list of X,Y,Z coordinates.

- **line-end-2**

  The end point of the line represented as a list of X,Y,Z coordinates.

Example 9-78. Project-point-to-line example

```
> (project-point-to-line '(1.0 1.0 0.0) '(1.0 0.0 0.0) '(0.0 0.0 0.0))
;; returns : (1.0 0.0 0.0)
> (project-point-to-line '(1.0 1.0 1.0) '(1.0 2.0 3.0) '(0.0 0.0 0.0))
;; returns : (0.4285714285714285 0.857142857142857 1.2857142857142856)
> (project-point-to-line '(1.0 2.0 3.0) '(1.0 2.0 3.0) '(0.0 0.0 0.0))
;; returns : (1.0 2.0 3.0)
```

See Also: point-line-distance

rotate-point [Function]

This function takes a 3d point and rotates it about the specified axis by a specified angle.

Arguments:

- **point**

  The point to be rotated about the axis represented as a list of X,Y,Z coordinates.

- **axis-points**

  A list of two points used to specify the axis.

- **angle**

  The rotation angle for the point. angle > 0 CCW. angle < 0 CW.

Example 9-79. Rotate-point example

```
> (rotate-point '(1.0 0.0 0.0) '((0.0 0.0 0.0) (0.0 1.0 0.0)) 45.0)
;; returns : (0.7071067811865475 0.0 -0.7071067811865475)
> (rotate-point '(1.0 0.0 0.0) '((0.0 1.0 0.0) (0.0 2.0 0.0)) 45.0)
;; returns : (0.7071067811865475 0.0 -0.7071067811865475)
```

See Also: x-rotate-point, y-rotate-point, z-rotate-point

same-point [Function]

This function expects 2 points and a tolerance as inputs and returns t or nil based on whether the points match within tolerance.

Arguments:

- **point1**

  List of the x,y,x coordinates of first point.

- **point2**

  List of the x,y,z coordinates of second point.

- **tolerance**

  Real number specifying tolerance.

subtract-points [Function]

This function performs the substraction of two points that must be in the XYZ coords and that makes them slightly more efficient when called repeatedly.

Arguments:

- **p1**

  First Point in XYZ format.

- **p2**

  Second Point in XYZ format.

Example 9-80. Add-points example

```
> (subtract-points '(6 5 4) '(3 2 1))
;;returns (3 3 3)
```

vector-component-in-plane [Function]

Returns the component of the vector in the plane with the given normal.

Arguments:

- **vector**

  A vector represented as a list of X,Y,Z coordinates.

- **normal**

  A vector represented as a list of X,Y,Z coordinates.

Example 9-81. Vector-component-in-plane example

```
> (vector-component-in-plane '(1.0 1.0 1.0) '(1.0 0.0 0.0))
;; returns : (0.0 1.0 1.0)
```

```
> (vector-component-in-plane '(1.0 1.0 1.0) '(0.0 1.0 0.0))
;; returns : (1.0 0.0 1.0)
> (vector-component-in-plane '(1.0 1.0 1.0) '(0.0 0.0 1.0))
;; returns : (1.0 1.0 0.0)
> (vector-component-in-plane '(1.0 1.0 1.0) '(1.0 1.0 1.0))
;; returns : (-2.220446049250313e-16 -2.220446049250313e-16 -
2.220446049250313e-16)
> (vector-component-in-plane '(1.0 0.0 0.0) '(1.0 1.0 1.0))
;; returns : (0.6666666666666666 -0.33333333333333337 -0.33333333333333337)
```

vector-rotation-angle [Function]

This function is used to determine the angle the first vector would have to rotate to point in the same direction as the second vector. If a constraint vector is supplied, the angle returned will be the angle that vector1 would have to rotate about the constraint vector to be in the same plane as vector2 and the constraint vector.

Arguments:

- **vector1**

  The vector to rotate represented as a list of X,Y,Z coordinates.

- **vector2**

  The vector that vector1 is rotated toward represented as a list of X,Y,Z coordinates.

Optional Arguments:

- **constraint**

  If supplied, vector1 will be rotated about this vector represented as a list of X,Y,Z coordinates.

Example 9-82. Vector-rotation-angle example

```
> (vector-rotation-angle (list 1.0 2.0 1.0) (list 1.0 0.0 0.0))
;; returns : 65.9051574478893
> (vector-rotation-angle (list 1.0 2.0 1.0) (list 1.0 0.0 0.0) (list 0.0 0.0
1.0))
;; returns : -63.43494882292201
```

vectors-are-parallel [Function]

If vectors are parallel (in same or opposite direction) then return t (TRUE), otherwise nil (FALSE).

Arguments:

- **vector1**

  A vector represented as a list of X,Y,Z coordinates.

- **vector2**

  A vector represented as a list of X,Y,Z coordinates.

Example 9-83. Vectors-are-parallel example

```
> (vectors-are-parallel '(1.0 2.0 3.0) '(1.0 2.0 3.0))
;; returns : T
> (vectors-are-parallel '(1.0 2.0 3.0) '(-1.0 -2.0 -3.0))
;; returns : T
> (vectors-are-parallel '(1.0 2.0 3.0) '(1.0 2.0 3.01))
;; returns : NIL
```

x-rotate-point [Function]

This function takes a 3d point and rotates it about the x-axis by a specified angle.

Arguments:

- **point**

  The point to be rotated about the x-axis.

- **angle**

  The rotation angle for the point.

Example 9-84. X-rotate-point example

```
> (x-rotate-point '(1.0 1.0 1.0) 25.4)
;; returns : (1.0 0.47440015946015496 1.3322704262664467)
> (x-rotate-point '(1.0 0.0 0.0) 90.0)
;; returns (0.0 0.0 1.0)
```

See Also: rotate-point, y-rotate-point, z-rotate-point

y-rotate-point [Function]

This function takes a 3d point and rotates it about the y-axis by a specified angle.

Arguments:

- **point**

  The point to be rotated about the y-axis.

- **angle**

  The rotation angle for the point.

Example 9-85. y-rotate-point example

```
> (y-rotate-point '(1.0 1.0 1.0) 25.4)
;; returns : (1.3322704262664467 1.0 0.47440015946015496)
> (y-rotate-point '(1.0 0.0 0.0) 90.0)
;; returns : (0.0 0.0 -1.0)
```

See Also: rotate-point, x-rotate-point, z-rotate-point

z-rotate-point [Function]

This function takes a 3d point and rotates it about the z-axis by a specified angle.

Arguments:

- **point**

  The point to be rotated about the z-axis.

- **angle**

  The rotation angle for the point.

Example 9-86. z-rotate-point example

```
> (z-rotate-point '(1.0 1.0 1.0) 25.4)
;; returns : (0.47440015946015496 1.3322704262664467 1.0)
> (z-rotate-point '(1.0 0.0 0.0) 90.0)
;; returns (0.0 1.0 0.0)
```

See Also: rotate-point, x-rotate-point, y-rotate-point

Tangent and Intersection Functions

These functions may be used to find basic relationships (intersections and tangents) between simple geometric objects. Lines are typically defined as a vector specifying a point on the line and a vector specifying the direction of the line. Line segments are defined similarly, with the first vector specifying the beginning of the segment and the second vector specifying the end of the segment relative to the beginning. Lines and line segments may be used interchangeably. Circles are defined by a vector specifying the center of the circle, a vector specifying the normal of the circle, and a scalar radius. (All Vectors are typically specified in list format) Within these defining parameters, arcs may be specified in addition to circles but no attempt is made to consider the limits of the arcs (i.e., arcs are treated as full circles). Results returned by these function are in a mulitple-value form. (see multiple-value-list to convert them into a list) Information on the results of these functions is returned in a flags variable. This variable combines information on many aspects of the function operation. Any combination of the flags possible for each function may be returned. The state of individual flags within the variable may be checked using the test-flags function.

arc-tangent-to-line-circle [Function]

This function returns up to eight possible arcs tangent to a line and a circle if :values? is true. Otherwise it just returns the first one. The existence of each of the tangent arcs depends on the geometry of the line and circle and the radius of the tangent arc. The object equivalent of this function is arc-ln-cl-tg. All vectors have ben normalized.

Arguments:

- **b**

  Base point of line (represented as a list of X,Y,Z coordinates).

- **d**

Direction vector of line (represented as a list of X,Y,Z coordinates).

- **c**

  Center of circle (represented as a list of X,Y,Z coordinates).

- **rc**

  Radius of circle.

- **n**

  Normal to the plane of the circle (represented as a list of X,Y,Z coordinates).

- **r**

  Radius of the tangent arc.

Keyword Arguments:

- **:values?**

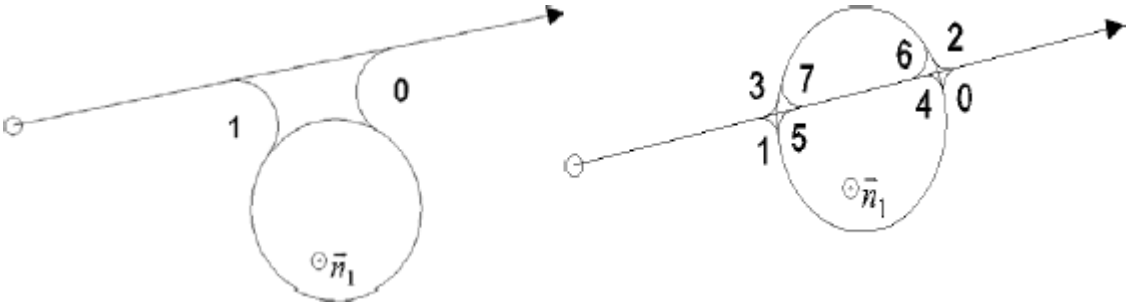
  If it is t, it will return a list with up to 8 tangent arcs, otherwise it will return just one.

  Default Value: nil

Return Values:

1. **arc1** A list containing information specifying the geometry of the first tangent arc. The list contains the arc-center (in list format), arc radius, arc normal (in list format), arc start vector (in list format), and included angle.
2. **...**
3. **arc8** A list containing information specifying the geometry of the last tangent arc.
4. **flags** A variable containing information on the results of the operation. 1 - There are no tangent arcs. 4 - The line and circle are not coplanar.

Figure 9-1. Arc-tangent-to-line-circle



Example 9-87. Arc-tangent-to-line-circle example

```
(arc-tangent-to-line-circle `(0.0 0.0 0.0) `(1.0 1.0 0.0) `(5.0 5.0 0.0) 4.0
`(0.0 0.0 1.0) 0.5 :values? t)
;; returns :
((8.515831050761653 7.808724269575105 0.0) 0.5 (0.0 0.0 1.0) (-
0.7071067811865476 0.7071067811865476 0.0) 83.62062979155722)
((2.191275730424895 1.484168949238347 0.0) 0.5 (0.0 0.0 -1.0) (-
0.7071067811865476 0.7071067811865476 0.0) 83.62062979155718)
```

```
((7.808724269575105 8.515831050761653 0.0) 0.5 (0.0 0.0 -1.0)
(0.7071067811865476 -0.7071067811865476 0.0) 83.62062979155722)
((1.484168949238347 2.191275730424895 0.0) 0.5 (0.0 0.0 1.0)
(0.7071067811865476 -0.7071067811865476 0.0) 83.62062979155718)
((7.803043133376452 7.095936352189904 0.0) 0.5 (0.0 0.0 -1.0) (-
0.7071067811865476 0.7071067811865476 0.0) 98.21321070173816)
((2.904063647810096 2.196956866623548 0.0) 0.5 (0.0 0.0 1.0) (-
0.7071067811865476 0.7071067811865476 0.0) 98.21321070173819)
((7.095936352189904 7.803043133376452 0.0) 0.5 (0.0 0.0 1.0)
(0.7071067811865476 -0.7071067811865476 0.0) 98.21321070173816)
((2.196956866623548 2.904063647810096 0.0) 0.5 (0.0 0.0 -1.0)
(0.7071067811865476 -0.7071067811865476 0.0) 98.21321070173819)
0)
(arc-tangent-to-line-circle `(0.0 0.0 0.0) `(1.0 1.0 0.0) `(5.0 5.0 0.0) 0.5
`(0.0 0.0 1.0) 1.0 :values? t)
;; returns :
(((6.4976761962286425 5.083462633855547 0.0) 1.0 (0.0 0.0 1.0) (-
0.7071067811865476 0.7071067811865476 0.0) 48.18968510422135)
((4.916537366144453 3.5023238037713575 0.0) 1.0 (0.0 0.0 -1.0) (-
0.7071067811865476 0.7071067811865476 0.0) 48.18968510422135)
((5.083462633855547 6.4976761962286425 0.0) 1.0 (0.0 0.0 -1.0)
(0.7071067811865476 -0.7071067811865476 0.0) 48.18968510422135)
((3.5023238037713575 4.916537366144453 0.0) 1.0 (0.0 0.0 1.0)
(0.7071067811865476 -0.7071067811865476 0.0) 48.18968510422135)
nil
nil
nil
nil
0)
```

arc-tangent-to-line-point [Function]

This function returns up to two possible arcs (if :values? is t, otherwise just one) tangent to a line and going through a point. The object equivalent of this function is arc-ln-pt-tg.

Arguments:

- **b**

  Base point of line (represented as a list of X,Y,Z coordinates).

- **d**

  Direction vector of line (represented as a list of X,Y,Z coordinates).

- **p**

  Location of point (represented as a list of X,Y,Z coordinates).

- **r**

  Radius of tangent arc

Keyword Arguments:

- **:values?**

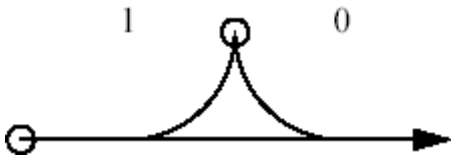
If it is t, it will return a list with up to 2 arcs, otherwise it will return just one.

Default Value: nil

Return Values:

1. **arc1** A list containing information specifying the geometry of the first tangent arc. The list contains the arc-center (in list format), arc radius, arc normal (in list format), arc start vector (in list format), and included angle.
2. **arc2** A list containing information specifying the geometry of the second tangent arc.
3. **flag** Variable containing information on the results of the operation. 1 - The specified point is on the given line. 2 - The specified point is too far from the line. 4 - The given line has no length.
4. **flag1** Variable containing information on the first arc. 1 - Arc doesn't touch line between b and b+d. 2 - Arc touches before b.
5. **flag2** Variable containing information on the second arc. 1 - Arc doesn't touch line between b and b+d. 2 - Arc touches before b.

Figure 9-2. arc-tangent-to-line-point



Example 9-88. Arc-tangent-to-line-point example

```
(arc-tangent-to-line-point `(0.0 0.0 0.0) `(1.0 1.0 0.0) `(1.0 0.0 0.0) 1.5
:values? t)
;; returns :
(((2.461026859179917 0.3397065156202743 0.0) 1.5 (0.0 0.0 1.0) (-
1.0606601717798214 1.0606601717798212 0.0) 58.08939395179045)
((0.6602934843797257 -1.4610268591799167 0.0) 1.5 (0.0 0.0 -1.0) (-
1.0606601717798212 1.0606601717798212 0.0) 58.08939395179045)
0 ;; No flag
1 ;; See flag 1
3) ;; See flag 1 and 2
(arc-tangent-to-line-point `(0.0 0.0 0.0) `(1.0 1.0 0.0) `(2.0 0.0 0.0) 0.5
:values? t)
;; returns :
(NIL
NIL
2 ;;; See flag 2
0
0)
```

arc-tangent-to-two-lines [Function]

This function returns the four arcs tangent to two lines if the :values? is t, otherwise just one arc. The arcs are returned as center, radius, normal, start vector, and angle. The arcs that are returned are

specified to start at line 1 and continue to line 2. The different arcs may have different normals. The flags for each arc indicate whether the arcs touch the line segments defined by the line vectors. The object equivalent of this function is arc-ln-ln-tg.

Arguments:

- **b1**

  Base point of line 1 (represented as a list of X,Y,Z coordinates).

- **r1**

  Direction vector of line 1 (represented as a list of X,Y,Z coordinates).

- **b2**

  Base point of line 2 (represented as a list of X,Y,Z coordinates).

- **r2**

  Direction vector of line 2 (represented as a list of X,Y,Z coordinates).

- **r**

  Radius of tangent circle.

Keyword Arguments:

- **:values?**

  If it is t, it will return a list with 4 tangent arcs, otherwise it will return just one

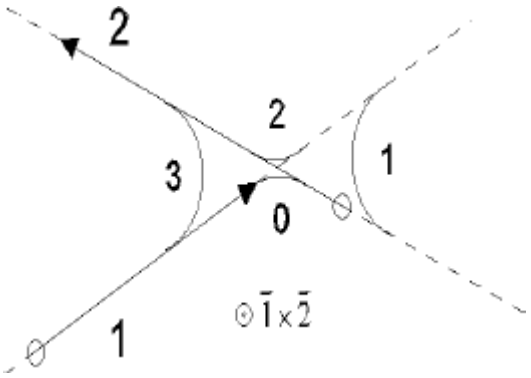
  Default Value: nil

Return Values:

1. **arc1** A list containing information specifying the geometry of the first tangent arc. The list contains: the arc center (in list format), arc radius, arc normal (in list format), arc start vector (in list format), and included angle.
2. **...**
3. **arc4** A list containing information specifying the geometry of the fourth tangent arc.
4. **flags** Variable containing information on results of operation. 1 - The lines are parallel. 2 - Reserved. 4 - Lines are not coplanar. 8 - Line 1 has zero length. 16 - Line 2 has zero length.
5. **flags1** A variable containing information on the first tangent arc. 1 - The start of the arc does not lie within the extent of line 1. 2 - The start of the arc lies before the beginning of line 1. 4 - The start of the arc does not lie within the extent of line 2. 8 - The start of the arc lies before the beginning of line 2.
6. **...**
7. **flags4** A variable containing information on the fourth tangent arc.
8. **intersection-flags** Flags variable returned from the LINE-LINE-INTERSECTION of line 1 and line 2. Only flags 1, 2, 4, and 8 are returned.

Warning: If the lines are not in the same plane the results are based on line 1 but are not an exact value and should be used with caution.

Figure 9-3. arc-tangent-to-two-lines



Example 9-89. Arc-tangent-to-two-lines example

```
(arc-tangent-to-two-lines '(0.0 0.0 0.0) '(1.0 1.0 0.0) '(6.0 0.0 0.0) '(0.0
1.0 0.0) 2.0 :values? t)
;; returns :
(((3.9999999999999999 1.1715728752538093 0.0) 2.0 (0.0 0.0 -1.0) (-
0.7071067811865476 0.7071067811865476 0.0) 135.0)
((7.9999999999999997 5.1715728752538075 0.0) 2.0 (0.0 0.0 1.0) (-
0.7071067811865476 0.7071067811865476 0.0) 45.0)
((7.9999999999999999 10.828427124746188 0.0) 2.0 (0.0 0.0 -1.0)
(0.7071067811865474 -0.7071067811865478 0.0) 135.0)
((3.9999999999999982 6.828427124746188 0.0) 2.0 (0.0 0.0 1.0)
(0.7071067811865476 -0.7071067811865476 0.0) 45.0)
0
5 ;;; See flag 1 and 4
5
5
5
5)
(arc-tangent-to-two-lines '(5.0 0.0 0.0) '(1.0 0.0 0.0) '(6.0 0.0 0.0) '(0.0
1.0 0.0) 2.0 :values? t)
;; returns :
(((4.0 -2.0 0.0) 2.0 (0.0 0.0 -1.0) (0.0 1.0 0.0) 90.0)
((8.0 -2.0 0.0) 2.0 (0.0 0.0 1.0) (0.0 1.0 0.0) 90.0)
((8.0 2.0 0.0) 2.0 (0.0 0.0 -1.0) (0.0 -1.0 0.0) 90.0)
((4.0 2.0 0.0) 2.0 (0.0 0.0 1.0) (0.0 -1.0 0.0) 90.0)
0
15
13
5
7
0)
```

circle-circle-intersection [Function]

This function returns the a list of two points of intersection (if they exist and if :values? is t) between two circles. By default it returns only one point.

Arguments:

- **center1**

  Center point of circle. A point represented as a list of X,Y,Z coordinates.

- **radius1**

  Radius of circle.

- **normal1**

  Normal vector of circle. A vector represented as a list of X,Y,Z coordinates.

- **center2**

  Center point of circle. A point represented as a list of X,Y,Z coordinates.

- **radius2**

  Radius of circle.

- **normal2**

  Normal vector of circle. A vector represented as a list of X,Y,Z coordinates.

Keyword Arguments:

- **:values?**

  If it is t, it will return a list of up to 2 points, otherwise it will return just one. As long as they exist.

  Default Value: nil

Return Values:

1. **intersection1** Point (in list format) specifying first intersection point of circles.
2. **intersection2** Point (in list format) specifying second intersection point of circles.
3. **flags** Variable containing information on results of operation. 1 - Circles do not intersect. 2 - Circles are tangent. There is only one intersection. 4 - Circles are not coplanar. 8 - Circles have same center. 16 - One circle is inside the other.

Note: If the circles are not coplanar, the intersection between circle 1 and a circle with radius2 centered about the projection of center2 on the plane of circle 1 will be found.

Example 9-90. Circle-circle-intersection example

```
(circle-circle-intersection '(2.0 0.0 0.0) 2.0 '(0.0 0.0 1.0) '(4.0 3.0 0.0)
3.0 '(0.0 0.0 1.0) :values? t)
;; returns :
((1.2307692307692308 1.846153846153846 0.0)
(4.0 -2.220446049250313e-16 0.0) 0)
(circle-circle-intersection '(2.0 0.0 0.0) 1.0 '(0.0 0.0 1.0) '(4.0 3.0 0.0)
1.0 '(0.0 0.0 1.0) :values? t)
;; returns :
(NIL NIL 1)
```

circle-tangent-to-line-circle [Function]

This function returns a list with up to eight possible circles tangent to a line and a circle, if :values? is t, otherwise it will return only one. The existence of each of the tangent circles depends on the geometry of the line and circle and the radius of the tangent circle. The pairs of vectors returned (tangent-points) are the tangent points (contact points) between the tangent circle with center and the line and circle. The object equivalents of this function are circle-ln-cl-tg and arc-ln-cl-tg.

Arguments:

- **b**

  Base point of line (represented as a list of X,Y,Z coordinates).

- **d**

  Direction vector of line (represented as a list of X,Y,Z coordinates).

- **c**

  Center of circle (represented as a list of X,Y,Z coordinates).

- **r**

  Radius of circle.

- **n**

  Normal to the plane of the circle (represented as a list of X,Y,Z coordinates).

- **rc**

  Radius of the tangent circle.

Keyword Arguments:

- **:values?**

  If it is t, it will return a list with up to 8 circles tangent to the line, otherwise it will return just one.

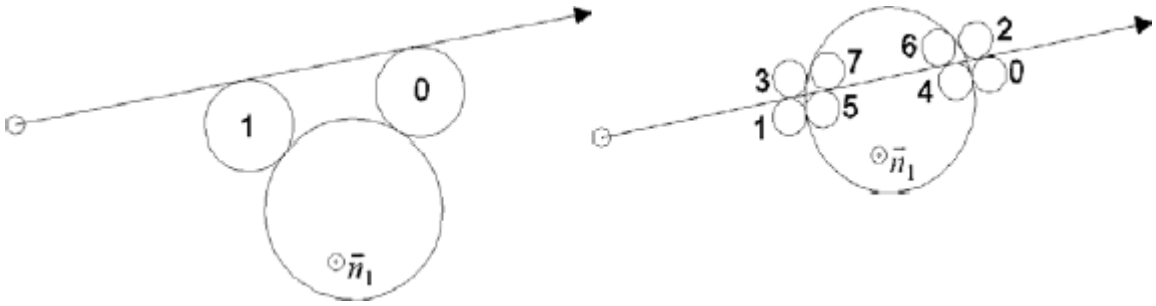
  Default Value: nil

Return Values:

1. **center1** Point (in list format) specifying the center of the first tangent circle.
2. **...**
3. **center8** Point (in list format) specifying the center of the eighth tangent circle.
4. **flags** Variable containing information on results of operation. 1 - There are no tangent circles. 2 - The line has zero length. 4 -The circle and line are not coplanar.
5. **tangent-point1** A list containing two points (in list format) which specify the point of contact between the first tangent circle and the line and original circle.
6. **...**
7. **tangent-point8** A list containing two points (in list format) which specify the point of contact between the eighth tangent circle and the line and original circle.

Note: If the line and circle are not coplanar, the tangent circles will be found in the plane of the circle.

Figure 9-4. circle-tangent-to-line-circle



Example 9-91. Circle-tangent-to-line-circle example

```
(circle-tangent-to-line-circle '(0.0 0.0 0.0) '(1.0 1.0 0.0) '(5.0 5.0 0.0)
4.0 '(0.0 0.0 1.0) 0.5 :values? t)
;; returns :
((8.515831050761653 7.808724269575105 0.0) (2.191275730424895
1.484168949238347 0.0) (7.808724269575105 8.515831050761653 0.0)
(1.484168949238347 2.191275730424895 0.0) (7.803043133376452
7.095936352189904 0.0) (2.904063647810096 2.196956866623548 0.0)
(7.095936352189904 7.803043133376452 0.0) (2.196956866623548
2.904063647810096 0.0)  0
((8.16227766016838 8.162277660168378 0.0) (8.12518315623258 7.496643795177871
0.0))
((1.8377223398316211 1.8377223398316207 0.0) (2.5033562048221287
1.8748168437674195 0.0))
((8.162277660168378 8.16227766016838 0.0) (7.496643795177871 8.12518315623258
0.0))
((1.8377223398316207 1.8377223398316211 0.0) (1.8748168437674195
2.5033562048221287 0.0))
((7.449489742783178 7.449489742783178 0.0) (8.203477866715945
7.395355831074176 0.0))
((2.550510257216822 2.550510257216822 0.0) (2.604644168925824
1.7965221332840549 0.0))
((7.449489742783178 7.449489742783178 0.0) (7.395355831074176
8.203477866715945 0.0))
((2.550510257216822 2.550510257216822 0.0) (1.7965221332840549
2.604644168925824 0.0)))
(circle-tangent-to-line-circle '(0.0 0.0 0.0) '(1.0 1.0 0.0) '(5.0 5.0 0.0)
0.5 '(0.0 0.0 1.0) 1.0 :values? t)
;; returns :
((6.4976761962286425 5.083462633855547 0.0) (4.916537366144453
3.5023238037713575 0.0)
(5.083462633855547 6.4976761962286425 0.0) (3.5023238037713575
4.916537366144453 0.0)
NIL NIL NIL NIL 0
((5.7905694150420945 5.7905694150420945 0.0) (5.4992253987428805
5.027820877951849 0.0))
((4.2094305849579055 4.2094305849579055 0.0) (4.972179122048151
4.5007746012571195 0.0))
```

```
((5.7905694150420945 5.7905694150420945 0.0) (5.027820877951849
5.4992253987428805 0.0))
((4.2094305849579055 4.2094305849579055 0.0) (4.5007746012571195
4.972179122048151 0.0))
NIL NIL NIL NIL)
```

circle-tangent-to-two-circles [Function]

This function returns a list of up to eight possible circles tangent to two circles, if :values? is t, otherwise just returns one. The centers of the circles are returned. The radius of the new circles is radius3 and they have the same normal as either circle. The existence of each of the tangent circles depends on the geometry of the two given circles and the specified radius of the tangent circles. The object equivalents of this function are arc-cl-cl-tg and circle-cl-cl-tg.

Arguments:

- **center1**

  Center point of circle. A point represented as a list of X,Y,Z coordinates.

- **radius1**

  Radius of circle.

- **normal1**

  Normal vector of circle. A vector represented as a list of X,Y,Z coordinates.

- **center2**

  Center point of circle. A point represented as a list of X,Y,Z coordinates.

- **radius2**

  Radius of circle.

- **normal2**

  Normal vector of circle. A vector represented as a list of X,Y,Z coordinates.

- **radius3**

  Radius of circle that is to be tangent.

Keyword Arguments:

- **:values?**

  If it is t, it will return a list with up to 8 circles, otherwise it will return just one.

  Default Value: nil

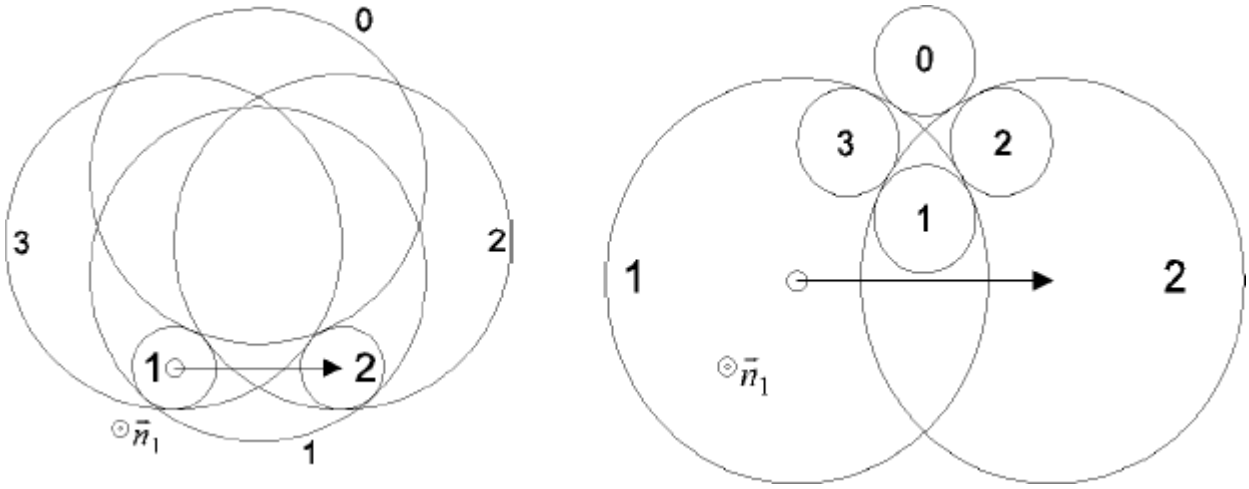
Return Values:

1. **flags** Variable containing information on results of operation. 1 - Circles intersect. There are no crossover tangents. 2 - Circles are tangent. 4 - Circles are not coplanar. 8 - One circle is inside the other. 16 - Circles have the same center. 32 - There are no tangent circles.
2. **center1** Point (in list format) specifying the center of the first tangent circle.
3. **...**

4. **center8** Point (in list format) specifying the center of the eighth tangent circle

Note: If the circles are not coplanar, the tangent circles are in the plane of circle 1. Circle 2 is treated as a circle of radius2 centered about the projection of center2 on the plane of circle 1.

Figure 9-5. circle-tangent-to-two-circles



Example 9-92. Circle-tangent-to-two-circles example

```
(circle-tangent-to-two-circles '(2.0 0.0 0.0) 2.0 '(0.0 0.0 1.0) '(4.0 3.0
0.0) 1.0 '(0.0 0.0 1.0) 3.0 :values? t)
;; returns :
(0
(0.3962638022109921 4.735824131859339 0.0)
NIL
(3.230769230769231 4.846153846153847 0.0)
(1.0467457811220566 0.30216947925196197 0.0)
(6.988351582404393 0.3410989450637385 0.0)
NIL
(6.0 3.0000000000000004 0.0)
(2.645561911185635 -0.7637079407904239 0.0))
(circle-tangent-to-two-circles '(2.0 0.0 0.0) 2.0 '(0.0 0.0 1.0) '(3.0 0.0
0.0) 1.0 '(1.0 1.0 1.0) 2.0 :values? t)
;; returns
(14 ; See flags 2, 4, and 8
(6.0 0.0 0.0)
NIL
NIL
NIL
(6.0 0.0 0.0)
NIL
NIL
NIL)
```

circle-tangent-to-two-lines [Function]

This function returns a list of up to four possible circles tangent to two lines, if :values? is t, otherwise just one. The value intersection-flags is returned from the line-line-intersection of the two lines. The normal of the resulting circles is found by the cross-product of the lines regardless of whether the two lines are coplanar or not. The object equivalent of this function is circle-ln-ln-tg.

Arguments:

- **b1**

  Base point of line 1 (represented as a list of X,Y,Z coordinates).

- **r1**

  Direction vector of line 1 (represented as a list of X,Y,Z coordinates).

- **b2**

  Base point of line 2 (represented as a list of X,Y,Z coordinates).

- **r2**

  Direction vector of line 2 (represented as a list of X,Y,Z coordinates).

- **r**

  Radius of tangent circle.

Keyword Arguments:

- **:values?**

  If it is t, it will return a list with up to 4 circles, otherwise it will return just one.

  Default Value: nil

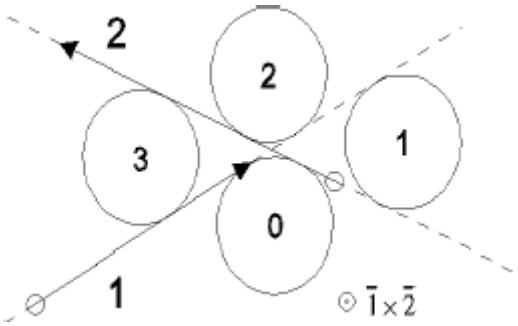
Return Values:

1. **center1** A point (in list format) specifying the center of the first tangent circle.
2. **...**
3. **center4** A point (in list format) specifying the center of the fourth tangent circle.
4. **normal** The normal to the face specified as a normalized vector.
5. **flags** Variable containing information on results of operation. 1 - The lines are parallel. 2 - Reserved. 4 - The lines are not coplanar. 8 - Line 1 has zero length. 16 - Line 2 has zero length.
6. **contact-point11** A point (in list format) specifying the contact point between tangent circle 1 and line 1.
7. **contact-point12** A point (in list format) specifying the contact point between tangent circle 1 and line 2.
8. **...**
9. **contact-point41** A point (in list format) specifying the contact point between tangent circle 4 and line 1.
10. **contact-point42** A point (in list format) specifying the contact point between tangent circle 4 and line 2.

11. **intersection-flags** The flags variable returned from the line-line-intersection of line 1 and line 2. Only flags 1, 2, 4, and 8 are returned.

Warning : If the lines are not in the same plane the results are based on line 1 but are not an exact value and should be used with caution.

Figure 9-6. circle-tangent-to-two-lines



Example 9-93. Circle-tangent-to-two-lines example

```
(circle-tangent-to-two-lines '(0.0 0.0 0.0) '(1.0 1.0 0.0) '(6.0 0.0 0.0)
'(0.0 1.0 0.0) 2.0 :values? t)
;; returns :
((3.999999999999999 1.1715728752538093 0.0) (7.999999999999997
5.1715728752538075 0.0) (7.999999999999999 10.828427124746188 0.0)
(3.9999999999999982 6.828427124746188 0.0) (0.0 0.0 1.0) 0 (2.585786437626904
2.585786437626904 0.0)
(5.999999999999999 1.1715728752538093 0.0) (6.585786437626902
6.585786437626902 0.0) (5.999999999999997 5.1715728752538075 0.0)
(9.414213562373094 9.414213562373092 0.0) (5.999999999999999
10.828427124746188 0.0) (5.414213562373093 5.414213562373093 0.0)
(5.999999999999998 6.828427124746188 0.0) 5)
(circle-tangent-to-two-lines '(5.0 0.0 0.0) '(1.0 0.0 0.0) '(6.0 0.0 0.0)
'(0.0 1.0 0.0) 2.0 :values? t)
;; returns :
((4.0 -2.0 0.0) (8.0 -2.0 0.0) (8.0 2.0 0.0) (4.0 2.0 0.0) (0.0 0.0 1.0) 0
(4.0 0.0 0.0) (6.0 -2.0 0.0) (8.0 0.0 0.0)
(6.0 -2.0 0.0) (8.0 0.0 0.0) (6.0 2.0 0.0) (4.0 0.0 0.0) (6.0 2.0 0.0) 0)
```

line-circle-intersection [Function]

This function returns a list of the intersection points (if any) between a line and a circle, if :values? is t, otherwise returns only one point. If the line and circle are not coplanar, the circle will be treated like a sphere and the results will be the intersection points between a line and a sphere of the given radius at the given center. **Warning:** This behavior is inconsistent with most of the other functions which force a coplanar result.

Arguments:

- **point1**

  Start point of line. A point represented as a list of X,Y,Z coordinates.

- **vector1**

  Direction vector of line. A vector represented as a list of X,Y,Z coordinates.

- **center**

  Center point of circle. A point represented as a list of X,Y,Z coordinates.

- **normal**

  Normal vector of circle. A vector represented as a list of X,Y,Z coordinates.

- **radius**

  Radius of circle.

Keyword Arguments:

- **:values?**

  If it is t, it will return a list of the intersection points (if any), otherwise it will return just one.

  Default Value: nil

Return Values:

1. **intersection1** Point (in list format) specifying first intersection of line and circle.
2. **intersection2** Point (in list format) specifying second intersection of line and circle.
3. **flags** Variable containing information on results of operation. 1 - There is only one intersection point (line is tangent to circle). 2 - The line does not intersect the circle. 4 - The line has zero length. 8 - The line and circle are not coplanar.

Example 9-94. Line-circle-intersection example

```
(line-circle-intersection '(0.0 0.0 0.0) '(1.0 0.0 0.0) '(5.0 1.0 0.0) '(0.0
0.0 1.0) 2.0 :values? t)
;; returns :
((6.732050807568877 0.0 0.0) (3.267949192431123 0.0 0.0) 0)
(line-circle-intersection '(4.0 0.0 0.0) '(1.0 0.0 0.0) '(5.0 1.0 0.0) '(0.0
0.0 1.0) 2.0 :values? t)
;; returns :
((6.732050807568877 0.0 0.0) (3.267949192431123 0.0 0.0) 0)
(line-circle-intersection '(0.0 0.0 0.0) '(1.0 0.0 0.0) '(5.0 2.0 0.0) '(0.0
0.0 1.0) 2.0 :values? t)
;; returns :
((5.0 0.0 0.0) (5.0 0.0 0.0) 1)
```

line-line-intersection [Function]

This function returns a vector indicating the intersection of the two lines. If :values? is t it and the lines are not in the same plane it will also return the closest approach of the lines with D the distance between the lines at those points. If the lines are in the same plane the intersection points will be the same point.

Arguments:

- **point1**

Start point of line. A point represented as a list of X,Y,Z coordinates.

- **vector1**

  Direction vector of line. A vector represented as a list of X,Y,Z coordinates.

- **point2**

  Start point of line. A point represented as a list of X,Y,Z coordinates.

- **vector2**

  Direction vector of line. A vector represented as a list of X,Y,Z coordinates.

Keyword Arguments:

- **:values?**

  If it is t, it will return a list with both intersections (if available) and the distance, otherwise it will return just one intersection.

  Default Value: nil

Return Values:

1. **intersection1** A point (in list format) specifying point on line 1 closest to line 2. If the lines are in the same plane then it is the intersection of the lines.
2. **flags** A variable containing information on results of operation. An intersection point will be returned for any value of flags less than 32. 1 - Intersection does not lie within the extent of line 1. 2 - Intersection lies before the start of line 1. 4 - Intersection does not lie within the extent of line 2. 8 - Intersection lies before the start of line 2. 16 - The lines are not in the same plane. 32 - The lines are parallel. 64 - Line 1 has zero length. 128 - Line 2 has zero length.
3. **intersection2** A point (in list format) specifying point on line 2 closest to line 1. If the lines are in the same plane, then it is the same as intersection1.
4. **D** The distance between intersection1 and intersection2 if the lines are not coplanar

Example 9-95. Line-line-intersection example

```
(line-line-intersection '(0.0 0.0 0.0) '(1.0 0.0 0.0) '(1.0 1.0 0.0) '(0.0
1.0 0.0) :values? t)
;; returns :
((1.0 0.0 0.0)
12 ; See flags 4 and 8
(1.0 0.0 0.0) 0.0)
(line-line-intersection '(0.0 3.0 0.0) '(1.0 0.0 0.0) '(1.0 1.0 0.0) '(0.0
1.0 0.0) :values? t)
;; returns :
((1.0 3.0 0.0)
4 ; See flag 4
(1.0 3.0 0.0) 0.0)
(line-line-intersection '(0.0 3.0 1.0) '(1.0 0.0 0.0) '(1.0 1.0 0.0) '(0.0
1.0 0.0) :values? t)
;; returns :
( (1.0 3.0 1.0)
20 ; See flags 4 and 16
```

```
(1.0 3.0 0.0) 1.0)
```

line-plane-intersection [Function]

This function returns t if the line lies in the plane. It returns nil if there is no intersection. Otherwise, it returns the intersection point.

Arguments:

- **line-point**

  Start point of line. A point represented as a list of X,Y,Z coordinates.

- **line-vector**

  Direction vector of line. A vector represented as a list of X,Y,Z coordinates.

- **plane-point**

  A point in the plane. A point represented as a list of X,Y,Z coordinates.

- **plane-normal**

  Normal vector of plane. A vector represented as a list of X,Y,Z coordinates.

Keyword Arguments:

- **:tolerance**

  Tolerance.

  Default Value: 1e-8

line-tangent-to-circle [Function]

This function returns a list of points (if :values? is t, otherwise just one point) on a circle such that a line passing through the given point and either returned point will be tangent to the specified circle. If the point is not in the plane of the circle, the tangent will be based on the projection of the point in the plane of the circle. The object line-pt-cl-tg corresponds to this function and creates a line which will remain tangent to the circle even if the geometric relationship between the point and circle changes.

Arguments:

- **point1**

  Start point of line. A point represented as a list of X,Y,Z coordinates.

- **center**

  Center point of circle. A point represented as a list of X,Y,Z coordinates.

- **normal**

  Normal vector of circle. A vector represented as a list of X,Y,Z coordinates.

- **radius**

  Radius of circle.

Keyword Arguments:

- **:values?**

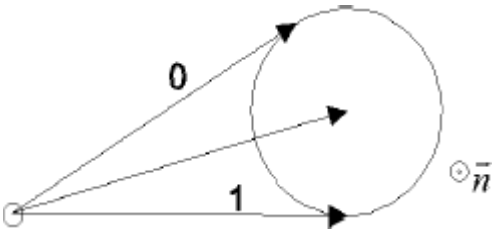
If it is t, it will return a list with of all points, otherwise it will return just one.

Default Value: nil

Return Values:

1. **tangent-point1** Point (in list format) specifying tangent point of line and circle.
2. **tangent-point2** Point (in list format) specifying tangent point of line and circle.
3. **flags** Variable containing information on results of operation. 1 - Specified point is on circle. 2 - Specified point is inside circle (no tangent line is possible). 4 - Specified point is not in plane of circle.

Figure 9-7. Line-tangent-to-circle



Example 9-96. Line-tangent-to-circle example

```
(line-tangent-to-circle '(0.0 0.0 0.0) '(5.0 1.0 0.0) '(0.0 0.0 1.0) 2.0
:values? t)
;; returns :
((3.869968018475121 2.650159907624396 0.0) (4.591570443063341 -
0.9578522153167035 0.0) 0)
(line-tangent-to-circle '(1.0 1.0 0.0) '(5.0 1.0 0.0) '(0.0 0.0 1.0) 2.0
:values? t)
;; returns :
((4.0 2.732050807568877 0.0) (4.0 -0.7320508075688772 0.0) 0)
(line-tangent-to-circle '(1.0 1.0 0.0) '(5.0 1.0 0.0) '(1.0 1.0 1.0) 2.0
:values? t)
;; returns :
((4.0 2.618033988749895 -0.6180339887498947) (4.0 0.3819660112501053
1.6180339887498947)
4 ; See flag 4
)
```

line-tangent-to-two-circles [Function]

This function returns a list of the four lines which may be tangent to two circles (if they exist), if :values? is t, otherwise it returns only one. The output lines are given as a vector which indicates a point on the first circle, and a vector which indicates the tangent point on the second circle relative to the first point. The object corresponding to this function is line-cl-cl-tg.

Arguments:

- **center1**

  Center point of circle. A point represented as a list of X,Y,Z coordinates.

- **radius1**

  Radius of circle.

- **normal1**

  Normal vector of circle. A vector represented as a list of X,Y,Z coordinates.

- **center2**

  Center point of circle. A point represented as a list of X,Y,Z coordinates.

- **radius2**

  Radius of circle.

- **normal2**

  Normal vector of circle. A vector represented as a list of X,Y,Z coordinates.

Keyword Arguments:

- **:values?**

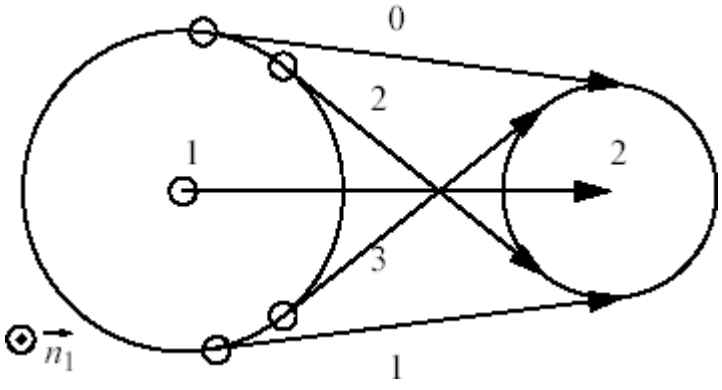
  If it is t, it will return a list of all 4 lines, otherwise it will return just one.

  Default Value: nil

Return Values:

1. **flags** Variable containing information on results of operation. 1 - Circles intersect. There are no crossover tangents. 2 - Circles are tangent. There are no crossover tangents. 4 - Circles are not coplanar. 8 - Circles have the same center. There are no crossover tangents. 16 - One circle is inside the other. There are no crossover tangents.
2. **line1-point** Point (in list format) on first tangent line.
3. **line1-vector** Vector (in list format) specifying direction of first tangent line.
4. **...**
5. **line4-point** Point (in list format) on fourth tangent line.
6. **line4-vector** Vector (in list format) specifying direction of fourth tangent line.

Figure 9-8. line-tangent-to-two-circles



Example 9-97. Line-tangent-to-two-circles

```
(line-tangent-to-two-circles '(2.0 0.0 0.0) 2.0 '(0.0 0.0 1.0) '(4.0 3.0 0.0)
1.0 '(0.0 0.0 1.0) :values? t)
;; returns :
(0 (0.7088761776287285 1.5274158815808478 0.0) (2.6455619111856357
2.236292059209576 0.0)
(3.906508437755887 -0.6043389585039246 0.0) (1.0467457811220564
3.3021694792519622 0.0) (2.0 2.0 0.0)
(1.9999999999999998 0.0 0.0) (3.8461538461538463 0.7692307692307693 0.0) (-
0.7692307692307692 1.846153846153846 0.0))
(line-tangent-to-two-circles '(2.0 0.0 0.0) 2.0 '(0.0 0.0 1.0) '(3.0 0.0 0.0)
1.0 '(0.0 0.0 1.0) :values? t)
;; returns :
(18 ; See flags 2 and 16
(4.0 0.0 0.0) (0.0 0.0 0.0) (4.0 0.0 0.0) (0.0 0.0 0.0) NIL NIL NIL NIL)
(line-tangent-to-two-circles '(2.0 0.0 0.0) 2.0 '(0.0 0.0 1.0) '(3.0 0.0 0.0)
1.0 '(1.0 1.0 1.0))
;; returns :
(22 ; See flages 2, 4, and 16
(4.0 0.0 0.0) (0.0 0.0 0.0) (4.0 0.0 0.0) (0.0 0.0 0.0) NIL NIL NIL NIL)
```

test-flags [Function]

If all of the non-zero bits in a are also non-zero in b then return t (TRUE), otherwise nil (FALSE) (i.e., checks the condition (a and b = b)). This function may be used to interrogate the flags returned by geometric functions.

Arguments:

- **a**

  Flag to compare.

- **b**

  Bits to compare.

Example 9-98. Test-flags example

```
> (test-flags 6 2)
;; returns : T
> (test-flags 6 1)
;; returns : NIL
```

compute-line/line-fillet-info [Function]

This function is used to calculate the fillets for two lines. This function returns a list containing the fillet center, the fillet point on line-1, the fillet point on line-2, the fillet angle, and the lines intersection point.

Arguments:

- **line-1-info**

A list of the end points of the line.

- **line-2-info**

  A list of the end points of the line.

- **radius**

  The fillet radius.

Example 9-99. Compute-line/line-fillet-info example

```
(compute-line/line-fillet-info '((0.0 1.0 0.0) (0.0 0.0 0.0)) '((1.0 1.0 0.0)
(2.0 2.0 0.0)) 0.5)
;; returns :
((0.5 1.2071067811865475 0.0) (0.0 1.2071067811865475 0.0)
(0.8535533905932736 0.8535533905932736 0.0) 45.000000000000014 (0.0
4.440892098500626e-16 0.0))
(compute-line/line-fillet-info '((0.0 1.0 0.0) (0.0 0.0 0.0)) '((1.0 1.0 1.0)
(2.0 2.0 2.0)) 0.5)
;; returns :
((0.3535533905932738 0.9659258262890681 0.3535533905932738) (0.0
0.9659258262890681 0.0)
(0.5576775358252053 0.5576775358252052 0.5576775358252053) 54.73561031724533
(0.0 -4.440892098500626e-16 0.0))
```

See Also: arc-tangent-to-two-lines

Trim Functions

split-arc-by-2-points [Function]

This function divides the specified arc into three segments at the given points. If :values? is t, the function returns a list of 3 sub-lists, one sub-list per arc segment. Each sub-list is of the form center, radius, normal, start-vector, and angle. If :values? is nil, it just returns the sub-list of the first segment.

Arguments:

- **point-1**

  Point coordinates on the arc used to split the arc into segments.

- **point-2**

  Point coordinates on the arc used to split the arc into segments.

- **arc-info**

  A list containing the center, radius, normal, start vector, and angle of the arc.

Keyword Arguments:

- **:values?**

  If :values? is t, the function returns a list of 3 sub-lists, one per arc segment. Each sub-list is of the form center, radius, normal, start-vector, and angle. If :values? is nil, it just returns the sub-list of the first segment.

Default Value: nil

Example 9-100. Split-arc-by-2-points example

```
> (split-arc-by-2-points '(0.0 2.0 0.0) '(-2.0 0.0 0.0) '((0.0 0.0 0.0) 2.0
(0.0 0.0 1.0) (1.0 0.0 0.0) 270.0) :values? t)
;; returns :
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 90.0) ((0.0 0.0 0.0) 2.0 (0.0
0.0 1.0) (0.0 2.0 0.0) 90.0)
((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (-1.0 0.0 0.0) 90.0))
> (split-arc-by-2-points '(0.0 1.0 0.0) '(-1.0 0.0 0.0) '((0.0 0.0 0.0) 2.0
(0.0 0.0 1.0) (1.0 0.0 0.0) 270.0) :values? t)
;; returns :
( NIL NIL)
> (split-arc-by-2-points '(0.0 2.0 0.0) '(-2.0 0.0 0.0) '((0.0 0.0 0.0) 2.0
(0.0 0.0 1.0) (1.0 0.0 0.0) 95.0) :values? t)
;; returns :
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 90.0) ((0.0 0.0 0.0) 2.0 (0.0
0.0 1.0) (0.0 2.0 0.0) 90.0)
((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (-1.0 0.0 0.0) -85.0))
```

See Also: split-arc-by-point

split-arc-by-point [Function]

This function divides the specified arc into two arcs at the given point. If :values? is t, the function returns a list of 2 sub-lists, one sub-list per arc segment. Each sub-list is of the form center, radius, normal, start-vector, and angle. If :values? is nil, it just returns the sub-list of the first segment.

Arguments:

- **point**

  Point coordinates on the arc used to split the arc into segments.

- **arc-info**

  A list containing the center, radius, normal, start vector, and angle for the arc.

Keyword Arguments:

- **:values?**

  If :values? is t, the function returns a list of 2 sub-lists, one sub-list per arc segment. Each sub-list is of the form center, radius, normal, start-vector, and angle. If :values? is nil, it just returns the sub-list of the first segment.

  Default Value: nil

Example 9-101. Split-arc-by-point example

```
> (split-arc-by-point '(0.0 2.0 0.0) '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0
0.0 0.0) 270.0) :values? t)
;; returns :
```

```
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 90.0) ((0.0 0.0 0.0) 2.0 (0.0
0.0 1.0) (0.0 1.0 0.0) 180.0))
> (split-arc-by-point '(0.0 1.0 0.0) '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0
0.0 0.0) 70.0) :values? t)
;; returns :
(NIL)
> (split-arc-by-point '(0.0 2.0 0.0) '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0
0.0 0.0) 70.0) :values? t)
;; returns :
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 90.0) ((0.0 0.0 0.0) 2.0 (0.0
0.0 1.0) (0.0 1.0 0.0) -20.0))
```

See Also: split-circle-by-2-points

split-circle-by-2-points [Function]

This function splits the circle into 2 arc segments given 2 points. If :values? is t, the function returns a list of 2 sub-lists, one sub-list per arc segment. Each sub-list is of the form center, radius, normal, start-vector, and angle. If :values? is nil, the sub-list of the first segment is returned.

Arguments:

- **point-1**

  Coordinates of first point on the circle.

- **point-2**

  Coordinates of second point on the circle.

- **circle-info**

  A list containing the center, radius, and normal.

Keyword Arguments:

- **:values?**

  If :values? is t, the function returns a list of 2 sub-lists, one sub-list per arc segment. Each sub-list is of the form center, radius, normal, start-vector, and angle. If :values? is nil, the sub-list of the first segment is returned.

  Default Value: nil

Example 9-102. Split-circle-by-2-points example

```
> (split-circle-by-2-points '(0.0 2.0 0.0) '(-2.0 0.0 0.0) '((0.0 0.0 0.0)
2.0 (0.0 0.0 1.0)) :values? t)
;; returns :
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (0.0 2.0 0.0) 90.0) ((0.0 0.0 0.0) 2.0 (0.0
0.0 1.0) (-2.0 0.0 0.0) 270.0))
> (split-circle-by-2-points '(0.0 1.0 0.0) '(-1.0 0.0 0.0) '((0.0 0.0 0.0)
2.0 (0.0 0.0 1.0)) :values? t)
;; returns :
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (0.0 1.0 0.0) 90.0) ((0.0 0.0 0.0) 2.0 (0.0
0.0 1.0) (-1.0 0.0 0.0) 270.0))
```

See Also: split-arc-by-2-points, split-arc-by-point

trim-arc/arc [Function]

This function is used to obtain information about the operation of trimming an arc by another arc. The function returns a list of arc segment sub-lists. Each sub-lists, one per resulting arc segment, contains the center, radius, normal, start-vector, and angle respectively.

Arguments:

- **trimmed-arc-info**

  The arc to be trimmed, specified as a list of the center, radius, normal, start-vector, and angle respectively.

- **trim-with-info**

  The arc to trim with, specified as a list of the center, radius, normal, start-vector, and angle respectively

Example 9-103. Trim-arc/arc example

```
(trim-arc/arc '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 270.0) '((0.0
3.0 0.0) 2.0 (0.0 0.0 -1.0) (1.0 0.0 0.0) 270.0))
;; returns 3 arc segments:
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 48.59)
 ((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.322 1.5 0.0) 82.82)
 ((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (-0.66 0.75 0.0) 138.59))
(trim-arc/arc '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 270.0) '((0.0
3.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 270.0))
;; returns 2 arc segments:
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 131.40)
 ((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (-0.66 0.75 0.0) 138.59))
```

trim-arc/circle [Function]

This function is used to obtain information about the operation of trimming an arc by a circle. The function returns a list of arc segment sub-lists. Each sub-lists, one per resulting arc segment, contains the center, radius, normal, start-vector, and angle respectively.

Arguments:

- **arc-info**

  The arc to be trimmed, specified as a list of the center, radius, normal, start-vector, and angle respectively.

- **circle-info**

  The circle to trim with specified as a list of the center, radius, and normal respectively.

Example 9-104. Trim-arc/circle example

```
(trim-arc/circle '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 270.0)
'((0.0 3.0 0.0) 2.0 (0.0 0.0 1.0)))
;; returns 3 arc segments:
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 48.59)
 ((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.32 1.5 0.0) 82.82)
 ((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (-0.66 0.75 0.0) 138.59))
```

trim-arc/line [Function]

This function is used to obtain information about the operation of trimming an arc by a line. The function returns a list of arc segment sub-lists. Each sub-lists, one per resulting arc segment, contains the center, radius, normal, start-vector, and angle respectively.

Arguments:

- **arc-info**

  The arc to be trimmed, specified as a list of the center, radius, normal, start-vector, and angle respectively.

- **line-info**

  The line to trim with specified as a list of the start point and the end point respectively.

Example 9-105. Trim-arc/line example

```
(trim-arc/line '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 270.0) '((0.0
0.0 0.0) (0.0 3.0 0.0)))
;; returns 2 arc segments:
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 90.0) ((0.0 0.0 0.0) 2.0 (0.0
0.0 1.0) (0.0 1.0 0.0) 180.0))
(trim-arc/line '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 270.0) '((-1.0
-3.0 0.0) (0.0 3.0 0.0)))
;; returns 3 arc segments:
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 94.81)
 ((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (-0.16 1.99 0.0) 151.44)
 ((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (-0.40 -0.91 0.0) 23.73))
(trim-arc/line '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.0 0.0 0.0) 270.0) '((0.0
0.0 0.0) (0.0 1.0 0.0)))
;; returns no arc segments:
nil
```

trim-circle/arc [Function]

This function is used to obtain information about the operation of trimming a circle by an arc segment. The function returns a list of arc segment sub-lists. Each sub-lists, one per resulting arc segment, contains the center, radius, normal, start-vector, and angle respectively.

Arguments:

- **circle-info**

  The circle to be trimmed, specified as a list of the center, radius, and normal respectively.

- **arc-info**

  The arc to trim with, specified as a list of the center, radius, normal, start-vector, and angle respectively.

Example 9-106. Trim-circle/arc example

```
(trim-circle/arc '((0.0 3.0 0.0) 2.0 (0.0 0.0 1.0)) '((0.0 0.0 0.0) 2.0 (0.0
0.0 1.0) (1.0 0.0 0.0) 270.0))
;; returns 2 arc segments:
(((0.0 3.0 0.0) 2.0 (0.0 0.0 1.0) (1.32 -1.5 0.0) -82.82)
((0.0 3.0 0.0) 2.0 (0.0 0.0 1.0) (-1.32 -1.5 0.0) -277.18))
```

trim-circle/circle [Function]

This function is used to obtain information about the operation of trimming a circle by another circle. The function returns a list of arc segment sub-lists. Each sub-lists, one per resulting arc segment, contains the center, radius, normal, start-vector, and angle respectively.

Arguments:

- **trimmed-circle-info**

  The circle to be trimmed, specified as a list of the center, radius, and normal respectively.

- **trim-with-circle-info**

  The circle to trim with, specified as a list of the center, radius, and normal respectively.

Example 9-107. Trim-circle/circle example

```
(trim-circle/circle '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0)) '((3.0 0.0 0.0) 2.0
(0.0 0.0 1.0)))
;; returns 2 arc segments:
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.5 1.32 0.0) - 82.81)
((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (1.5 -1.32 0.0) -277.18))
(trim-circle/circle '((1.0 1.0 0.0) 2.0 (0.0 0.0 1.0)) '((3.0 0.0 0.0) 2.0
(0.0 0.0 1.0)))
;; returns 2 arc segments:
(((1.0 1.0 0.0) 2.0 (0.0 0.0 1.0) (1.74 0.98 0.0) -112.02)
((1.0 1.0 0.0) 2.0 (0.0 0.0 1.0) (0.25 -1.98 0.0) -247.97))
(trim-circle/circle '((1.0 1.0 0.0) 2.0 (0.0 0.0 1.0)) '((7.0 0.0 0.0) 2.0
(0.0 0.0 1.0)))
;; returns no arc segments:
nil
```

trim-circle/line [Function]

This function is used to obtain information about the operation of trimming a circle by a line. The function returns a list of arc segment sub-lists. Each sub-lists, one per resulting arc segment, contains the center, radius, normal, start-vector, and angle respectively.

Arguments:

- **circle-info**

  The circle to be trimmed, specified as a list of the center, radius, and normal respectively.

- **line-info**

  The line to trim with, specified as a list of the start point and the end point respectively.

Example 9-108. Trim-circle/line example

```
(trim-circle/line '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0)) '((0.0 -3.0 0.0) (0.0
3.0 0.0)))
;; returns 2 arc segments:
(((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0) (0.0 2.0 0.0) 180.0) ((0.0 0.0 0.0) 2.0
(0.0 0.0 1.0) (0.0 -2.0 0.0) 180.0))
(trim-circle/line '((0.0 0.0 0.0) 2.0 (0.0 0.0 1.0)) '((0.0 0.0 0.0) (0.0 3.0
0.0)))
;; returns no arc segments:
nil
```

trim-line/arc [Function]

This function is used to obtain information about the operation of trimming a line by an arc. The function returns a list of segment sub-lists. Each sub-lists, one per resulting segment, is a list of 2 point coordinates.

Arguments:

- **line-info**

  The line to trim, specified as a list containing the end points coordinates of the line.

- **arc-info**

  The arc to trim with, specified as a list containing the center, radius, normal, start vector, and angle respectively.

Keyword Arguments:

- **:values?**

  If it is t, it will return a list of sub-lists of all trimmed segments, otherwise it will return the sub-list of the first segment.

  Default Value: nil

Example 9-109. Trim-line/arc example

```
(trim-line/arc '((0.0 0.0 0.0) (0.0 3.0 0.0)) '((0.0 0.0 0.0) 2.0 (0.0 0.0
1.0) (1.0 0.0 0.0) 270.0) :values? t)
;; returns :
(((0.0 0.0 0.0) (0.0 2.0 0.0)) ((0.0 2.0 0.0) (0.0 3.0 0.0)))
(trim-line/arc '((-1.0 -3.0 0.0) (0.0 3.0 0.0)) '((0.0 0.0 0.0) 2.0 (0.0 0.0
1.0) (1.0 0.0 0.0) 270.0) :values? t)
;; returns :
(((-1.0 -3.0 0.0) (-0.80 -1.83 0.0))
```

```
((-0.80 -1.83 0.0) (-0.16 1.99 0.0))
((-0.16 1.99 0.0) (0.0 3.0 0.0)))
(trim-line/arc '((0.0 0.0 0.0) (0.0 1.0 0.0)) '((0.0 0.0 0.0) 2.0 (0.0 0.0
1.0) (1.0 0.0 0.0) 270.0) :values? t)
;; returns :
nil
```

trim-line/circle [Function]

This function is used to obtain information about the operation of trimming a line by an arc. The function returns a list of segment sub-lists. Each sub-lists, one per resulting segment, is a list of 2 point coordinates.

Arguments:

- **line-info**

  The line to trim, specified as a list containing the end points coordinates of the line.

- **circle-info**

  The circle to trim with, specified as a list containing the center, radius, and normal respectively.

Keyword Arguments:

- **:values?**

  If it is t, it will return a list of sub-lists of all trimmed segments, otherwise it will return the sub-list of the first segment.

  Default Value: nil

Example 9-110. Trim-line/circle example

```
(trim-line/circle '((0.0 0.0 0.0) (0.0 3.0 0.0)) '((0.0 0.0 0.0) 2.0 (0.0 0.0
1.0)) :values? t)
;; returns :
(((0.0 0.0 0.0) (0.0 2.0 0.0))((0.0 2.0 0.0) (0.0 3.0 0.0)))
(trim-line/circle '((-1.0 -3.0 0.0) (0.0 3.0 0.0)) '((0.0 0.0 0.0) 2.0 (0.0
0.0 1.0)) :values? t)
;; returns :
(((-1.0 -3.0 0.0) (-0.80 -1.83 0.0))
((-0.80 -1.83 0.0) (-0.16 1.99 0.0))
((-0.16 1.99 0.0) (0.0 3.0 0.0)))
(trim-line/circle '((0.0 0.0 0.0) (0.0 1.0 0.0)) '((0.0 0.0 0.0) 2.0 (0.0 0.0
1.0)) :values? t)
;; returns :
nil
```

trim-line/line [Function]

This function is used to obtain information about the operation of trimming a line by another line. The function returns a list of segment sub-lists. Each sub-lists, one per resulting segment, is a list of 2 point coordinates

Arguments:

- **trimmed-line-info**

  The line to trim, specified as a list containing the end point coordinates of the line.

- **trim-with-line-info**

  The line to trim with, specified as a list containing the end point coordinates of the line.

Keyword Arguments:

- **:values?**

  If it is t, it will return a list of sub-lists of trimmed segments, otherwise it will return the sub-list of the first segment.

  Default Value: nil

Example 9-111. Trim-line/line example

```
(trim-line/line '((0.0 0.0 0.0) (10.0 10.0 0.0)) '((3.0 0.0 0.0) (3.0 10.0
0.0)) :values? t)
;; returns :
(((0.0 0.0 0.0) (2.99 2.99 0.0))
((2.99 2.99 0.0) (10.0 10.0 0.0)))
(trim-line/line '((0.0 0.0 0.0) (10.0 10.0 0.0)) '((3.0 0.0 0.0) (3.0 2.0
0.0)) :values? t)
;; returns :
nil
```

Data Structure Functions

List Functions

append [Function]

The append function will concatenate its arguments (lists) into a single list. The original arguments are not affected by the operation.

Arguments:

- **arguments**

  Any number of expressions that result in a list.

Example 9-112. Append example

```
> (APPEND '(a b c) '(1 2 3))
;; returns (A B C 1 2 3)
```

See Also: append-list

append-list [Function]

Append-list takes a list of lists and appends them. This removes one level of lists.

Arguments:

- **list**

  A list of lists.

Example 9-113. Append-list example

```
(append-list '((1.0 2.0) (3.0 4.0 5.0) ((6.0 7.0) 8.0)))
;; returns :
(1.0 2.0 3.0 4.0 5.0 (6.0 7.0) 8.0)
```

See Also: append

butlast [Function]

The butlast function creates a copy of a list that includes all elements except for the last n elements. If n is not specified, it will return all but the last element in the list.

Arguments:

- **list**

  Any list of elements.

Optional Arguments:

- **n**

  An integer defining how many elements to exclude from list.

  Default Value: 1

Example 9-114. Butlast example

```
> (butlast '(a b c d))
;; returns : (a b c)
> (butlast '(a b c d) 3)
;; returns : (a)
> (butlast '((a b) (c d)))
;; returns : ((a b))
> (butlast '(a))
;; returns : NIL
```

See Also: rest

copy-list [Function]

The copy-list function creates a new copy of a list that is equal to the original list. This is useful when using destructive functions: To avoid the destruction of the original list, the developer can copy the list first, using copy-list, and apply destructive functions on the copied list.

Arguments:

- **list**

  Any expression that returns a list.

Example 9-115. Copy-list example

```
(copy-list '(1 2 3))
;; returns : (1 2 3)
(let ((list '(a b c)) )
      (equal list (copy-list list)) )
;; sets (a b c) = (1 2 3)
(let ((list '(a b c)) )
        (eq list (copy-list list)) )
;; returns nil, because it's equal to original, not eq to it
```

See Also: copy-seq

divide-list [Function]

This function divides the list into sublists with the maximum number of elements n.

Arguments:

- **list**

  The list to be divided.

- **n**

  The maximum number of elements in a sublist.

Example 9-116. Divide-list Example

```
(divide-list '(1 2 3 4 5 6 7 8 9 10) 3)
;;returns ((1 2 3) (4 5 6) (7 8 9) (10))
```

first [Function]

Returns the first element of a list.

Arguments:

- **list**

  A list of any elements.

Example 9-117. First example

```
(first '(a b c)
;; returns : a

Note : Also defined for accessing elements in a list are the following
functions.
```

```
second
third
fourth
fifth
sixth
seventh
eighth
ninth
tenth
```

See Also: butlast, last, nth, rest

intersection [Function]

This function returns the set intersection of two list. The elements of the intersection list returned are not necessarily in order of their occurence in the original lists.

Arguments:

- **list1**

  First list.

- **list2**

  Second list.

Keyword Arguments:

- **:key**

  An optional keyword specifying a function to be called on each element of the lists. The value returned by the key is used for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

- **:test**

  Optional keyword to control how the elements are compared to determine equality.

Example 9-118. Intersection example

```
(intersection '(1 2 3) '(2 4 5 3 2 6))
;; returns (3 2)
(intersection '(("string1" 1) ("string3" 3) ("string4" 4) ("string6" 6))
         '(("string3" 3) ("string6" 5) ("string6" 6)) :key 'first :test
'string=)
;; returns : (("string6" 6) ("string3" 3))
```

last [Function]

Returns the last element of a list. The returned value is a list of length 1, it contains the last element. If the list is empty, nil is returned. If the optional argument is added, the function returns a list of all elements after the nth element specified.

Arguments:

- **list**

  A list of any elements.

Optional Arguments:

- **n**

  The integer defining how many elements from the start of the list to exclude, 0 being the first.

Example 9-119. Last example

```
> (last '(a b c))
;; returns : (c)
> (last '((a 1) (b 2)))
;; returns : ((b 2))
> (last (list (cons 'a 1) (cons 'b 2)))
;; retunrs : ((b . 2))
> (last '(1 2 3 4 5) 2)
;; returns : (4 5)
```

See Also: butlast, first, nth, rest

list [Function]

The list function evaluates each of its arguments and collects the result of each evaluation into a list. This is different from using the quote character (') to create a list because the quote does not evaluate the atoms of the list.

Arguments:

- **atoms**

  Any number of items to be collected into a single list

Example 9-120. List example

```
> (list 1 2 3)
;; returns : (1 2 3)
> (list 'a 'b 'c)
;; returns (a b c)
> (list (+ 1 2) (list 'a 'b))
;; returns : (3 (A B))
> (let ((a 1) (b 2))
    (list a 'a b 'b) )
;; returns : (1 A 2 B)
```

list-to-vector [Function]

This function converts a list to a vector.

Arguments:

- **list**

  The list to be converted.

Example 9-121. List-to-vector example

```
> (list-to-vector (list 1 2 3))
;; returns : #<1 2 3>
```

See Also: vector-to-list

member [Function]

Arguments:

- **item**

  The item to search for in the list.

- **list**

  The list of items to be searched.

Keyword Arguments:

- **:test**

  An optional keyword to control how the elements are compared to determine equality.

- **:key**

  An optional keyword specifying a function to be called on each element of the list. The value returned by the key is used for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

Example 9-122. Member example

```
(member 'e '(a b c d))
;; returns : nil
(member "jim" '("Bob" "Jim" "Steve"))
;; returns : nil
(member "jim" '("Bob" "Jim" "Steve") :test 'string-equal)
;; returns : ("Jim" "Steve")
(member 'b '((a 1) (b 2) (c 3)))
;; returns : nil
(member 'b '((a 1) (b 2) (c 3)) :key 'first)
;; returns : ((b 2) (c 3))
```

See Also: position

nth [Function]

Returns the nth element from a list. The first element is the 0th. If the length of the list is less than the requested element, the result is nil.

Arguments:

- **n**

  The element number to return from the list.

- **list**

  A list of elements.

Example 9-123. Nth example

```
> (nth 0 '(a b c))
;; returns : a
> (nth 1 '(a b c))
;; returns : b
> (nth 1 (list (cons 1 2) (cons 3 4)))
;; returns : (3 . 4)
> (nth 3 '(a b c))
;; returns : nil
```

See Also: butlast, elt, first, last, rest

pop [Function]

The pop function removes the first element of a list. Because pop is destructive the list must be an object that evaluates to a list.

Arguments:

- **list**

  An expression that returns a list.

Example 9-124. Push example

```
> (let ((var '(a b c d)))
        (print (pop  var)) var )
;; returns  a   and the new var list : (b c d)
```

See Also: push

push [Function]

The push function places a new element onto the front of a list. Because push is destructive the list must be an object that evaluates to a list.

Arguments:

- **item**

The new element to be placed onto the front of the list.

- **list**

  An expression that returns a list.

Example 9-125. Push example

```
> (let ((var '(b c d)))
        (push 'a var) var )
;; returns the new var list : (a b c d)
```

See Also: pushnew

pushnew [Function]

The pushnew function is similar to push except that only elements that are not already in the list are pushed onto the list.

Arguments:

- **item**

  The new element to be placed onto the front of the list.

- **list**

  An expression that returns a list.

Example 9-126. Pushnew example

```
(let ((var '(b c d)) )
        (pushnew 'a var)
        (pushnew 'b var)
        var )
;; returns : (a b c d)
;; b was not pushed into the list because it was already in the list.
```

See Also: push

remove-duplicates [Function]

The remove-duplicates function compares each item to every other item in the sequence. When an item matches another the first occurrence is removed unless the :from-end parameter is set as t. Because remove-duplicates does not modify the original sequence it is necessary to set the returned result to something.

Arguments:

- **sequence**

  Any expression that results in a sequence have item removed from.

Keyword Arguments:

- **:from-end**

  An optional keyword when used with a t value the search will start from the end of the list.

- **:test**

  An optional keyword to control how the elements are compared to determine equality.

- **:start**

  An optional keyword defining at which index to start the search in the sequence.

- **:end**

  An optional keyword defining at which index to end the search in the sequence.

- **:key**

  An optional keyword specifying a function to be called on each element of the sequence. The value returned by the key is used for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

Example 9-127. Remove-duplicates example

```
(remove-duplicates '(a b a b c b))
;; returns : (a c b)
(remove-duplicates '(a b a b c b) :from-end t)
;; returns : (a b c)
(remove-duplicates '((a 1) (b 2) (a 2)) :key 'second)
;; returns : ((a 1) (a 2))
(remove-duplicates '((a 1) (b 2) (a 2)) :key 'first)
;; returns : ((b 2) (a 2))
(remove-duplicates '(("a" 1) ("b" 2) ("a" 2)) :key 'first :test 'string-
equal)
;; returns : (("b" 2) ("a" 2))
(remove-duplicates '(("a" 1) ("b" 2) ("a" 2)) :key 'first :test 'string-equal
:from-end t)
;; returns : (("a" 1) ("b" 2))
```

See Also: remove

rest [Function]

Returns a list excluding the first element of the list.

Arguments:

- **list**

  A list of elements.

Example 9-128. Rest example

```
> (rest '(a b c))
;; returns : (b c)
```

See Also: butlast, first, last, nth

set-difference [Function]

This function returns a list resulting from a set difference operation between two lists. It will not necessarily return the elements of the difference list in the order of their occurrence in the original lists.

Arguments:

- **list1**

  List to subtract from.

- **list2**

  List to subtract.

Keyword Arguments:

- **:test**

  Keyword changes how function determines list element equality.

- **:key**

  Keyword specifies a function to call on each element in the lists. set-difference uses this function's return value for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

Example 9-129. Set-difference example

```
(set-difference '(("string1" 1) ("string3" 3) ("string4" 4) ("string6" 6))
        '(("string3" 3) ("string6" 5) ("string6" 6)) :key 'first :test
'string=)
;; returns : (("string4" 4) ("string1" 1))
(set-difference '(3 4 5) '(5 4))
;; returns : (3)
(set-difference '(4 5) '(5 3 4))
;; returns : nil
```

set-exclusive-or [Function]

This function returns a list resulting from a set exclusive-or operation between two lists. In other words, the list returned contains all elements excluding the ones that exist in both lists. The elements in the list returned are not necesarily in order of their occurence in the original lists.

Arguments:

- **list1**

  First list.

- **list2**

  Second list

Keyword Arguments:

- **:test**

  Optional keyword to control how the elements are compared to determine equality.

- **key**

  An optional keyword specifying a function to be called on each element of the lists. The value returned by the key is used for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

Example 9-130. Set-exclusive-or example

```
(set-exclusive-or '(1 2 3) '(3 4 5 6))
;; returns : (6 5 4 2 1)
(set-exclusive-or '(("label1" 1) ("label2" 2)) '(("label1" a)
       ("label3" c)) :test 'string= :key 'first)
;; returns : (("label3" C) ("label2" 2))
;; it only compares the firs string, that's why label1 is avoided
```

sort [Function]

The sort function is used to put a sequence in some determined order. The original sequence is destructively modified; therefore, if the sequence to be sorted is a property value, it is recommended to copy the sequence (using copy-seq) into a variable before sorting it.

Arguments:

- **sequence**

  The sequence to be sorted

- **predicate**

  Name (symbol) of a the predicate function. The predicate function is a function that takes two arguments and returns a true or nil value to determine the order of items in the sequence. Note that in case the predicate function requires more than two arguments, the value of this keyword can be a list of the form '(function-name argument3 argument4 ...).

Keyword Arguments:

- **:key**

  An optional keyword specifying a function to be called on each element of the sequence before this element is compared with another element. The value returned by the key function will be used for comparison instead of the element itself. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

Example 9-131. Sort example

```
(sort '(a c d b e) 'string<)
;; returns in alphabetical order : (a b c d r)
(sort '(3 6 4 2 5 1) '<)
```

```
;; returns in ascending order : (1 2 3 4 5 6)

;; Define a predicate function that sums the values in
;; two lists and returns a true value if the first sum is larger:
(defun sum> (list1 list2) ('> (loop for value in list1 sum value)
     (loop for value in list2 sum value) ))
;; defines sum>
;; Now use sum> to sort a list
(sort '((2 3) (1 7) (2 4)) 'sum'>)
;; returns : ((1 7) (2 4) (2 3))
(sort '((2 1 2) (1 5) (2 4 2)) 'sum>)
;; returns : ((2 4 2) (1 5) (2 1 2))
(sort '((2 "two") (3 "three") (1 "one")) '< :key 'first)
;; returns : ((1 "one") (2 "two") (3 "three"))

;; Using a predicate function that take more than two arguments:
;; The function roughly'< returns true if v1 is less than v2 given a
tolerance value
;; (The function returns true if v1 '< v2 and the difference is more then the
tolerance value)
(defun roughly'< (v1 v2 tolerance)
 (and
  ('< v1 v2)
  (not (roughly-equal v1 v2 tolerance)))
)
;; Use roughly'< to sort as follows:
(sort '(1 5 4 2.3 2 4.3 4.9)  '(roughly'< 0.5))
```

stable-sort [Function]

The sequence is destructively sorted according to an order determined by the predicate. The predicate should take two arguments, and return non-nil if and only if the first argument is strictly less than the second. If the first argument is greater than or equal to the second then the predicate should return nil. The function determines the relationship between two elements by giving keys extracted from the elements to the predicate. The :key argument, when applied to an element, should return the key for that element. The difference between function stable-sort and sort is the following: for function sort, elements considered "equal" by the predicate may or may not stay in their original order, but the function stable-sort guarantees the equivalent elements stay in their original order.

Arguments:

- **sequence**

  The sequence to be sorted

- **predicate**

  Name (symbol) of a the predicate function. The predicate function is a function that takes two arguments and returns a true or nil value to determine the order of items in the sequence. Note that in case the predicate function requires more than two arguments, the value of this keyword can be a list of the form '(function-name argument3 argument4 ...).

Keyword Arguments:

- **:key**

  An optional keyword specifying a function to be called on each element of the sequence before this element is compared with another element. The value returned by the key function will be used for comparison instead of the element itself. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

Example 9-132. Stable-sort example

```
(setf l '(1 2 3 4 5 6 7 8 9 0))
;;;(1 2 3 4 5 6 7 8 9 0)
(defun ff (x y) (and (oddp x) (evenp y)))
(sort l 'ff)
;;;(1 9 3 7 5 0 4 8 2 6)
(setf l '(1 2 3 4 5 6 7 8 9 0))
;;;(1 2 3 4 5 6 7 8 9 0)
(stable-sort l 'ff)
;;;(1 3 5 7 9 2 4 6 8 0)
```

subsetp [Function]

This function returns t if every element of a given list (list1) is a member of another given list (list2). The function returns nil otherwise.

Arguments:

- **list1**

  List of elements to test for.

- **list2**

  List that should contains all elements of list1 for the function to return a non-nil value.

Keyword Arguments:

- **:test**

  Optional keyword to control how the elements are compared to determine equality.

  Default Value:

- **:key**

  An optional keyword specifying a function to be called on each element of the lists. The value returned by the key is used for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

Example 9-133. Subsetp example

```
(subsetp '(1 2) '(5 3 1 7 2))
;; returns : t
```

```
(subsetp '((a 0) (c 2)) '((alpha 0) (beta 1) (gamma 2)) :key 'second :test
'=)
;; returns : T
;; it compares the numbers, not the strings
```

substitute [Function]

This function substitutes one item for another item in a sequence and returns the resulting sequence. The operation is not destructive.

Arguments:

- **new-item**

  New item to place in the sequence.

- **old-item**

  Item to be substituted by new-item.

- **sequence**

  The sequence in which to perform the substitution.

Keyword Arguments:

- **:from-end**

  If t, performs substitution starting from end of sequence.

  Default Value: nil

- **:test**

  Function symbol that compares elements and determines equality.

- **:test-not**

  Function symbol that compares elements and determines inequality.

- **:start**

  Numeric index specifies where to start substituting items.

- **:end**

  Numeric index specifies where to stop substituting items.

- **:count**

  Number of substitutions to perform. When nil, substitutes all matching items.

  Default Value: nil

- **:key**

  Function symbol called on each element. This function's return value is used for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

Example 9-134. Substitute example

```
(substitute 10 13 '(2 3 4 5 13 15 13 16))
;; returns : (2 3 4 5 10 15 10 16)
(substitute 10 13 '(2 3 4 5 13 15 13 16) :count 1)
;; returns : (2 3 4 5 10 15 13 16)
(substitute 10 13 '(2 3 4 5 13 15 13 16) :test '>)
;; returns : (10 10 10 10 13 15 13 16)
(substitute 10 13 '(2 3 4 5 13 15 13 16) :test '<)
;; returns : (2 3 4 5 13 10 13 10)
(substitute 10 13 '(2 3 4 5 13 15 13 16) :start 5)
;; returns: (2 3 4 5 13 15 10 16)
```

transpose-2d-list [Function]

This function transposes the elements of a 2-dimensional list (list of lists) and returns the transposed list.

Arguments:

- **matrix**

  This is a list (dimension 1) of lists (dimension 2).

- **m**

  The number of elements of dimension 1, which is the number of elements in dimension 2 of the transposed matrix.

- **n**

  The number of elements of dimension 2, which is the number of elements in dimension 1 of the transposed matrix.

Example 9-135. Transpose-2d-list example

```
(transpose-matrix '((1 2 3) (a b c)) 2 3)
;;returns  ((1 a) (2 b) (3 c))
```

union [Function]

Returns the set union of two lists. The elements returned are not necessarily in order of thier occurrance in list1 or list2.

Arguments:

- **list1**

  First list.

- **list2**

  Second list.

Keyword Arguments:

- **:test**

Function symbol that compares elements and determins equality.

- **:key**

  Function called on each element. This function's return value is used for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

Example 9-136. Union example

```
> (union '(1 2 3) '(5 6 2 7 3))
;; returns (1 5 6 2 7 3)
> (union '(("string1" 1) ("string3" 3) ("string4" 4) ("string6" 6))
'(("string3" 3) ("string6" 5) ("string6" 6)) :key 'first :test 'string=)
;; returns :
(("string4" 4) ("string1" 1) ("string3" 3) ("string6" 5) ("string6" 6))
```

Vector Functions

aref [Function]

The function returns the element of a vector given a position index. The function leads to an error if the index given is out of range.

Arguments:

- **vector**

  A vector.

- **index**

  An integer starting at 0 specifying the position of the desired element.

Example 9-137. Aref example

```
> (aref (vector 2 3 4) 0)
;; returns : 2
```

See Also: vector

create-adjustable-vector [Function]

This function creates an empty vector with the specified length, expand-size defines the additional size when the vector needs to be expanded.

Arguments:

- **length**

  The length of the vector.

Keyword Arguments:

- **expand-size**

  The additional size to expand.

empty-vector [Function]

Removes the contents of a vector. The vector must be a vector with a fill pointer. It modifies the original vector in the process.

Arguments:

- **vector**

  A vector

Example 9-138. Empty-vector example

```
> (setf vector (make-vector 1 2 3 4 5 6 7))
;; creates a vector
> vector
;; returns : #<1 2 3 4 5 6 7>
> (empty-vector vector)
;; removes everything from the vector
> vector
;; retunrs : #<>
```

vector [Function]

This function is used to create a simple vector.

Arguments:

- **objects**

  An arbitrary number of elements to be used as the contents of the vector.

Example 9-139. Vector example

```
> (vector 1.0 2.0 3.0)
;; returns : #<1.0 2.0 3.0>
> (vector 1 2 3)
;; returns : #<1 2 3>
```

See Also: list-to-vector

vector-append [Function]

Takes an arbitrary number of vectors and turns them into a single vector.

Arguments:

- **vectors**

  An arbitrary number of vectors.

Example 9-140. Vector-append example

```
> (vector-append (vector 1 2 3) (vector 4 5 6) (vector 7 8 9) (vector 10))
;; returns : #<1 2 3 4 5 6 7 8 9 10>
```

vector-delete-at-index [Function]

This function deletes the item from the vector at the specified index.

Arguments:

- **index**

  The index of the item to delete.

- **vector**

  A vector containing the item to be deleted.

See Also: vector-insert-at-index

vector-insert-at-index [Function]

This function inserts the item to the vector at the specified index.

Arguments:

- **item**

  The item to be inserted.

- **index**

  The index of the item to insert.

- **vector**

  A vector where the item is to be inserted.

See Also: vector-delete-at-index

vector-pop [Function]

Returns the last element of the vector while modifying the original vector to remove the last element. It modifies the original vector in the process.

Arguments:

- **vector**

  A vector with a fill pointer.

Example 9-141. Vector-pop example

```
> (setf a (make-vector 1 2 3))
;; sets a to be a vector
> a
;; returns the element of a : #<1 2 3>
> (vector-pop a)
```

```
;; pops : 3
> a
;; returns the new a : #<1 2>
```

See Also: vector-push, vector-push-new

vector-push [Function]

Returns the vector with the item supplied pushed onto the end of the vector. It modifies the original vector in the process.

Arguments:

- **item**

  The item to push onto the vector.

- **vector**

  A vector with a fill pointer.

Example 9-142. Vector-push example

```
(setf a (make-vector 1 2 3))
;; sets a to be a vector
a
;; returns a : #<1 2 3>
(vector-push 4 a)
;; pushes 4 at the end of a
 a
;; returnsh the new a : #<1 2 3 4>
```

See Also: vector-pop, vector-push-new

vector-push-new [Function]

If the item specified is not currently in the vector, it will be added to the end of the vector. This function modifies the original vector.

Arguments:

- **item**

  The item to push onto the vector.

- **vector**

  A vector with a fill pointer.

Example 9-143. Vector-push-new example

```
(setf a (make-vector 1 2 3))
;; sets a to be a vector
a
;; returns a : #<1 2 3>
```

```
(vector-push-new 3 a)
;; a stays the same because 3 is already in it
a
;; returns a : #<1 2 3>
(vector-push-new 4 a)
;; adds 4 at the end of a
a
;; returns a : #<1 2 3 4>
```

See Also: vector-pop, vector-push

vector-to-list [Function]

This function converts a vector to a list.

Arguments:

- **vector**

  The vector to be converted.

Example 9-144. Vector-to-list example

```
> (vector-to-list (vector 1 2 3))
;; returns a list : (1 2 3)
```

See Also: list-to-vector

Sequence Functions

A sequence is a generic type. It can be a list, a vector or a string. In AML lists and strings are considered sequrences. All functions defined in this section work on both lists and strings.

copy-seq [Function]

The copy-seq function makes and returns a copy of a given sequence. The new sequence is a copy of the old so that destructive operations on the new sequence do not affect the original list.

Arguments:

- **sequence**

  Any expression that returns a sequence.

Example 9-145. Copy-seq example

```
> (copy-seq '(1 2 3))
;; returns : (1 2 3)
> (copy-seq "abc")
;; returns : "abc"
```

See Also: copy-list

count [Function]

The count function returns the number of instances of the desired element in the sequence.

Arguments:

- **element**

  Element of interest

- **sequence**

  Any expression that returns a sequence.

Example 9-146. Count example

```
> (count 1 '(1 2 3 4 5 1 6 7)
;; returns : 2
> (count "Hello" '("abc" "Hello"  "Hello World" "Hello ")
;; returns : 1
```

elt [Function]

The elt function returns the given nth element from a sequence. This is similar to nth except that nth only operates on lists (note that the order of the arguments is reversed). If the length of the list is less than the requested element the result is nil. The first element is numbered 0 and count up from there.

Arguments:

- **sequence**

  Any sequence.

- **index**

  The index number of the item that is to be returned.

Example 9-147. Elt example

```
> (elt '(a b c) 0)
;; returns : a
> (elt '(a b c) 2)
;; returns : c(elt "abcdef" 2)
> (elt "abcdef" 2)
;; returns : #\c
```

See Also: nth

find [Function]

The find function searches a sequence for an element that satisfies the test. If a match is found the item is returned otherwise nil is returned.

Arguments:

- **item**

The item to search for in sequence.

- **sequence**

  Sequence to look in.

Keyword Arguments:

- **:from-end**

  An optional keyword when used with a t value the search will start from the end of the list.

- **:test**

  An optional keyword to control how the elements are compared to determine equality.

- **:start**

  An optional keyword defining at which index to start the search in the sequence.

- **:end**

  An optional keyword defining at which index to end the search in the sequence.

- **:key**

  An optional keyword specifying a function to be called on each element of the sequence. The value returned by the key is used for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

Example 9-148. Find example

```
(find "a" "bacd" :test 'string-equal)
;; returns : "a"
(find 'b '((a 1) (b 2) (c 3)) :key 'first)
;; returns : (b 2)
(find "a" '(("a" 1) ("b" 2) ("A" 2)) :key 'first :test 'string-equal :from-
end t)
;; returns : ("a" 2)
(find "a" '(("a" 1) ("b" 2) ("A" 2)) :key 'first :test 'string-equal)
;; returns : ("a" 1)
(find "A" '(("a" 1) ("b" 2) ("A" 2)) :key 'first :test 'string=)
;; returns : ("a" 2)
(find "A" '(("a" 1) ("b" 2) ("A" 2)) :key 'first :test 'string=)
;; returns : ("a" 2)
```

See Also: member, position

length [Function]

The length function returns the length of a given sequence. The sequence may be any sequence such as a list or a string.

Arguments:

- **sequence**

Any expression that returns a sequence.

Example 9-149. Length example

```
> (length '(1 2 3))
;; returns : 3
> (length "abc")
;; returns : 3
> (length '((1 2) (3 4)))
;; returns : 2
> (length "(a b c)")
;; returns : 7 - which is the characters + spaces inside " "
```

position [Function]

The position function searches a sequence for an element that satisfies the test. If a match is found the index number for the position in the sequence is returned otherwise nil is returned. The index of the first element is zero.

Arguments:

- **item**

  The item to search for in the sequence.

- **sequence**

  The sequence of item to be searched.

Keyword Arguments:

- **:test**

  An optional keyword to control how the elements are compared to determine equality.

- **:key**

  An optional keyword specifying a function to be called on each element of the sequence. The value returned by the key is used for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

Example 9-150. Position example

```
(position 'b '(a b c d))
;; returns : 1
(position 'b '("a" "b" "c"))
;; returns : NIL
(position 'b '("a" "b" "c") :test 'string-equal)
;; returns : 1
(position 'b '((a 1) (b 2) (c 3)) :key 'first)
;; returns : 1
```

See Also: find

remove [Function]

The remove function creates a new sequence with a specified item removed. Because remove does not modify the original sequence it is necessary to set the result to something.

Arguments:

- **item**

  The item to be removed from sequence.

- **sequence**

  Any expression that results in a sequence to have item removed from.

Keyword Arguments:

- **:from-end**

  An optional keyword when used with a t value the search will start from the end of the list.

- **:test**

  An optional keyword to control how the elements are compared to determine equality.

- **:start**

  An optional keyword defining at which index to start the search in the sequence.

- **:end**

  An optional keyword defining at which index to end the search in the sequence.

- **:count**

  An optional keyword defining how many instances of item to remove.

  Default Value: Remove all.

- **:key**

  An optional keyword specifying a function to be called on each element of the sequence. The value returned by the key is used for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

Example 9-151. Remove example

```
(remove 3 '(1 3 4 3))
;; returns : (1 4)
(remove 'b '(a b c d))
;; returns : (A C D)
(remove 3 '(1 3 4 3) :from-end t :count 1)
;; removes only 1 3 from the end and returns : (1 3 4)
(remove 3 '(1 2 3 4 5 6) :test '>)
;; removes smaller thant 3 and returns : (3 4 5 6)
(remove 3 '((a 1) (b 2) (c 3)) :key 'second)
;; removes (c 3) and returns : ((a 1) (b 2))
(let (list '(1 2 3)) (setf list (remove 2 list)) list )
;; returns : (1 3)
```

See Also: remove-duplicates

replace [Function]

This function replaces some elements of a sequence (seq1) by the elements of another sequence provided (seq2) and returns the resulting sequence. seq1 is modified by copying successive elements into it from seq2. This function is destructive, so seq1 needs to be copied if its original form is needed later on.

Arguments:

- **seq1**

  sequence to modify

- **seq2**

  sequence to mofify seq1 with.

Keyword Arguments:

- **:start1, :end1**

  Indices specifying the subsequence of seq1 to modify.

  Default Value: :start1 defaults to 0 and :end1 defaults to nil meaning the end of the sequence.

- **:start2, :end2**

  Indices specifying the subsequence of seq2 to use.

  Default Value: :start2 defaults to 0 and :end2 defaults to nil meaning the end of the sequence.

Example 9-152. Replace example

```
> (let ((l1 '(1 2 3 4 5)) (l2 '(a b))) (replace l1 l2 :start1 2) l1)
;; returns : (1 2 A B 5)
> (let ((l1 "12345") (l2 "ab")) (replace l1 l2 :start1 2) l1)
;; returns : "12ab5"
```

reverse [Function]

The reverse function creates and returns a new sequence with the elements in the reverse order of the original sequence.

Arguments:

- **sequence**

  Any expression that returns a sequence.

Example 9-153. Reverse example

```
> (reverse '(1 2 3))
;; returns : (3 2 1)
> (reverse "abc")
```

```
;; returns : "cba"
> (reverse '((a 1) (b 2)))
;; returns : ((b 2) (a 1))
```

search [Function]

Searches for the occurence of a sequence within anotther sequence. A search is conducted for a subsequence of seq2 that matches seq1 element-wise. The value returned is the index of the leftmost element into sequence2 of the leftmost matching sequence. Nil is returned if no match is found.

Arguments:

- **seq1**

  sequence to search for in seq2.

- **seq2**

  sequence to search in.

Keyword Arguments:

- **:from-end**

  An optional keyword when used with a t value the search will start from the end of the list.

- **:test**

  An optional keyword to control how the elements are compared to determine equality.

- **:start**

  An optional keyword defining at which index to start the search in the sequence.

- **:end**

  An optional keyword defining at which index to end the search in the sequence.

- **:key**

  An optional keyword specifying a function to be called on each element of the sequence. The value returned by the key is used for element comparison. If the function takes additional arguments, a list can be used in the following format: (function additional-arg1 additional-arg2 ....).

  Default Value: Element as is.

- **:start1, :end1**

  subsequence of seq1 to consider. The element whose index is :end1 will not be included in the subsequence to consider. In other words to get the first 2 elements of a sequence for example, :start1 should be 0 and :end1 should be 2.

  Default Value: :start1 defaults to 0 and :end1 defaults to nil meaning the end of the sequence.

- **:start2, :end2**

  subsequence of seq2 to consider. The element whose index is :end2 will not be included in the subsequence to consider. In other words to get the first 2 elements of a sequence for example, :start2 should be 0 and :end2 should be 2.

  Default Value: :start2 defaults to 0 and :end2 defaults to nil meaning the end of the sequence.

Example 9-154. Search example

```
(search "23" "1234")
;; returns : 1
(search "23" "1234" :start2 2)
;; returns : NIL
(search '(("a")) '(("b" 2) ("a" 1) ("c" 3)) :key 'first :test 'string=)
;; returns : 1
```

subseq [Function]

The subseq function returns the subsequence (also known as a substring if the sequence is a string) of a sequence specified by start and end. The result is a new sequence that is contains only the subsequence.

Arguments:

- **sequence**

  Any expression that returns a sequence.

- **start**

  The index number of the start of the subsequence to be returned. The element specified by the start is included in the subsequence.

- **end**

  The optional index number of the end of the subsequence to be returned. If not specified, the end of the sequence is assumed. When specified the element in the end position is not included in the list.

Example 9-155. Subseq example

```
> (subseq '(0 1 2 3 4 5 6) 1)
;; returns : (1 2 3 4 5 6)
> (subseq '(0 1 2 3 4 5 6) 1 2)
;; returns : (1)
> (subseq "abcdefg" 3)
;; returns : "defg"
> (subseq "abcdefg" 3 5)
;; returns : "de"
```

See Also: search

Hash Table Functions

gethash [Function]

This function is used to query a hash table.

Arguments:

- **key**

  The key for the needed elements.

- **hash-table**

  Instance of a hash-table

Optional Arguments:

- **default**

  default is the value returned if no record matching the key is found.

See Also: make-hash-table

make-hash-table [Function]

Returns a newly-created hashtable.

Keyword Arguments:

- **:test**

  Function symbol used for comparing equality of hash table keys. Currently, the only valid values are 'eq, 'eql and 'equal.

- **:size**

  Suggestion of the number of entries expected.

- **:rehash-size**

  When the table expands, the number of entries it should grow.

- **:rehash-threshold**

  Floating point number between 0.0 and 1.0 representing 0% and 100% respectively. Determines how full the table should get before expansion occurs.

See Also: gethash

Example 9-156. Make-hash-table Example

```
;; The function setf is used to populate a hash table :

> (let ((hash-table (make-hash-table :size 200))
     )
 ;; populating hash table
         (setf (gethash 1 hash-table) '(a "A"))
         (setf (gethash 2 hash-table) '(b "B"))
         (setf (gethash 3 hash-table) '(c "C"))
          ...etc
        )
```

remhash [Function]

This function is used to remove an element from a hash table.

Arguments:

- **key**

  The key of the element.

- **hash-table**

  Instance of a hash-table

Example 9-157. Make-hash-table Example

```
;; The function setf is used to populate a hash table :

> (let ((hash-table (make-hash-table :size 200))
     )
 ;; populating hash table
         (setf (gethash 1 hash-table) '(a "A"))
         (setf (gethash 2 hash-table) '(b "B"))
         (setf (gethash 3 hash-table) '(c "C"))
          ...etc
        )

;;; removing the element whose keyword is 2
> (remhash 2 hash-table)
;;; returns t

> (gethash 2 hash-table)
;;; returns nil
```

See Also: make-hash-table gethash

Comparison and Type-Checking Functions

Number Comparison Functions

The following functions compare 2 or more numbers. If any of these functions encounter any non-numeric values an error occurs.

```
= Equal.
/= Not Equal.
< Less Than.
> Greater Than.
<= Less Than or Equal.
>= Greater Than or Equal.
```

Example 9-158. Number Comparioson Functions example

```
(= 1 1.0)
;; returns : t
(let ((a 1) (b 1.0))
   (= a b 1.0))
;; returns : t
(= 3 3.001)
```

```
;; returns : nil
(/= 3 3.001)
;; returns : t
(let ((a 1) (b 1.0))
  (/= a b 1.0))
;; returns : nil
(< 2 3)
;; returns : t
(< 3 7 9)
;; returns : t
(let ((low 2) (hi 10.0))
    (< low 3 hi))
;; returns : t
(let ((low 2) (hi 10.0))
    (< low 10 hi))
;; returns : nil
(> 3 4)
;; returns : nil
(let ((low 2) (hi 10.0))
   (> hi 3 low))
;; returns : t
(let ((low 2) (hi 10.0))
   (> hi 10 low))
;; returns : nil
(let ((low 2) (hi 10.0))
(<= low 10 hi))
;; returns : t
(let ((low 2) (hi 10.0))
(>= hi 10 low))
;; returns : t
```

General Comparison Functions

There are a number of functions for determining equality. From most specific to least specific they are: eq, eql, equal, and equalp. The most commonly used is equal followed by eq. The others may be used but in general are unnecessary.

equal [Function]

This function returns t if the two arguments are the same type and their contents are equal.

Arguments:

- **value1**

  Any expression or value to test.

- **value2**

  Any expression or value to test.

Example 9-159. Equal example

```
> (equal 'a 'a)
;; returns : t
> (equal 5 5)
;; returns : t
> (equal #\a #\a)
;; returns : t
> (equal #\a #\A)
;; returns : nil
> (equal "bob" "bob")
;; returns : t
> (equal "bob" "Bob")
;; returns : nil
> (equal 5 5.0)
;; returns : nil
> (equal 5.0 5.0)
;; returns : t
```

equalp [Function]

The equalp function returns true where equal is true, where numerical values have the same value, and where characters are the same regardless of the case.

Arguments:

- **value1**

  Any expression or value to test.

- **value2**

  Any expression or value to test.

Example 9-160. Equalp example

```
> (equalp 5 5.0)
;; returns : t
> (equalp #\a #\A)
;; returns : t
> (let ((a 1.0))
    (equalp a 1))
;; returns : t
```

typep [Function]

This function returns true when item1 and item2 are in agreement with respect to data type or class type.

Arguments:

- **item1**

  Any value to test.

- **item2**

  Any value to test.

Example 9-161. Typep example

```
> (typep 'asdf 'symbol)
;; returns : t
> (create-model 'name-generator)
;; creates a new name generator model
> (add-object (the) 'box 'box-object)
;; adds a box-object to the name generator
> (typep (the box) 'cylinder-object)
;; returns : nil - not a cylinder
> (typep (the box) 'box-object)
;; returns : t
> (typep (the box) 'object)
;; returns : t
> (inheritance-list (the box))
;; returns the list:
(BOX-OBJECT PRIMITIVE-CLASS SIMPLE-GEOMETRY-CLASS GRAPHIC-OBJECT
POSITION-OBJECT LAYER-OBJECT
SOLID-OBJECT OBJECT OBJECT-ROOT-CLASS ADAPTIVE-CLASS ...)
> (typep (the box) 'SIMPLE-GEOMETRY-CLASS)
;; returns : t
> (typep 'asdf 'string)
;; returns : nil
> (typep "asdf" 'string)
;; returns : t
```

See Also: type-of

String Comparison Functions

string= [Function]

The string= function compares two strings and returns a true value if they are composed of the same sequence of characters and false if not. This is what the function equal calls when given two strings. The function string-equal is similar except that it ignores case differences.

Arguments:

- **string1**

  Any expression that returns a string.

- **string2**

  Any expression that returns a string.

Keyword Arguments:

- **:start1**

  The character in string1 to use as the start of the string1 for comparison. This is inclusive.

- **:end1**

  The character in string1 to use as the end of the string1 for comparison. This is exclusive.

- **:start2**

  The character in string2 to use as the start of the string2 for comparison. This is inclusive.

- **:end2**

  The character in string2 to use as the end of the string2 for comparison. This is exclusive.

Example 9-162. String= example

```
(string= "abc" "abc")
;; returns : t
(string= "ABC" "abc")
;; returns : nil
(string= "def" "abcdef")
;; returns : nil
(string= "def" "abcdef" :start2 3)
;; returns : t
(string= "abc" "abcdef" :end2 3)
;; returns : t
(string= "together" "frog" :start1 1 :end1 3 :start2 2)
;; returns : t
```

See Also: string-equal

string/= [Function]

The string/= function compares two strings and returns t if they are not composed of the same sequence of characters and nil otherwise. The non-nil value being returned is the length of the substring that is common to the beginning of both strings.

Arguments:

- **string1**

  Any expression that returns a string.

- **string2**

  Any expression that returns a string.

Keyword Arguments:

- **:start1**

  The character in string1 to use as the start of the string1 for comparison. This is inclusive.

- **:end1**

  The character in string1 to use as the end of the string1 for comparison. This is exclusive.

- **:start2**

  The character in string2 to use as the start of the string2 for comparison. This is inclusive.

- **:end2**

  The character in string2 to use as the end of the string2 for comparison. This is exclusive.

Example 9-163. String/= example

```
(string/= "abc" "abc")
;; returns : nil
(string/= "ABC" "abc")
;; returns : t
(string/= "def" "abcdef")
;; returns : t
(string/= "def" "abcdef" :start2 3)
;; returns : nil
(string/= "ABC" "ABc")
;; returns : t
```

string-equal [Function]

The string-equal function is the like string= except that it ignores case differences.

Arguments:

- **string1**

  Any expression that returns a string.

- **string2**

  Any expression that returns a string.

Keyword Arguments:

- **:start1**

  The character in string1 to use as the start of the string1 for comparison. This is inclusive.

- **:end1**

  The character in string1 to use as the end of the string1 for comparison. This is exclusive.

- **:start2**

  The character in string2 to use as the start of the string2 for comparison. This is inclusive.

- **:end2**

  The character in string2 to use as the end of the string2 for comparison. This is exclusive.

Example 9-164. String-equal example

```
(string-equal "ABC" "abc")
;; returns : t
(string-equal "DEF" "abcdef")
;; returns : nil
(string-equal "DEF" "abcdef" :start2 3)
;; returns : t
```

See Also: string=

string< [Function]

The string< compares two strings and returns a true value if string1 is less than string2. When a test is true the result is not t but is the length of the substring that is common to the beginning of both strings.

Arguments:

- **string1**

  Any expression that returns a string.

- **string2**

  Any expression that returns a string.

Keyword Arguments:

- **:start1**

  The character in string1 to use as the start of the string1 for comparison. This is inclusive.

- **:end1**

  The character in string1 to use as the end of the string1 for comparison. This is exclusive.

- **:start2**

  The character in string2 to use as the start of the string2 for comparison. This is inclusive.

- **:end2**

  The character in string2 to use as the end of the string2 for comparison. This is exclusive.

Example 9-165. String< example

```
(string< "abced" "abc")
;; returns : nil
(string< "abc" "abced")
;; returns : t
(string< "abc" "ABCED")
;; returns : nil
(string< "abced" "abced")
;; returns : nil
```

string> [Function]

The string> compares two strings and returns true value if string1 is greater than string2. When a test is true the result is not t but is the length of the substring that is common to the beginning of both strings.

Arguments:

- **string1**

  Any expression that returns a string.

- **string2**

  Any expression that returns a string.

Keyword Arguments:

- **:start1**

  The character in string1 to use as the start of the string1 for comparison. This is inclusive.

- **:end1**

  The character in string1 to use as the end of the string1 for comparison. This is exclusive.

- **:start2**

  The character in string2 to use as the start of the string2 for comparison. This is inclusive.

- **:end2**

  The character in string2 to use as the end of the string2 for comparison. This is exclusive.

Example 9-166. String> example

```
(string> "abc" "abced")
;; returns : nil
(string> "abced" "abc")
;; returns : t
(string> "abced" "abced")
;; returns : nil
```

string<= [Function]

The string<= compares two strings and a returns a true value if string1 is less than or equal to string2. When a test is true the result is not t but is the length of the substring that is common to the beginning of both strings.

Arguments:

- **string1**

  Any expression that returns a string.

- **string2**

  Any expression that returns a string.

Keyword Arguments:

- **:start1**

  The character in string1 to use as the start of the string1 for comparison. This is inclusive.

- **:end1**

  The character in string1 to use as the end of the string1 for comparison. This is exclusive.

- **:start2**

  The character in string2 to use as the start of the string2 for comparison. This is inclusive.

- **:end2**

  The character in string2 to use as the end of the string2 for comparison. This is exclusive.

Example 9-167. String<= example

```
(string<= "abced" "abc")
;; returns : nil
(string<= "abc" "abced")
;; returns : t
(string<= "abc" "ABCED")
;; returns : nil
(string<= "abced" "abced")
;; returns : t
```

string>= [Function]

The string>= compares two strings and returns a true value if string1 is greater than or equal to string2. When a test is true the result is not t but is the length of the substring that is common to the beginning of both strings.

Arguments:

- **string1**

  Any expression that returns a string.

- **string2**

  Any expression that returns a string.

Keyword Arguments:

- **:start1**

  The character in string1 to use as the start of the string1 for comparison. This is inclusive.

- **:end1**

  The character in string1 to use as the end of the string1 for comparison. This is exclusive.

- **:start2**

  The character in string2 to use as the start of the string2 for comparison. This is inclusive.

- **:end2**

  The character in string2 to use as the end of the string2 for comparison. This is exclusive.

Example 9-168. String>= example

```
(string>= "abc" "abced")
;; returns : nil
(string>= "abced" "abc")
;; returns : t
(string>= "abced" "abced")
;; returns : t
```

Type-Checking Functions

evenp [Function]

The evenp function returns true if a given value is an even number and nil if not. The number must be an integer or an error occurs.

Arguments:

- **number**

  Any expression that returns a number.

Example 9-169. Evenp example

```
> (evenp 1234)
;; returns : t
> (evenp 133)
;; returns : nil
> (evenp (expt 4 5))
;; returns : t
```

floatp [Function]

The floatp function returns true if a given value is a float and nil if not.

Arguments:

- **value**

  Any expression or value to test.

Example 9-170. Floatp example

```
> (floatp 123.1)
;; returns : t
> (floatp (/ 13 3))
;; returns : t
> (floatp (/ 12 3))
;; returns : nil
> (floatp 1e10)
;; returns : t
> (floatp (round 1e10))
;; returns : nil
```

integerp [Function]

The integerp function returns true if a given value is an integer and nil if not.

Arguments:

- **value**

  Any expression or value to test.

Example 9-171. Integerp example

```
> (integerp 123)
;; returns : t
> (integerp (/ 13 3))
;; returns : nil
> (integerp (/ 12 3))
;; returns : t
> (integerp 1e10)
;; returns : nil
> (integerp (round 1e10))
;; returns : t
```

listp [Function]

The listp function returns true if the given value is a list and nil if not. Note that nil is a valid list (equivalent to an empty list). Use null to check for empty lists.

Arguments:

- **value**

  Any expression or value to test.

Example 9-172. Listp example

```
> (listp '(1 2 3))
;; returns : t
> (listp (list 1 2 3))
;; returns : t
> (listp 1)
;; returns : nil
> (listp (the))
;; returns : nil
> (listp nil)
;; returns : t
```

See Also: null

minusp [Function]

The minusp function returns true if a given value is less than zero and nil if not.

Arguments:

- **number**

  Any expression that returns a number.

Example 9-173. Minusp example

```
> (minusp -1)
;; returns : t
```

```
> (minusp 0)
;; returns : nil
> (minusp 1)
;; returns : nil
```

null [Function]

The null function returns true if the given value is nil or the empty list and nil if not. This will result in exactly the value as the not function but is generally used to show the intent of testing if a list is empty.

Arguments:

- **value**

  Any expression or value to test.

Example 9-174. Null example

```
> (null nil)
;; returns : t
> (null (remove 1 '(1 1)))
;; returns : t
> (null '(1 2 3))
;; returns : nil
```

numberp [Function]

The numberp function returns true if a given value is a number and nil if not.

Arguments:

- **value**

  Any expression or value to test.

Example 9-175. Numberp example

```
> (numberp (expt 4 5))
;; returns : t
AML> (numberp 'number)
;; returns : nil
AML> (numberp "number")
;; returns : nil
AML> (setf a 100)
;; sets a to = 100
AML> (numberp a)
;; returns : t
```

oddp [Function]

The oddp function returns true if a given value is an odd number and nil if not. The number must be an integer or an error occurs.

Arguments:

- **number**

  Any expression that returns a number.

Example 9-176. Oddp example

```
> (oddp 3)
;; returns : t
> (oddp 4)
;; returns : nil
> (oddp (expt 4 5))
;; returns nil
```

plusp [Function]

The plusp function returns true if a given value is a greater than zero and nil if not.

Arguments:

- **number**

  Any expression that returns a number.

Example 9-177. Plusp example

```
> (plusp 0.1)
;; returns : t
> (plusp 0.0)
;; returns : nil
> (plusp -1)
;; returns : nil
```

stringp [Function]

The stringp function returns true if the given value is a string and nil if not.

Arguments:

- **value**

  Any expression or value to test.

Example 9-178. Stringp example

```
> (stringp "mike")
;; returns : t
> (stringp "Mike")
;; returns : t
> (stringp 'Mike)
;; returns : nil
> (stringp :mike)
;; returns : nil
```

```
> (stringp 100)
;; returns : nil
```

symbolp [Function]

The symbolp function returns true if the given value is a symbol and nil if not.

Arguments:

- **value**

  Any expression or value to test.

Example 9-179. Symbolp example

```
> (symbolp "mike")
;; returns : nil
> (symbolp "Mike")
;; returns : nil
> (symbolp 'Mike)
;; returns : t
> (symbolp :mike)
;; returns : t
> (symbolp 100)
;; returns : nil
```

subclassp [Function]

Returns t if class1 is a subclass of class2. Returns nil otherwise.

Arguments:

- **class1**

  A predefined class

- **class2**

  A predefined class

Example 9-180. Subclass example

```
(subclassp 'box-object 'graphic-object)
Returns: t
(subclassp 'box-object 'sphere-object)
Returns: nil
```

type-of [Function]

This function returns the first member of an object's inheritance list.

Arguments:

- **object**

AML object

Example 9-181. Type-of example

```
> (create-model 'name-generator)
;; creates a new name-generator model
> (add-object (the) 'box 'box-object)
;; adds a box to the name-generator
> (type-of (the))
;; returns : NAME-GENERATOR
> (type-of (the box))
;; returns : BOX-OBJECT
```

See Also: typep

zerop [Function]

The zerop function returns true if a given value is zero and nil if not.

Arguments:

- **number**

  Any expression that returns a number.

Example 9-182. Zerop example

```
> (zerop 0.0)
;; returns : t
> (zerop 1E-100)
;; returns : nil
```

String and Character Functions

Functions

alphanumericp [Function]

This function takes a character and returns t if the character is alphabetic or numeric and nil otherwise.

Arguments:

- **char**

  char must be a character object. it returns t if char is either alphabetic (a-z or A-Z) or numeric (0-9)

Example 9-183. Alphanumericp example

```
> (alphanumericp #\a)
t
> (alphanumericp #\?)
```

```
nil
> (alphanumericp #\*)
nil
> (alphanumericp #\5)
t
```

char [Function]

Returns the character object inside string at index.

Arguments:

- **string**

  Any non-empty string.

- **index**

  Character position in the string. Character indexing begins at 0.

Example 9-184. Char example

```
> (char "example" 4)
;; returns: #\p
```

character [Function]

Returns the character object corresponding to code.

Arguments:

- **code**

  An ASCII integer value, one-character string, or symbol representing an ASCII integer value.

See Also: char-int

Example 9-185. Character example

```
> (character "A")
;; returns: #\A
> (character 65)
;; returns: #\A
> (character '65)
;; returns: #\A
```

characterp [Function]

Returns t if obj is a character, nil otherwise.

Arguments:

- **obj**

  Any object.

Example 9-186. Characterp example

```
> (characterp #\A)
;; returns: t
> (characterp 42)
;; returns: nil
> characterp "example")
;; returns: nil
```

char-int [Function]

Returns the ASCII integer code corresponding to char.

Arguments:

- **char**

  A character object.

See Also: code-char character

Example 9-187. Char-int example

```
> (char-int #\A)
;; returns: 65
```

code-char [Function]

Returns the character object corresponding to code.

Arguments:

- **code**

  An ASCII integer value.

See Also: char-int

Example 9-188. Code-char example

```
> (code-char 65)
;; returns: #\A
```

empty-string? [Function]

The function returns t if its argument is a string that contains blank characters only or no characters at all.

Arguments:

- **string**

  string string to be checked.

princ-to-string [Function]

This function returns a string from the object that the user gives. If the object given has a symbol inside with package information, princ-to-string will not keep it.

Arguments:

- **object**

  Any object that needs to be converted to a string.

Example 9-189. Princ-to-string example

```
> (princ-to-string 1234)
"1234"
> (princ-to-string 'tsi::kjk)
"kjk" ;; doesn't keep the package (tsi) information)
```

See Also: write-to-string

read-from-string [Function]

Takes a string and converts it to the corresponding AML entity. An entity is defined as a list, a number, a keyword or a symbol.

Arguments:

- **string**

  String in question.

Optional Arguments:

- **eof-errorp**

  When nil, does not display an error message and returns eof-value if an error is encountered before the end of the string is reached. For example: To avoid an error, this argument needs to be nil when string is an empty string or only contains semi-colons.

- **eof-value**

  Values to be returned when an error is encountered while parsing the string and eof-errorp is nil.

Example 9-190. Read-from-string example

```
> (read-from-string "3")
;; returns : 3
> (read-from-string "(a b c)")
;; returns : (a b c)
> (read-from-string "")
;; returns : nil
> (read-from-string "" nil nil)
;; returns : nil
> (read-from-string ";;;")
;; returns : nil
```

string-downcase [Function]

This converts all characters of a string to lower case characters and returns the new lower case string.

Arguments:

- **String**

  string to be converted

string-to-delimited-token-list [Function]

The function takes a string as an input and splits it up to return a list of tokens extracted from the string.

Arguments:

- **string**

  Input string to tokenize.

Keyword Arguments:

- **delimiter**

  Specifies the delimiter(s) used to split the string into tokens. It can be a single character or string, or a list of characters or strings.

  Default Value: #\

- **test**

  Specifies the equality function used to compare the token to split the string. The default value of nil implies the test is 'equal. Other functions such as 'string-equal (for a case-insensitive search) could be specified. The value defaults to nil, and can be a function name as a symbol. The function should accept two arguments.

  Default Value: nil

- **blank-token?**

  Specifies how to handle the case where there are multiple consecutive delimiters. In those cases, if this keyword is nil, it ignores the "empty" tokens between the consecutive delimiters. If it is set to t, for every set of consecutive delimiters, the function returns a nil token.

  Default Value: nil

- **string-token?**

  Specifies whether the tokens are returned as strings or if AML should read each token string and return the resultant value, i.e. tokens such as "12" would be returned as the number 12, and "hello" would be returned as a symbol 'hello.

  Default Value: t

Example 9-191. String-to-delimited-token-list example 1

```
1. Simple examples
(string-to-delimited-token-list "this is a test")
> ("this" "is" "a" "test")
(string-to-delimited-token-list "this,is,a,test" :delimiter #\,)
> ("this" "is" "a" "test")
```

Example 9-192. String-to-delimited-token-list example 2

```
2. Multiple delimiters
(string-to-delimited-token-list "this is a,test" :delimiter '(#\space #\,))
> ("this" "is" "a" "test")
```

Example 9-193. String-to-delimited-token-list example 3

```
3. Illustrating keyword :string-token?
(string-to-delimited-token-list "this is a test of 2 numbers 1 2")
> ("this" "is" "a" "test" "of" "2" "numbers" "1" "2")
(string-to-delimited-token-list "this is a test of 2 numbers 1 2" :string-
token?
nil)
> (this is a test of 2 numbers 1 2)
```

Example 9-194. String-to-delimited-token-list example 4

```
4. Illustrating strings as delimiters
(string-to-delimited-token-list "this is a test of 2 numbers 1 2" :delimiter
'("of"))
> ("this is a test "    " 2 numbers 1 2")
```

Example 9-195. String-to-delimited-token-list example 5

```
5. Illustrating keyword blank-token?
(string-to-delimited-token-list "this is a    test")
> ("this" "is" "a" "test")
(string-to-delimited-token-list "this is a    test" :blank-token? t)
> ("this" "is" "a" nil nil nil "test")
```

Example 9-196. String-to-delimited-token-list example 6

```
6. Illustrating keyword :test
(string-to-delimited-token-list "this is ONE test" :delimiter "one")
> ("this is ONE test")
(string-to-delimited-token-list "this is ONE test" :delimiter "one" :test
'string-equal)
> ("this is "      " test")
```

string-upcase [Function]

This converts all characters of a string to upper case characters and returns the new upper case string.

Arguments:

- **String**

  string to be converted

write-to-string [Function]

This function returns a string from the object that the user gives. If the object given has a symbol inside with package information, write-to-string will keep it and include everything in the string.

Arguments:

- **object**

  Any object that needs to be converted to a string.

Example 9-197. Write-to-string example

```
> (write-to-string 1234)
"1234"
```

See Also: princ-to-string

File System

Functions

copy-file [Function]

Copy-file is used to copy files.

Arguments:

- **file1**

  The file to be copied.

- **file2**

  The copy file.

Keyword Arguments:

- **flag**

  If nil and the file exists then the file is overwritten. If t and the file exists, the function fails.

  Default Value: nil

create-directory [Function]

This function creates a new directory.

Arguments:

- **directory-path**

  String representing the directory path and ending by the name of the new directory to create. The path of the directory must exist.

delete-directory [Function]

Function that deletes an empty directory.

Arguments:

- **directory-path**

  String representing a directory path.

directory [Function]

Directory takes a string representing a directory path and returns a list of the contents of that directory. The list returned contains elements in the form of a full path string.The functions directory-string and file-namestring are typically used in parsing the strings returned.

Arguments:

- **directory-path**

  String representing a directory path.

See Also: directory-string, file-namestring

directory? [Function]

Function that returns true if the specified directory-path is a directory.

Arguments:

- **directory-path**

  String representing a directory path.

directory-string [Function]

Function directory-string takes a string representing the path and name of a file and returns a string with the path only.

Arguments:

- **pathname-string**

  String representing the full path and name of a file.

drop-dot-extension [Function]

This function takes a file name string and returns it without the file name extension.

file-copy [Function]

File-copy is used to copy files.

Arguments:

- **path1**

  The file to be copied.

- **path2**

  The copy file.

Example 9-198. File-copy Example

```
> (file-copy (logical-path :home "logical.paths") (logical-path
        :home "logical.copy"))
;; returns : "~/logical.copy"
```

file-delete [Function]

File-delete is used to delete a file.

Arguments:

- **path**

  The full path of the file to be deleted.

Example 9-199. File-delete Example

```
> (file-delete (logical-path :home "logical.copy"))
;; returns : "~/logical.copy"
```

file-length [Function]

File-length returns the length (number of bytes) of a file.

Arguments:

- **file-path**

  The full path of a file.

Example 9-200. File-length Example

```
(file-length (logical-path :home "myevaluation.doc"))
;; returns : length of file in number of bytes: 6513
```

file-move [Function]

File-move is used to move a file from one directory to the other or to simply rename a file. File-move can be used to rename a file at the same time it moves it to another folder or can also just rename a file in its place without moving it to another folder. Executing: (file-move "c:\\test-file.txt" "c:\\renamed-test- file.txt") simply renames test-file.txt to renamed-test-file.txt in the C: root folder.

Arguments:

- **file1**

  The path string of the file to be moved or renamed.

- **file2**

  The new path string for the file.

Example 9-201. File-move Example

```
(file-move "c:\\temp\\output.txt" "c:\\TechnoSoft\\output.txt")
;; Moves the file output.txt from c:\temp\ to c:\TechnoSoft\
(file-move (logical-path :home "logical.paths") (logical-path :home
"logical.copy"))
;; Renames the file
(file-move "c:\\test-file.txt" "c:\\renamed-test-file.txt")
;; Renames the file test-file.txt to renamed-test-file.txt
```

file-namestring [Function]

Function file-namestring takes a string representing the path and name of a file and returns a string with the file name only.

Arguments:

- **pathname-string**

  String representing the full path and name of a file.

formatted-list-from-file [Function]

The formatted-list-from-file function can read data from a file and create a list from the data.

Arguments:

- **file**

  A file name to open and read from data from.

Keyword Arguments:

- **:element-format**

  The format of how the list should be created.

Example 9-202. Formatted-list-from-file Example

```
Assume a file of data points needs to read into a coordinate list.
File: ~/coords.xyz
0 0 0
1 0 1
1 1 1
0 1 0
To read the values into a coordinate list use the following:
> (FORMATTED-list-FROM-FILE (logical-path :home "coords.xyz")
        :element-format '(x y z))
;; returns : ((0 0 0) (1 0 1) (1 1 1) (0 1 0))
nil

If instead only the x and y value were needed:
> (FORMATTED-list-FROM-FILE (logical-path :home "coords.xyz")
        :element-format '(x y))
;; returns : ((0 0) (0 1) (0 1) (1 1) (1 0) (1 0))
```

```
nil

or if the list is an x value with the y and z value as a sublist:
> (FORMATTED-list-FROM-FILE (logical-path :home "coords.xyz")
        :element-format '(x (y z)))
;; returns : ((0 (0 0)) (1 (0 1)) (1 (1 1)) (0 (1 0)))
nil
```

probe-file [Function]

This function is used to check if a file exists. It returns nil if the file specified does not exist.

Arguments:

- **file-pathname**

  String representing the full path and name of the file in question.

read [Function]

The read function reads one entry from a given stream (file) and sets the fill pointer to the end of that entry so that consecutive calls can step through a file one entry at a time. Items in a file that are between parenthesis are considered one entry by the read function and are returned as a list.

Arguments:

- **stream**

  Any open stream. The default value is the standard output (or evaluation buffer).

Optional Arguments:

- **eof-error-p**

  This value determines if an error should occur when the end of the file is reached. The default is true and can (usually) be set to nil to avoid the error. This is not a key, it is optional.

  Default Value: t

- **eof-value**

  The is the value that gets returned when :eof-error-p is nil and the end of file is reached. Be careful when using the nil value because it may be hard to determine if nil signals the end of a file or just a value of nil in the file. This is not a key, it is optional.

Example 9-203. Read Example

```
Assume a file of data values needs to read into a list.
File: ~/coords.xyz
0 0 0
1 0 1
1 1 1
0 1 0
To read the values into a single list use the following:
> (with-open-file (file (logical-path :home "coords.xyz")
        :direction :input
        )
```

```
  (loop for line = (read file nil :eof)
   until (equal line :eof)
   for xyz = (read-from-string (format nil "(~a)" line))
   append xyz
   ))
;; returns : (0 0 0 1 0 1 1 1 1 0 1 0)
```

read-binary-int [Function]

This function reads an integer stored in binary data from a file stream.

Arguments:

- **stream**

  Any open stream.

Keyword Arguments:

- **:big-endian?**

  When t, big-endian is implemented. Otherwise, little-endian is implemented.

  Default Value: t

- **eof-error-p**

  This value determines if an error should occur when the end of the file is reached. The default is t and can (usually) be set to nil to avoid the error.

  Default Value: t

- **eof-value**

  The is the value that gets returned when :eof-error-p is nil and the end of file is reached. Be careful when using the nil value because it may be hard to determine if nil signals the end of a file or just a value of nil in the file.

read-binary-double [Function]

This function reads a double stored in binary data from a file stream.

Arguments:

- **stream**

  Any open stream.

Keyword Arguments:

- **:big-endian?**

  When t, big-endian is implemented. Otherwise, little-endian is implemented.

  Default Value: t

- **eof-error-p**

  This value determines if an error should occur when the end of the file is reached. The default is t and can (usually) be set to nil to avoid the error.

  Default Value: t

- **eof-value**

  The is the value that gets returned when :eof-error-p is nil and the end of file is reached. Be careful when using the nil value because it may be hard to determine if nil signals the end of a file or just a value of nil in the file.

read-binary-float [Function]

This function reads a float stored in binary data from a file stream.

Arguments:

- **stream**

  Any open stream.

Keyword Arguments:

- **:big-endian?**

  When t, big-endian is implemented. Otherwise, little-endian is implemented.

  Default Value: t

- **eof-error-p**

  This value determines if an error should occur when the end of the file is reached. The default is t and can (usually) be set to nil to avoid the error.

  Default Value: t

- **eof-value**

  The is the value that gets returned when :eof-error-p is nil and the end of file is reached. Be careful when using the nil value because it may be hard to determine if nil signals the end of a file or just a value of nil in the file.

read-line [Function]

The read-line function reads an entire line from a stream (file) and sets the fill pointer to the next line so that consecutive calls can step through a file one line at a time.

Arguments:

- **stream**

  Any open stream. The default value is the standard output (or evaluation buffer).

Optional Arguments:

- **eof-error-p**

  This value determines if an error should occur when the end of the file is reached. The default is true and can (usually) be set to nil to avoid the error.

  Default Value: t

- **eof-value**

  The is the value that gets returned when :eof-error-p is nil and the end of file is reached. Be careful when using the nil value because it may be hard to determine if nil signals the end of a file or just a value of nil in the file.

Example 9-204. Read-line Example

```
> (with-open-file (file "/etc/fstab")
(read-line file))
"/dev/sd0a / 4.2 rw 1 1"
NIL
>
Assume a file of data points needs to read into a coordinate list.
File: ~/coords.xyz
0 0 0
1 0 1
1 1 1
0 1 0
To read the values into a coordinate list use the following:
> (with-open-file (file (logical-path :home "coords.xyz")
        :direction :input
        )
   (loop for line = (read-line file nil :eof)
    until (equal line :eof)
    for xyz = (read-from-string (format nil "(~a)" line))
    collect xyz
    )
   )
((0 0 0) (1 0 1) (1 1 1) (0 1 0))
```

with-open-file [Function]

The with-open-file function is used to open files for reading and writing. The specified file is opened as a stream and can then be accessed using the functions read, read-line, read-binary-int, read-binary-double, read-binary-float , or written to using the functions format, write-binary-int, write-binary-double, write-binary-float.

Arguments:

- **stream**

  A symbol that is used to refer to the open stream file.

- **filename**

  A file path name.

Keyword Arguments:

- **:direction**

  This keyword can have 3 values - :input, :output and :io. :input is used to read from the file. :output is used to write to the file. :io - to read and write.

  Default Value: :input

- **:if-exists**

  This keyword can accepts any of the following 4 values - :append , :error , :overwrite or nil. This keyword is effective if :direction is equal to :output, and the file already exists. The effect of the different values of this keyword is as follows: . :append -> appends to the file. . :error -> cause an

error message to pop-up. . :overwrite-> overwrites the file. . nil -> return without writing to the file. Omitting this keyword or giving it any other value will overwrite the file.

Default Value: :overwrite

- **:if-does-not-exist**

  Can be :error or nil. Omitting this keyword or or giving it any other value will cause an error if :direction is :input and the file does not exist.

  Default Value: nil

Example 9-205. With-open-file Example

```
(with-open-file (file (logical-path :home "coords.xyz")
        :direction :output)
    (format file "This is a new file")
    (format file "~% file is ~s" (logical-path :home "coords.xyz")))
nil
This creates a new file in the home directory containing the following:
This is a new file
 file is "P:\\CadC\\CAD-C-v2\\startup-aml4\\coords.xyz"

(with-open-file (file "/etc/fstab")
        (read-line file))
"/dev/sd0a / 4.2 rw 1 1"
nil

Assume a file of data points needs to read into a coordinate list.
File: ~/coords.xyz
0 0 0
1 0 1
1 1 1
0 1 0
To read the values into a coordinate list use the following:
(with-open-file (file (logical-path :home "coords.xyz")
        :direction :input
        )
    (loop for line = (read-line file nil :eof)
     until (equal line :eof)
     for xyz = (read-from-string (format nil "(~a)" line))
     collect xyz))
;; returns : ((0 0 0) (1 0 1) (1 1 1) (0 1 0))
```

write-binary-int [Function]

This function writes an integer to a file stream in binary data. The number should match the type, i.e. it should be an integer.

Arguments:

- **number**

  Number to be written to file.

- **stream**

  Any stream.

Keyword Arguments:

- **:big-endian?**

  When t, big-endian is implemented. Otherwise, little-endian is implemented.

  Default Value: t

write-binary-double [Function]

This function writes a double to a file stream in binary data. The number should match the type, i.e. it should be a double.

Arguments:

- **number**

  Number to be written to file.

- **stream**

  Any stream.

Keyword Arguments:

- **:big-endian?**

  When t, big-endian is implemented. Otherwise, little-endian is implemented.

  Default Value: t

write-binary-float [Function]

This function writes a float to a file stream in binary data. The number should match the type, i.e. it should be a float.

Arguments:

- **number**

  Number to be written to file.

- **stream**

  Any stream.

Keyword Arguments:

- **:big-endian?**

  When t, big-endian is implemented. Otherwise, little-endian is implemented.

  Default Value: t

Inter-Process Communication

This section describes some of the mechanisms available to communicate with external appplications outside of AML and also between two AML sessions.

Client Streams

The following functions are used to communicate with foreign applications through streams.

close-aml-client-stream [Function]

This function closes a stream that communicates to and from AML with external applications.

Arguments:

- **socket**

  This argument is ignored in the current version of AML. It is given here and must still be passed in the function due to backward compatibility purposes. Any valid data type may be given.

- **stream**

  The stream returned from a call to open-aml-client-stream.

get-output-stream-string [Function]

Given a stream (returned by a call to open-aml-client-stream), this function returns a string containing all the characters output to the stream so far. The stream is then reset and each call to this function gets only the characters since the last call.

open-aml-client-stream [Function]

This function opens a stream that communicates to and from AML with external applications. The function returns a handle to the open stream.

Keyword Arguments:

- **port**

  Port number on host in number form.

- **host**

  Machine name or IP address ("123.456.789.012") of the host machine given in string format.

Example 9-206. Open-aml-client-stream Example

The following examples assume that the user is on a machine with the hostname machine_name_1 and they are connecting to machine_name_2 through port 8006. Note the first argument given when closing the stream is ignored, thus any valid AML data can be given.

```
(open-aml-client-stream :port 8006 :host "machine_name_2")
(setf my-stream *)
(close-aml-client-stream 'junk-arg my-stream)
;; returns : t
```

Other Functions

Functions

apply [Function]

The function applies a specific function to a list of arguments. This is useful when the developer has a list of values that need to be passed as arguments to a function.

Arguments:

- **function-name**

  Function that is applied to the list in arg-list.

- **arg-list**

  List of arguments. The function given in function-name will operate on the items in this arg-list.

Example 9-207. Apply example

```
> (max 2 3 100 5 7)
;; returns : 100
> (apply 'max '(2 3 100 5 7))
;; returns again 100
> (average 2 3 100 5 7)
;; returns : 23.4
> (apply 'average '(2 3 100 5 7))
;; returns again : 23.4
```

decode-universal-time [Function]

This function decodes the universal time into Decoded Universal Time Format. It returns the seconds of the current time. If the :values? keyword is not nil, it returns a list with the following order from left to right:(seconds minutes hours day month year).

Arguments:

- **time**

  Any function that returns a valid whole number.

Keyword Arguments:

- **values?**

  When non-nil, the function returns a list in the following format: :(seconds minutes hours day month year).

Example 9-208. Decode-universal-time example

```
> (decode-universal-time 0 :values? t)
;; returns (0 0 19 31 12 1969)

> (decode-universal-time (get-universal-time) :values? t)
```

```
;; returns (26 7 18 21 10 2009)

> (decode-universal-time (get-universal-time))
;; returns 19
```

eval [Function]

The function evaluates a quoted AML expression.

Arguments:

- **expression**

  Quoted AML expression.

Example 9-209. Eval example

```
>(eval '(+ 2 3))
;; returns : 5
> (create-model 'name-generator)
;; creates a new name-generator model
> (eval (read-from-string "(add-object (the) 'box 'box-object)"))
;; returns a box-object instance
```

force-memory-cleanup [Function]

This function forces freeing unneeded memory (garbage collection). It may be time consuming.

get-universal-time [Function]

This function returns the time in seconds in Universal Time Format. The relative date is December 31st, 1969 at 7:00pm.

Example 9-210. Get-universal-time example

```
> (get-universal-time)
;;returns 1256160213
```

getenv [Function]

The function returns the value of the given system's environment variable.

Arguments:

- **var-name**

  String specifying the environment variable.

Example 9-211. Getenv example

```
> (getenv "AML")
```

```
;; returns the current version of AML
```

make-symbol [Function]

The function takes a string and returns its content as a symbol.

Arguments:

- **print-name**

  string specifying the symbol name.

Example 9-212. Make-symbol example

```
(make-symbol "symbol-0001")
;;; returns symbol-0001
```

print [Function]

This function sends the value passed to the standard output or to the destination stream specified. It returns the value after printing. This function is useful for debugging especially because it returns it value argument, so any expression in the source code can be wrapped by a call to print.

Arguments:

- **value**

  Value to print. This can be any valid expression.

Optional Arguments:

- **stream**

  Stream to send the value to. The default sends the value to the standard output. Any other open stream like a file or a socket stream can be used.

  Default Value: nil

run-program [Function]

This function runs an executable program, batch file, or shell script (unix) located on the local file system or a mounted directory. When the :wait keyword is t, this function returns the exit code of the program (as defined by the program's developer). When :wait is nil, it returns the process id of the program.

Arguments:

- **command**

  String specifying the command and (optionally) its corresponding command line arguments.

Keyword Arguments:

- **:arguments**

  A list of strings where every string is another command line argument to command. Strictly speaking, command line arguments can also be supplied in command; however, this keyword allows the user to forgo formatting arguments into a large string.

Default Value: nil

- **:input**

  Input stream for external program. nil means standard input.

  Default Value: nil

- **:output**

  Output stream for external program. nil means standard output.

  Default Value: nil

- **:wait**

  When t, waits for the external program to finish. When nil, run-program returns immediately.

  Default Value: t

- **:show-window**

  When :show, shows command prompt window. When :hide or nil, hides command prompt window.

  Default Value: :show

- **:windows-batch-file?**

  When t, indicates that command is a Windows batch file or a command that requires a command prompt shell. This keyword is ignored on non-Windows platforms.

  Default Value: nil

- **:show-abort-form?**

  When t and AML user interface is loaded, also displays a form allowing the user to kill the process by clicking on a button. When AML user interface not loaded or not t, this keyword is ignored.

  Default Value: nil

Example 9-213. Run-program example

```
(run-program "start example.doc" :windows-batch-file? t :show-window :hide)
;; returns: Exit code of start program.
;; In this example, start would run the default handler for .doc files
(usually
;; Microsoft Word) and would hide the subsequent command prompt window.
Start is
;; a command line windows program so it requires the :windows-batch-file?
argument
;; to be t.

(run-program "xterm" :wait nil)
;; returns: Process ID of xterm process launched.
```

sleep [Function]

This function causes execution within the AML process to cease for n seconds of real time.

Arguments:

- **n-seconds**

  Integer specifying the number of seconds to cease operation

Example 9-214. Sleep example

```
> (loop for i from 0 to 2
   do  (print i) (print "Starting Sleeping")
      (sleep 2)
        (print "Done Sleeping")
        )
;; returns :
0
"Starting Sleeping"
"Done Sleeping"
1
"Starting Sleeping"
"Done Sleeping"
2
"Starting Sleeping"
"Done Sleeping"
nil
```

time [Function]

Runs form, prints runtime statistics for form, and returns the value of form.

Arguments:

- **form**

  Any valid AML form to execute.

Prints the form execution time used down to millisecond precision. Prints the space allocated in number of symbols, conses (lists), additional memory allocated, and other AML structures (such as hash tables).

Example 9-215. Time example

```
> (time (sleep 0.5))

;; Prints:
time used: 0 second 500 mill-second
space allocated:
        symbol:         0
        cons:           0
        data:           0
        other:          1

;; returns: t
```

today [Function]

Returns the date (dd-mmm-yyyy) as a string. If :values? is t, returns a list containing the date (dd-mmm-yyyy) as a string, the time of day (hh:mm:ss), day of the week, and the number of seconds since the midnight of January 1, 1970 GMT.

Example 9-216. Today example

```
(today)
;; returns: "21-Jan-2004"
(today :values? t)
;; returns: ("21-Jan-2004" "16:56.16" "Wednesday" 1074722176)
```

get-time [Function]

Returns the local time and date in a list format. The list has seven entries whic are from left to right: seconds, minutes, hours, day of the month, month, year, day of the year. The day of the year is given as an integer from 0 to 6, 0 being Sunday.

Example 9-217. get-time example

```
(get-time)
;; returns: (16 36 17 22 2 2010 1)
;; The time above represents: 5:36:16 pm on Monday February 22, 2010
```

Chapter 10. Language Constructs

Variables

There are two main types of variables: LOCAL and GLOBAL. Local variables are defined within a local context and when the context is exited the memory used to hold the variables is released. Global variables are defined within the context of the entire system so that any part of any program can refer to the variables. Global variables are never released by the system and the memory used by them is not freed.

Global Variables

defvar [Function]

The defvar function is used to create a globally-defined variable that is evaluated the first time the variable is demanded. This means that the value is not calculated until the first time that it is required and will do nothing for any successive calls with the same variable name. If there is no init-value supplied the variable is created for a value to be set for it at some other time. Defvar is generally used to define a variable that will be changed during execution of a program.

Arguments:

- **variable**

  Any symbol. The common convention is to use a beginning and ending asterisk (*).

Optional Arguments:

- **init-value**

  Any optional expression whose return value will be stored in variable.

Example 10-1. Defvar example

```
> (defvar *global-variable* 4)
;; sets *global-variable* = 4
> *GLOBAL-VARIABLE*
;; returns : 4
> (defvar *global-variable* 5)
:; does nothing because this variable is already defined
> *GLOBAL-VARIABLE*
4 ; the value from the original definition of the variable
> (setf *GLOBAL-VARIABLE* 5)
;; now that changes *global-variable* to 5
> *GLOBAL-VARIABLE*
;; returns : 5
```

defconstant [Function]

The defconstant function, like defvar and defparameter, defines a variable to be used within programs. A defconstant requires an init-value like defparameter but may not be modified by a program. Defconstant is typically used to hold values that will not change, such as pi.

Arguments:

- **variable**

  Any symbol. The common convention is to use a beginning and ending asterisk (*).

Optional Arguments:

- **init-value**

  Any optional expression whose return value will be stored in variable.

Example 10-2. Defconstant example

```
> (defconstant *yard-to-meter* 0.914400)
;; sets *yard-to-meter* = 0.914400
> (* 50 *YARD-TO-METER*)
;; returns : 45.72
> (setf *YARD-TO-METER* 1)
;; returns : Error: Cannot change the value of *YARD-TO-METER* -- it is a
constant.
```

Local Variables

let/let* [Function]

The let form is the most common mechanism for creating local variables. Let(or previously used let*) binds values to the variables in sequence which means that a variable may use a variable previously defined in the same let/let* assignments.

Arguments:

- **assignment-clauses**

  The let clauses are a list of variable formula lists and/or variables.

- **body**

  Any number of expressions to be evaluated within the context of the assigned variables.

Example 10-3. Let/let* Example

```
> (let ((a-variable 1.0) (b-variable 2.0)) (+ a-variable b-variable) )
;; sets a = 1, b = 2 and returns : 3.0 from the addition
> (let ((one 3.0) (twice-one (twice one))) (+ 1.0 twice-one))
;; sets one = 3, twice one = 2*one = 6 and returns 6 + 1 = 7.0
> (defun QUADRATIC (a b c)
        (let ((radical (- (expt b 2) (* 4 a c)))
        (demoninator (twice a))
```

```
        (numberator+ (+ (- b) (sqrt radical)))
        (numberator- (- (- b) (sqrt radical))))
        (list (/ numberator+ demoninator) (/ numberator- demoninator))
        )
  )
;; sets quadratic to the formula
 (QUADRATIC 1 3 2)
;; evaluates and returns : (-1.0 -2.0)
```

Logical Operators

Functions

and [Function]

The and function evaluates any number of expressions in order until one of the expressions returns nil or the last expression is reached. If all expressions are processed the value returned will be the value from the last expression. If all of the expressions are not evaluated, a nil value will be returned because one of the expressions returned a nil value. Remember that any value that is not nil is considered true.

Arguments:

- **expression**

  Any expression to be evaluated.

Example 10-4. And Example

```
> (and (= 1 1.0) (evenp (expt 6 12)) (oddp (* 333 11)) )
;; returns : T
> (and (> 3 2) (= 4 4))
;; returns : T
> (and t nil)
;; returns nil
> (and (= 1 1.0) (evenp (expt 6 12)) (oddp (* 333 11)) "All is true" )
;; returns : "All is true"
> (defun CHECK-VALUE (number)
(let ((low 0.0) (hi 10.0)) (if (and (< low number) (> hi number)) "In Range"
"Invalid" ) ) )
;; defines check-value
> (check-value 1)
;; returns : "In Range"
> (check-value -1)
;; returns : "Invalid"
> (check-value 10)
;; returns : "Invalid"
> (check-value 9.99)
;; returns : "In Range"
```

logand [Function]

The logand function returns the binary bitwise AND of integers.

Optional Arguments:

- **integers**

  Integers to be ANDed.

Example 10-5. Logand Example

```
> (logand 14 15)
;; Binary: 1110 AND 1111 = 1110
;;returns 14
> (logand 14 18)
;; Binary: 00001110 AND 00010010 = 00000010
;; returns 2
>
```

logor [Function]

The logor function returns the binary bitwise OR value on one or more input integers.

Optional Arguments:

- **integers**

  Integers to be ORed.

Example 10-6. logor Example

```
>(logor 1 2)
;; Binary: 0001 OR 0010 = 0011
;; returns 3

>(logor 5 2)
;; Binary: 0101 OR 0010 = 0111
;; returns 7

>(logor 5 2 8)
;; Binary 0101 OR 0010 OR 1000 = 1111
;; returns 15
```

lognot [Function]

The lognot function returns the binary bitwise NOT on one or more input integers.

Arguments:

- **integer**

  Integers to have bitwise NOT applied on it.

Example 10-7. Lognot Example

```
> (lognot 1)
;; returns  -2
> (lognot 10)
;; returns  -11
> (lognot 123)
;; returns  -124
```

logxor [Function]

The logxor function returns the binary bitwise XOR value on one or more input integers.

Optional Arguments:

- **integers**

  Integers to be XORed.

Example 10-8. logxor Example

```
> (logxor 3 5)
;; Binary: 0011 XOR 0101 = 0110
;; returns 6
> (logxor 8 5)
;; Binary: 1000 XOR 0101 = 1101
;; returns 13
> (logxor 11 5)
;; Binary: 1011 XOR 0101 = 1110
;; returns 14
```

or [Function]

The or function evaluates any number of expressions in order until one of the expressions returns true or the last expression is reached. If all expressions are processed the value returned will be the value from the last expression. If all of the expressions are not evaluated a true value will be returned because one of the expressions returned a true value and the following statements are ignored. Remember that any value that is not nil is considered true.

Arguments:

- **expression**

  Any expression to be evaluated.

Example 10-9. Or Example

```
> (or t nil)
;; returns : t
> (or nil nil)
;; returns : nil
> (or (> 5 3) "5 less than 3")
```

```
;; returns : t
> (or (= 1 1.01) (evenp (* 333 11)) (oddp (expt 6 12)) "All are false" )
;; returns : "All are false"
> (defun INVERT (number) (or (= number 0) (/ 1 number)) )
;; defines INVERT
> (invert 4)
;; returns : 1/4
> (invert 4.0)
;; returns : 0.25
> (invert (expt 7 5))
;; returns : 1/16807
> (invert (expt 7 5.0))
;; returns : 5.949901826619861e-5
```

not [Function]

The not function is generally used to invert a true or false value. This may be used in the same instances as null. As a matter of style, null is generally used to test for the empty list and not is used for negative logic.

Arguments:

- **value**

  Any expression or value whose result will be inverted.

See Also: null

Example 10-10. Not Example

```
> (not (= 1 1.0))
;; returns : nil
> (not (evenp (expt 3 3)))
;; returns : t
> (not (eq 1.0 1))
;; returns : t
```

xor [Function]

This function provides exclusive OR functionality. It takes an indefinite number of arguments. If only one argument is not nil, it returns t. If all arguments are nil, it returns nil. If more than one argument is not nil, it returns nil.

Arguments:

- **expression**

  Any expression to be evaluated.

Example 10-11. Xor Example

```
(xor t nil)
;; returns : t
```

```
(xor (> 5 3) (> 5 6))
;; returns : t
(xor (= 5 3) (> 5 6))
;; returns : nil
(xor nil nil)
;; returns : nil
```

Conditionals

Functions

case [Function]

The case statement compares a test-key to a number of keys and evaluates the expressions that are included with the first matching key.

Arguments:

- **test-key**

  Any expression that will evaluate to a result that gets compared against all of the keys.

- **key**

  A symbol or list of symbols that will be compared to test-key. If the value is a list the key-test is compared to each of the keys in the list. A key of t may be included as the last key to operate as a default that will match in all instances.

- **expressions**

  Any number of expressions to be evaluated if the key matches the test-key.

Example 10-12. Case Example

```
> (case 3
(1 "Value: One")
(2 "Value: Two")
(3 "Value: Three")
(t "Value: Unknown"))
;; returns : "Value: Three"
> (case 5
(1 "Value: One")
(2 "Value: Two")
(3 "Value: Three")
(t "Value: Unknown"))
;; returns : "Value: Unknown"

;;; The following function will return; 1 for a name of box-object or
sheet-object, 2 for a name of cylinder-object or sphereobject,
and of those fail the value will be 3.
> (defun object-id ()
(case (object-name (the))
((box-object sheet-object) 1)
```

```
((cylinder-object sphere-object) 2)
(t 3)) )
;; defines object-id
> (create-model 'box-object)
;; initiates box-object
> (object-id)
;; returns : 1
> (create-model 'sphere-object)
;; initiates another object - sphere
> (object-id)
;; returns : 2
> (create-model 'pipe-object)
;; initiates pipe-object
> (object-id)
;; returns 3
```

cond [Function]

The cond statement is the most general conditional available. The cond is composed of a number of clauses that each must include a test followed by optional expressions to evaluate if the test is not nil. When a test is found to be non nil, the accompanying expressions are evaluated and control exits the cond without looking at any other keys (only the first true statement gets evaluated).

Arguments:

- **test1**

  Any expression that will evaluate to a result that gets compared against all of the keys.

- **expressions**

  Any number of expressions to be evaluated if the key matches the test-key.

Example 10-13. Cond Example

```
> (let ((a 1) (b 3) (c 5)) (cond ((evenp (+ a b c)) (print "The sum is
even.") )
((oddp (+ a b)) (print "The first two sum if odd") (t (+ a b c)) ) ))
;; returns t = 9
> (defun init-model () (cond ((or (eq (object-name (the)) 'cylinder-object)
(eq (object-name (the)) 'sphere-object))
;; If the model is named cylinder-object or sphere-object
  (undraw (the)) (change-value (the diameter) (twice (the diameter)))
   (draw (the)) 1 )
   ((and (eq (object-name (the)) 'box-object)
   (> (the height) 1.0))
;; If a box is taller that 1
   (change-value (the height) 1.0)
   (change-value (the color) 'green)
   (regen)
   2
   )
  (t ;; If all else fails do this stuff
   (delete-all-models t)
```

```
   (create-model 'box-object)
   (draw (the))
   3
   )
   )
)
;; defines unit-model
> (delete-all-models t)
> (init-model)
;; returns : 3
> (create-model 'cylinder-object)
> (INIT-MODEL)
;; returns : 1
> (create-model 'box-object)
> (change-value (the height) 2.0)
;; changes to 2.0
> (INIT-MODEL)
;; returns : 2
> (the height)
;; returns : 1.0
```

if [Function]

The if statement is the simplest conditional. If the test is not nil, the then clause is evaluated and if the test is nil, the else clause is executed. The else clause is optional and will return nil if not included. Because it only allows a single expression for the then and else clauses, it is sometimes necessary to use a progn statement, which treats many expressions as a single function, for one of these clauses.

Arguments:

- **test**

  Any expression that will be used to determine whether to evaluate the then or else clause.

- **then**

  Any single expression that will be evaluated when test is not nil.

Optional Arguments:

- **else**

  Any optional single expression that will be evaluated when test is nil.

Example 10-14. If Example

```
> (if (= 2 3) "True." "False.")
'' returns "False."
> (if (< 2 3) "True." "False.")
;; returns "True."
> (if (evenp (expt 3 2)) "Even")
;; returns : nil
> (defun CHECK-RANGE (number)
        (let ((lo 1)
        (hi 10) )
```

```
        (if (<= lo number hi)
        ;; If in range print message and return a true value
        (progn (format t "The value: ~,2f is between ~,2f and ~,2f"
        number lo hi)
        t)
        ;; If not in range print message and return a false
        (progn (format t "WARNING value: ~,2f is NOT between ~,2f and
        ~,2f"
        number lo hi)
        nil)))
        )
;; defines check-range
> (check-range 7)
;; returns : The value: 7.00 is between 1.00 and 10.00
;; t
> (check-range 17)
;; returns : WARNING value: 17.00 is NOT between 1.00 and 10.00
;; NIL
```

unless [Function]

The unless statement is a conditional that evaluates a test, and if the result is nil all of the expressions of the body are evaluated, otherwise nil is returned. Unless is not absolutely necessary because an if statement with a then clause of nil and an else clause containing a progn can perform the same function. However it is more convenient to use unless because it will evaluate any number of expressions in the body. In addition, unless can make code easier to read because there is no then clause that needs to be included as nil.

Arguments:

- **test**

  Any expression that will be used to determine whether to evaluate the then or else clause.

- **body**

  Any number of expressions to be evaluated when test is true.

Example 10-15. Unless examples

```
> (unless (evenp (expt 3 4))
   (format t "The expt value is not even ") (* 3 4))
;; returns The expt value is not even 12


;; Returns true if value-1 is greater than value-2, nil otherwise.
> (defun greater-than? (value-1 value-2)
    (unless (> value-2 value-1)
      t
      )
    )
;; Defines function "greater-than?"
> (greater-than? 1 5)
```

```
;; Returns nil

> (greater-than? 17 9)
;; Returns t
```

when [Function]

The when statement is a conditional that evaluates a test and if the test returns non-nil, all of the expressions of the body are evaluated.When returns the return value of the last expression of the body. When returns nil if the test fails. When is not absolutely necessary because if with a progn "then" clause and no "else" clause will perform the same function. However it is more convenient to use when because it will evaluate any number of expressions in the body. In addition, when can make code easier to read because there is no else clause that may or may not be included.

Arguments:

- **test**

  Any expression that will be used to determine whether to execute the body.

- **body**

  Any number of expressions to be evaluated when test is true.

Example 10-16. When Exmaple

```
> (when (evenp (expt 3 4))
    (format t "The expt value is even")
    (* 3 4) )
;; returns : nil
> (when (oddp (expt 3 4))
    (format t "The expt value is even")
    (* 3 4) )
;; returns : The expt value is even 12
> (when (the (:error nil)); If there is a model created clear
   (clear); the display and draw everything.
   (draw (the))
   (zoom 'all))
;; returns : (4.5466332f0 3.6373067f0)
```

Loop

The loop facility may be used for virtually all needed iterations. Due to the versatility of the loop statement, there are many control parameters that are used to perform required tasks. These control parameters are discussed next according to functionality.

Iteration Control

The following iteration controls must precede other loop arguments except for the with, initially, and finally arguments. There can be any number of iteration controls in a single loop statement. The iterations occur simultaneously and the loop will finish when any one of the iterations is completed.

for

The for argument is a general increment (decrement) control. There are a number of paramenters that can be used to control how iteration should procede. The parameters are used with the loop's for control to accomplish many different types of iteration.

```
                from, to, downto, upto
                downfrom, upfrom
                by
                =
                then
                in, on
                across
                inhash, inhash-keys, inhash-values
                (for inhash the variables have two elements - one for hash
key and one for hash value)

(loop for i from 0 to 3
        do (print i))
;; returns :
0
1
2
3
nil

(loop for var inhash hash-table ...)
This iterates through the records of a hash table created by the function
make-hash-table. At every iteration "var" is assigned a hash table record, a
hash table record contains 2 element: The first element is the hash key and
the second element is the hash value.

Example
(let ((hash-table (make-hash-table :size 200))
     )
  ;; Populating hash table
  (setf (gethash 1 hash-table) '(a "A"))
  (setf (gethash 2 hash-table) '(b "B"))
  (setf (gethash 3 hash-table) '(c "C"))
  ;; Looping and printing all records of the hash table
  (loop for record inhash hash-table
    do
    (print record)
    ))
```

```
(loop for var inhash-keys hash-table ...)
This iterates through the records of a hash table created by the function
make-hash-table. At every iteration "var" is assigned a different hash table
key.


(loop for var inhash-keys hash-table ...)
This iterates through the records of a hash table created by the function
make-hash-table. At every iteration "var" is assigned a different hash table
value.
```

repeat

The repeat argument is a simple control that iterates a specified number of times. Examples:

```
(loop for i from 10
        downto 0 by 3 do (print i))
;; returns :
10
7
4
1
nil
(loop for i upfrom 1
        repeat 3 do (print i))
;; returns :
1
2
3
nil
(loop for i downfrom 1
        repeat 3 do (print i))
;; returns :
1
0
-1
nil
(loop for i downfrom 10 by 3
        repeat 3 do (print i))
;; returns :
10
7
4
nil
(loop for i downfrom 5 to 3
        do (print i))
;; returns :
5
4
3
```

```
nil
(loop for i from 1 to 100
        for val = (expt i 3)
        until (> val 800) collect i)
;; returns : (1 2 3 4 5 6 7 8 9)
(loop for i from 0 downto -10
        by 4 do (print i))
;; returns :
0
-4
-8
nil
(loop for i in '(a b c) do (print i))
;; returns :
A
B
C
nil
(loop for i on '(a b c d) do (print i))
;; returns :
(A B C D)
(B C D)
(C D)
(D)
nil
```

Termination Control

while

The while argument allows a loop to iterate until a specified test evaluates to a false value.

until

The until argument allows a loop to iterate until a specified test evaluates to a true value.

Value Accumulation

The resulting accumulated value will be returned by the loop statement. When more than one accumulation is needed in a single loop statement the into parameter is needed to create a local variable to hold the results of the accumulations. If these accumulated values need to be returned, use the finally and return statements.

append

The append argument takes a list each time it is encountered in the loop and appends each of those lists into a single list during the iteration process.

collect

The collect argument takes a value each time it is encountered in the loop and adds each value to a list during the iteration of the loop.

maximize

The maximize argument evaluates its parameter each time through the loop and returns the greatest value when the loop is finished.

minimize

The mimimize argument evaluates its parameter each time through the loop and returns the lowest value when the loop is finished.

sum

The sum argument accumulates a running total of its parameter. When more than one accumulation is needed in a single loop statement the INTO parameter is needed to create a local variable to hold the results of the accumulations.

```
(loop for i in '(a b c d f)
        for j in '(4 3 2 1 0)
        append (list i j))
;; returns : (A 4 B 3 C 2 D 1 F 0)
(loop for i in '(a b c d f)
        for j in '(4 3 2 1 0)
        collect (list i j))
;; returns : ((A 4) (B 3) (C 2) (D 1) (F 0))
(loop for i in '(5 3 7 1 9)
        maximize i into max
        minimize i into min
        finally (return (list min max)))
;; returns : (1 9)
(loop for i from 0 to 1000 by 4.125
        count i)
;; returns : 243
(loop for i from 1 to 10
        sum i)
;; returns : 55
(loop for i in '(a b c d f)
        for j in '(4 3 2 1 0)
        for k in '(one two three)
        maximize j into jmax
        minimize j into jmin
        collect (list k i j) into l1
        append (list k i j) into l2
        finally (return (list jmin jmax l1 l2)))
;; returns : (2 4 ((ONE A 4) (TWO B 3) (THREE C 2)) (ONE A 4 TWO B 3 THREE C
2))
```

Local Variable Assignment

with

The with argument is used to create local variables with in the scope of the loop. These must appear before any iteration control arguments.

```
(loop with x = 20
        with y = 50
        with z = 10
        for i from x to y by z
        collect i)
;; returns : (20 30 40 50)
```

Conditional Execution

if [else]

The if argument is used to perform conditional execution within a loop. The else parameter is optional.

```
(loop for i from 0 to 5
        if (evenp i)
        do (print "Even")
        else
        do (print "Odd"))
;; returns :
"Even"
"Odd"
"Even"
"Odd"
"Even"
"Odd"
nil
(loop for i from 0 to 10 by 3
        if (evenp i)
        collect i)
;; returns : (0 6)
```

when

The when argument is used to perform some operations when a condition is true.

```
(loop for i from 0 to 10 by 3
        when (evenp i)
        do (print i)
        collect i)
;; returns :
0
```

```
6
(0 3 6 9)
(loop for i from 0 to 10 by 3
        when (evenp i)
        do (print i) and
        collect i)
;; returns :
0
6
(0 6)
```

Note: Using the and in the second example allows more that one loop argument to be considered as part of the when body. In the first, the collect executed for each iteration and in the second collect executed only when the when test was true.

unless

The unless argument is used to perform some operations when a condition is NOT true. It can be easier to read and understand to use a when with a not function around the test instead of unless.

```
(loop for i from 1 to 1000
        unless (> i 5)
        do (print i))
;; returns :
1
2
3
4
5
nil
```

Body Execution

do

The do parameter is used to perform general execution. Do will take any number of expressions and evaluate them in order like a progn statement so it is not necessary to supply a do in front of multiple statements to be executed.

```
(loop for i from 1 to 3
        do
        (print i)
        (print (* i i)))
;; returns :
1
1
2
4
```

```
3
9
nil
(loop for i from 1 to 3
        do (print i)
        do (print (* i 1)))
;; returns :
1
1
2
2
3
3
nil
```

Initial and Final Execution

finally

The finally argument is used to perform some operation once after finishing the loop iteration. The location of finally within the loop does not matter because it will always be evaluated after finishing iteration.

```
(loop for i from 5 downto 1
        do (print i)
        with j (print "Count Down:")
        finally (return (print "Blast Off!")))
;; returns :
"Count Down:"
5
4
3
2
1
"Blast Off!"
nil
(loop for i from 1 to 10
        if (evenp i)
        collect i into evens
        else
        collect i into odds
        finally (return (list odds evens))
        )
;; returns : ((1 3 5 7 9) (2 4 6 8 10))
```

Returning Values

return [Function]

The return function terminates the loop and returns a value as the return value of the loop.

Arguments:

- **value**

  Any value that user wants to return.

Example 10-17. Return Example

```
> (loop for i from 1 to 100
   when (> i 9) do (return i)
         do ... ;;the else statements
   )
;; returns 10
```

Expression Grouping

The following constructs are used to wrap more than 1 expression into one expression group and to control the value returned by the group. Grouping expressions is particularly useful with if conditionals for example because if requires a single expression for its then and else statements.

Functions

progn [Language Construct]

This construct groups a list of expressions by executing them sequentially and returning whatever the last expression of the group returns.

Example 10-18. Progn Example

```
(defun return-greater (a b)
      (if (> a b)
        (progn
         (format t "Value a is greater then value b ")
         a
         )
      (progn
        (format t "Value b is greater then value ")
        b))
     )
;; defines return-greater
(return-greater 5 6)
;; returns :
Value b is greater then value 6
```

without-gc [Function]

This function disables garbage collection during the execution of the body.

Arguments:

- **body**

  One or more AML statements

Example 10-19. Without-gc Example

```
(without-gc (delete-object (the boxes box1)) (add-object (root-object) 'box2
'box-object))
```

Formatted Output

Format Directives

format [Function]

The format function is very useful for creating formatted text. Format may be used to print to a stream (which may be a file), to construct a string, or to print to standard output.

Arguments:

- **destination**

  This is where the output is to be sent. If it is nil, a string is created and returned by the function. If the value is t, the output is sent to standard output, which is the evaluation buffer. If the value is any other stream the output is sent to that stream which can be used to write to a file or a socket.

- **control-string**

  A string of characters used to create the output. Characters are sent as output except for the tilde (~) character which indicates a directive. A directive consists of one or more elements to create formatted output. For each directive in the control there must be an argument to be processed by the directive.

Optional Arguments:

- **arguments**

  Values that are to be processed by the directives in the control-string. There must be at least as many arguments as there are directives in the control-string. If there are more arguments than directives the extras are ignored, but if there are less the format function will cause an error.

~A - ASCII. An argument is printed without escape characters. The string representation of the argument is used as output.

~S - S-expression. This is just like ~A but the argument is printed with escape characters. The output is therefore suitable for input to read. ~S accepts all the arguments and modifiers that ~A does.

~D - Decimal. An argument is printed in decimal. This does not place a decimal point after the number. An output column width may be specified by using the ~mincolD syntax where mincol is the column width. To pad the output with a character other than a space use the ~mincol,padcharD syntax. To change the comma separator use the third parameter to ~D. Therefore the most general form for ~D is ~mincol,padchar,commacharD. Examples :

```
(format nil "~10d" 45)
;; returns : "          45"
(format nil "~10,'0d" 45)
;; returns : "0000000045"
```

~B - Binary. The binary directive is like ~D but it generates binary format instead of decimal. The full form is therefore ~mincol,padchar,commacharB. Examples :

```
(format nil "~10b" 45)
;; returns : "    101101"
(format nil "~10,'0b" 45)
;; returns : "0000101101"
```

~O - Octal. The octal directive is like ~D but it creates octal format instead of decimal. The full form is therefore ~mincol,padchar,commacharO.

~X - Hexadecimal. The hexadecimal directive is like ~D but it creates hexadecimal format instead of decimal. The full form is therefore ~mincol,padchar,commacharX.

~F - Floating-Point. An argument is printed as a floating point number. The complete general format for the floating point directive is ~w,d,k,overflowchar,padcharF. The w parameter is the field width; d is the number of digits after the decimal point; k is a scale factor (default 0); overflowchar defines what to print if the number exceeds the w value; padchar defines what to use to fill a column if the value does not fill column.

```
(format nil "~f" 123.456)
;; returns : "123.456"
(format nil "~10f" 123.456)
;; returns : " 123.456"
(format nil "~10,2f" 123.456)
;; returns : " 123.46"
```

~E - Exponential floating-point. An argument is printed in exponential notation. The complete general format for exponential notation is ~w,d,e,k,overflowchar,padchar,exponentcharE. The w parameter is the field width; d is the number of digits after the decimal point; k is a scale factor (default 0); overflowchar defines what to print if the number exceeds the w value; padchar defines what to use to fill a column if the value does not fill column; and exponentchar defines the character to signify the exponential. In addition the @ sign may be added to cause the sign to be used for positive and negative values (default is only negative).

```
(format nil "~8e" 12.345612)
;; returns : "1.235e+1"
(format nil "~8,2e" 12.345612)
;; returns : " 1.23e+1"
(format nil "~8,2e" 1234.5612)
;; returns : " 1.23e+3"
```

~% - New Line. This directive terminates the current line and requires no arguments. To create multiple new lines include an integer between the ~ and %.

```
(format t "~2%TechnoSoft Inc.~%")
;; returns :

TechnoSoft Inc.

NIL
```

~& - ~& Line Begin. If the line position is not currently at the end of the line a new line is created.

```
(format t "~&TechnoSoft Inc.")
;; returns : TechnoSoft Inc.
nil
```

~~ - Tilde. This will generate a tilde.

~c - Tilde. This will insert a character in the string.

```
(format nil "~c" '#\a)
;; returns : "a"

(format nil "~c~c~c" '#\a '#\b '#\c)
;; returns : "abc"

;; in XEmacs, enter <ctrl-q><tab> to insert a tab-character.
(format nil "Column 1~cColumn 2" '#\    )
;; returns : "Column 1  Column 2"
```

~T - Tabulate. This spaces over to a given absolute column. Using ~n@T will add n spaces after the previous character (relative tab).

```
(format t "This is~10ttabbed")
;; returns : This is   tabbed
nil
(format t "This is~10@ttabbed")
;; returns : This is          tabbed
nil
```

~(str~) - Lower case conversion.

~:(str~) - Capitalize case conversion.

~@(str~) - Capitalize first word case conversion.

~:@(str~) - Upper case conversion.

```
(format nil "~(~a~)" "John Doe")
;; returns : "john doe"
(format nil "~:(~a~)" "john doe")
;; returns : "John Doe"
(format nil "~@(~a~)" "John Doe")
; returns : "John doe"
(format nil "~:@(~a~)" "John Doe")
;; returns : "JOHN DOE"
```

~{str~} - Iteration. The iteration directive requires a list of elements that will be processed one at a time by the parameters of str.

~:{str~} - Sublist Iteration. This works like ~{str~} except that it works for a list of sublists.

```
(format nil "~{~a~}" '(a b c d e f g))
;; returns : "abcdefg"
(format nil "~{~a ~}" '(a b c d e f g))
;; returns : "a b c d e f g "
(format nil "(~{ ~a ~})" '(a b c d e f g))
;; returns : "( a b c d e f g )"
(format t "The users: ~{ ~:(~a~)~}." '(bob jim steve))
;; returns : The users: Bob Jim Steve.
(format nil "Coordinates: ~{~a ~}." '((1 2 3) (4 5 6)))
;; returns : "Coordinates: (1 2 3) (4 5 6) ."
```

Functions

with-output-to-string [Function]

This function is used to write data to a string using the format function.

Arguments:

- **string-name**

  This is a local variable accessable within the scope of the body which refers to the name of the string. This will be used as the stream argument in any format functions within the body.

- **body**

  Body of the function used to make format statements.

Example 10-20. With-output-to-string example

```
(with-output-to-string (local-string)
        (loop
        for item in '("Hello" "How are you?" "Great, thanks!")
        do (format local-string "~a~%" item)
        )
        )
;; returns :
"Hello
```

```
How are you?
Great, thanks!
"
```

Errors And Error Handling

Functions

with-error-handler [Function]

This function executes an AML statement given by the user and runs a user-defined error-handling function. When an AML/Lisp error occurs: If the user has provided an error-function, the function is executed and the value returned by the error-function is returned. It also means that the keywords "error-message", "show-system-error?" and "error-return-value" are ignored If error-function is nil (default behavior), the error-message (when provided) is printed on the message pane, then the error-return-value is returned.

Keyword Arguments:

- **:error-function**

  Name of a user-defined function called by the error handler when an error occurs. The function should have two arguments. The error handler calls the error-function with the following 2 arguments respectively: 1- the error-function-argument provided by the user. 2- the AML system error message. (the user can decide to use it or ignore it).

  Default Value: nil

- **:error-function-argument**

  First argument of the user-defined error-function.

- **:error-message**

  Error string printed on the message pane is case no error-function is provided.

- **:show-system-error?**

  When non-nil, prints the AML/Lisp system error on the message pane right before printing the error-message.

  Default Value: t

- **:error-return-value**

  Value returned in case of an error when no error-function is provided by the user.

  Default Value: nil

Example 10-21. With-error-handler Example 1

```
(with-error-handler (:error-message "Could not successfully create a name-
generator and add a box to it"
                :error-return-value 'return-value-on-error
                )
                (create-model 'name-generator)
```

```
                (add-object (the) 'box2 box-object)
                )
AML error: Attempt to take the value of the unbound variable `BOXOBJECT'.
Could not successfully create a name-generator and add a box to it
RETURN-VALUE-ON-ERROR
```

Example 10-22. With-error-handler Example 2

This example shows how an error recovery method can be written on a class in order to recover from errors that will occur in instances of that class during program execution. (The method error-recovery-method below can be redefined for subclasses of object to perform class-specific error recovery)

```
(define-method error-recovery-method object ()
  (message (format nil "The object ~a is recovering from its last error"
(object-name self)))
  'error-recovery-return-value
  )

(defun error-recovery-function (object system-error-message)
  (message system-error-message)
  (error-recovery-method object)
  )

(with-error-handler (:error-function 'error-recovery-function
                        :error-function-argument (root-object)
                        )
                        (initialize-object (root-object))
                        )
AML error: attempt to call `INITIALIZE-OBJECT' which is an undefined
function.
The object NAME-GENERATOR is recovering from its last error
ERROR-RECOVERY-RETURN-VALUE
```

Chapter 11. Logical Paths

Logical paths files

AML has the capability of defining logical-path-reference variables to locate resources on the file system. They are defined in logical paths files. On Unix platform, a user logical paths file (logical.paths) can be created under the user's home directory. On WINDOWS platforms, a logical paths file (logical.pth) exists under the AML working (or startup) directory and the user can append entries to it. The logical paths file contains lines with logical-path-reference and corresponding path definitions.

```
logical-path-reference  path-definition
        :home                ~/
        :tmp                 /tmp/
        :engine           :home engine/
```

Compiler directives can be used in a logical paths file to distinguish between a WINDOWS and a non-WINDOWS entry in the following way:

```
#-WINDOWS
:home ~/
#+WINDOWS
:home c:\users\user1\
```

A #+WINDOWS directive specifies that the following line is only visible on WINDOWS machines, while a #-WINDOWS directive specifies the the following line is only visible on non- WINDOWS machines. Directives should not be followed by logical path entries on the same line, and only one directive per line is allowed.

Logical Paths Functions

disable-logical-path-file [Function]

This function disables a logical path file. This will disable all paths defined in the file and will make them unaccessible through the logical-path function.

Arguments:

- **file**

  File path and name string of a logical path file.

See Also: enable-logical-path-file

enable-logical-path-file [Function]

This function enables a logical path file for use during a running AML session. All entries in that file will be accessible through the logical-path function. The file being enabled must follow the format of the standard AML logical.pth file. If a logical path file is enabled and it contains entries that also exist in the already enabled logical path file, the entries in the new file will take precedence.

Arguments:

- **file**

  File path and name string of a logical path file.

See Also: disable-logical-path-file

logical-path [Function]

logical-path is a function that converts a logical path name into a path value. The logical paths are defined in the AML logical paths file(s) or defined at runtime using the function set-logical-path. A logical path name is in the form of an AML keyword (Example :my-path) and the path value is a string that typically refers to a file or directory on the file system. Logical paths that are set using the set-logical-path function take precedence over the paths defined in a logical path file. If a logical path name is not found, the function returns nil. The logical-path function can also be used to concatenate strings to construct a platform dependent path. The function can also be used to query for system environment variables. In the logical-path.pth file, it will automatically put slashes between names with spaces. It will do this in a platform independent way, e.g.,

```
:final-directory :parent-directory aml test-folder output ""
```

will produce a platform independent path.

Arguments:

- **logical-path-name**

  A logical path name (Example :my-path) or en environment variable (Example: :$variable) or a string representing the root directory of a file path.

Optional Arguments:

- **file-name**

  A string that is concatenated to the logical-path-name reference. More than one string can be specified.

Remarks: The logical-path function can be given a system environment variable, and it will return the value of that variable. Furthermore, a logical path definitio in a logical path file can also reference a system environment variable. Examples: 1. (logical-path :$TECHNSFT_LICENSE_FILE) This call returns the value of the TECHNSFT_LICENSE_FILE license file environment variable. A $ character must follow the colon to indicate that the argument refers to an environment variable and not a logical path definition. 2. A logical path file can contain the following logical path definition: :license :$TECHNSFT_LICENSE_FILE This will cause the call to (logical-path :license) to return the value of the system environment variable named TECHNSFT_LICENSE_FILE

Example 11-2. Logical-path function examples

```
(logical-path :home)
;; returns the address of the AML folder
(logical-path :tmp)
;; returns the address of the temp folder
(logical-path :tmp "MESH/MESH.crd")
;; returns the address to the MESH/MESH.crd file
(logical-path :$TECHNSFT_LICENSE_FILE)
;; This call returns the value of the TECHNSFT_LICENSE_FILE license file
;; environment variable. A $ character must follow the colon to indicate that
the
;; argument refers to an environment variable and not a logical path
definition.
(logical-path "users" "amluser1" "file1.dat")
;; returns : "users/amluser1/file1.dat" on a UNIX system
(logical-path "users/" "amluser1" "file1.dat")
;; returns again : "users/amluser1/file1.dat" on a UNIX system
(logical-path "\\\\server" "users" "amluser1" "file1.dat")
;; returns : "\\\\server\\users\\amluser1\\file1.dat" on a WINDOWS system
(logical-path "c:" "users" "amluser1" "file1.dat")
;; returns : "c:\\users\\amluser1\\file1.dat" on a WINDOWS system
(logical-path "c:\\" "users" "amluser1" "file1.dat")
;; returns : "c:\\users\\amluser1\\file1.dat" on a WINDOWS system
```

Example 11-3. Logical path file content example

```
:my-application         c:\TechnoSoft\my-application\ ;; This is a logical
path definition
:my-application-data    :my-application data\         ;; This is a path
definition relative to another logical path name
:my-application-temp    :my-application temp\         ;; This is a path
definition relative to another logical path name
:system-variable1       :$system-variable1            ;; This refers to a
system environment variable
```

See Also: set-logical-path, unset-logical-path

set-logical-path [Function]

This function sets a runtime logical path that is valid while AML is running or until it is unset. It can be used to set up a project path based on user input or override an existing path.

Arguments:

- **path-name**

  The name of the temporary logical path reference.

- **path**

  A string representing the path pointed to by the logical path reference.

Example 11-4. Set-logical-path example

```
(logical-path :demo)
;; returns : nil
(set-logical-path :demo "/tmp/engine/")
;; returns : "/tmp/engine/"
(logical-path :demo)
;; returns : "/tmp/engine/"
```

See Also: logical-path, unset-logical-path

unset-logical-path [Function]

This function removes a logical path definition that was set using set-logical-path. Remark: If the logical path name given also exists in a logical path file, the definition that exist in the logical path file stays valid; therefore, the next call to the function logical-path will read the definition from the logical path file.

Arguments:

- **path-name**

  The name of a logical path reference that needs to be undefined.

Example 11-5. Unset-logical-path example

```
> (unset-logical-path :demo)
```

See Also: logical-path, set-logical-path

logical-path? [Function]

This function checks whether a particular logical path reference exists in the logical.pth file. If the entry exists, it returns the string corresponding to the reference. Otherwise, the function returns nil.

Arguments:

- **logical-path-name**

  The name of a logical path reference defined in logical.pth file.

- **file-name**

  A string that is concatenated to logical-path-name reference.

Example 11-6. Logical-path? example

```
(logical-path? :demo)
;; returns : nil
(set-logical-path :demo "/tmp/engine/")
;; returns : "/tmp/engine/"
(logical-path? :demo)
;; returns : "/tmp/engine/"
```

Chapter 12. System Management

System Management

The management of source code is accomplished through the definition of systems. A system is a set of source code files that are grouped together. Defining a system allows the code in a system to be treated as a module that may be loaded, compiled, and archived as a single entity. Compiling systems creates version subdirectories and archives the source files and the binary files so updating older versions is possible by using the archived code. A system can contain binaries for multiple platforms within a version to allow different platforms to be operating with the same system version.

A system may require other systems to automatically load before loading or compiling itself. Organizing code into systems that may be loaded is a methodology for the reuse of code. Cut-and-paste is not code reuse but rather typing reuse.

A logical path is a reference to files and directories in the system. The logical.pth file stores the logical path references, making modification easy. These references are queried by the use of the logical-path function. The logical.pth file is typically located in the working directory of the AML session.

System Management Functions

aml-full-version-label [Function]

This function will return the enhanced AML version name (string) in the following format: v.n1.n2 (n3-n4-n5-n6-n7-n8).

Description of the different components of the version string:

- v: AML major release number.
- n1: AML minor release number.
- n2: Sub-release control number or update pack control number.
- n3: AML executable control number.
- n4: AML graphics and geometry server control number.
- n5: AML graphical user interface library (VIL) control number.
- n6: AML model parser control number.
- n7: AML geometry exchange importer control number.
- n8: AML geometry exchange exporter control number

Example 12-1. Aml-full-version-label Example

```
(aml-full-version-label)
;; returns : "AML Version 4.6.2 (103-78-2557-6-7-9)"
```

define-system [Function]

The main mechanism for defining systems. The definition should appear in a system definition file named system.def. The system.def must be in the directory returned by the call (logical-path :system-name). The system-name must be an entry in the logical path file for the system to be found when compiling or loading systems.

Arguments:

- **system-name**

  The name of the system being defined. It is recommended that the system have a keyword name to make it package independent.

Keyword Arguments:

- **:require-systems**

  A list of system names that must be loaded or compiled before the system being defined may be compiled or loaded. The example below shows the format accepted by the :require-systems keyword: 1. Version 4 of the system named :base-system must always be loaded anytime :my-aml-application is compiled or loaded. 2. The system named :extension-system-1 is also required. Since no constraining list is provided, this system may also be compiled during the compilation of :my-aml-application. Refer to the :all? keyword defined with the function compile-system later on in the section. 3. The system named :extension-system-2 is also required for loading only. Since no version is specified, the latest version will always be loaded.

- **:require-libs**

  This is a list of shared library names. These libraries are loaded when the system is loaded. This list must be a list of file names without their extension. It is expected that the file extensions are: .dll on Windows, .sl on HPUX, and .so on all other UNIX and Linux platforms. The developer must create a directory named lib as a sibling directory to the system's sources directory. The lib directory must contain platform dependent sub-directories for each supported operating system. Lib sub-directories are named as follows: windows, solaris, linux, sgi, aix, and hp. The library files must be placed in their corresponding platform dependent sub-directory. Note that not all sub-directories must exist, only the needed ones must be created based on what platform the AML system will be running on. When the system is compiled, the lib directory will be copied to the version archive directory. The function "load-system" loads the libraries from the archived lib directory.

- **:files**

  A list of files that form the system. The files must be located in a subdirectory named sources only.

Example 12-2. Define-system example

```
(define-system :MY-AML-APPLICATION
        :require-systems '((:base-system load-only 4)
                           :extension-system-1
                           (:extension-system-2 load-only)
                           )
        :require-libs '("myapplib")
        :files '(
                    "file1.aml"
                    "file2.aml"
```

```
                    "file3.aml"
                    )
        )

The directory structure for the system my-aml-application must be as shown
below:

my-aml-application/
        system.def          File containing the system definition.
        sources/
                file1.aml       Source code file.
                file2.aml           Source code file
                file3.aml       Source code file.
        lib/
                windows/
                        myapplib.dll
                solaris/
                        myapplib.so

The logical path file must contain the following entry, so the system to be
located:
:my-aml-application     <path to my-aml-application folder>/

Remarks:
The system directory does not need to be named like the system; however, this
practice is encouraged.
The path to the system folder in the logical path file must include a
trailing slash (or back slash on Windows)
```

compile [Function]

Compiles an AML source code file into a binary loadable file.

Arguments:

- **input-file**

  File path-name string of the AML source code file to compile.

- **:output-file**

  File path string of the compiled binary file to create. AML typically uses ".btc" as the extension for compiled file names.

compile-system [Function]

This function compiles a system into binary files and loads the binary files. The source files of a system are defined by define-system (see above) in the system.def file. Every system must be compile in order for it to be functional. During the first compilation of a system, a version subdirectory named "<system-name>-1" (version 1) is created under the system's main directory. The version subdirectory contains: 1- An archive "sources" subdirectory that contains a copy of the main source files from the time of compile. 2- A platform dependent subdirectory containing the binary files created by the compilation of the source files. This subdirectory will be used by the function load-system to load the binary files. "Platform dependent subdirectory names are: bins for WINDOWS, hp-bins for HPUX, mip-bins for

SGI, solaris-bins for solaris, ibm-bins for IBM AIX, and linux-bins for LINUX." 3- A system.def file which is a copy of the original system.def at the time of compile. In subsequent compilations of the system, the user has the choice of creating a new version or updating the latest version. This is controlled by the keyword :new?. If the user decides to create a new version, the system is compiled into a new version subdirectory similarly to the first compilation of the system described above. The new version subdirectory will be named by incrementing the suffix of the latest version subdirectory. If the user decides to update the latest compiled version, the source files of the archive are compared to the main source files. The source files that have changed are overwritten in the archive and compiled to overwrite their latest binary version. The user can force the compilation and archiving of all source files by setting the :force? keyword to t.

Arguments:

- **system-name**

  The name of a system to compile. The system name must be an entry that is found in the logical.pth file.

Keyword Arguments:

- **:force?**

  This keyword is ignored during the first compilation of the system. Default only compiles files that have changed since the last compilation. When non-nil, all files are compiled.

  Default Value: nil

- **:forget?**

  This keyword is ignored during the first compilation of the system. It controls the redefinition of a system. The default will read the system.def file from the latest version subdirectory. When non-nil, the original system.def is read. Changes made to a system.def file of a loaded system will not be recognized unless :forget? is t.

  Default Value: nil

- **:new?**

  When non-nil, creates a new version subdirectory; Otherwise, the system is compiled into the latest version subdirectory.

  Default Value: nil

- **:all?**

  When non nil, all systems required by system-name as defined in define-system are compiled prior to compiling system-name unless they are defined as load-only systems. When nil, required systems are only loaded.

  Default Value: nil

Example 12-3. Compile-system example

```
; The following compiles the system files the first time on a SOLARIS
; platform and creates MY-AML-APPLICATION-1 subdirectory structure
; shown below.
> (compile-system :my-aml-application)
```

```
/home/
   apps/
      my-aml-application/
        system.def          File containing the system definition.
        sources/
             file1.aml      Source code file.
             file2.aml      Source code file.
             file3.aml      Source code file.
        MY-AML-APPLICATION-1/
             system.def
             sources/
               file1.aml    Archived source code file.
               file2.aml    Archived source code file.
               file3.aml    Archived source code file.
             solaris-bins/
               file1.sbin   Binary file.
               file2.sbin   Binary file.
               file3.sbin   Binary file.

; The following reads the system.def file and compiles only changed
; or uncompiled files into the existing version.
> (compile-system :system :forget? t :force? nil :new? nil)
```

create-module [Function]

This function compacts a collection of AML systems and their resources into one module. An AML module consists of a directory containing one btc file (that includes all btc files from all required AML systems) and other resource directories based on the "require-libs" and "require-resources" keywords of the included AML systems. Modules are loaded into AML using the function load-module. A directory named "module-name-version" will be created by create-module. When a module is created using create-module, a module.dat file is created inside the module directory listing information regarding the module name, module version, AML version and AML systems included in the module. **Remark:** The AML version specified in the module.dat file is not necessarily the AML version that the included systems were compiled under, it is simply the AML version that was used at the time the module was created.

Arguments:

- **module-name**

  Name (string) of the module (can include the full path to directory where the module will be saved, otherwise module will be saved in current directory). The directory "module-name-version" will be created.

- **version**

  Version (string) of the module.

- **system-name**

  Name of the root system that the module will be created from. Typically, the root system is a system that requires other AML systems. All required system will be included in the created module. This parameter can also be a list of system names to indicate that the module consists of more than one root system

- **system-versions**

  Version of the root system. When this parameter is nil, the module will be created from the latest version. This parameter should be a list of versions when the system-name parameter is a list of system names.

Keyword Arguments:

- **temp-path**

  Directory for temporary files.

- **documentation**

  This is a string the will be embeded in the module btc file. It can be queried using the function binfile-user-data

- **pre-load-files**

  Optional list of compiled btc files. These files will be included in the module and will be loaded first when the module is loaded.

- **additional-files**

  Optional list of compiled btc files. These files will be included in the module and will be loaded last when the module is loaded.

- **license-info**

  String for checking out license when the module is loaded, format need to be "Feature Version Count".

This function creates a new file module-name-version/module-name.btc (.btc is for Windows, .lbtc for Linux) and copies all needed libs to the module-name-version/directory. This directory needs to be copied to (logical-path :additional-modules) in order for the function load-module to be able to locate it.

See Also: define-system, compile-system, load-system, load-module, binfile-user-data

Example 12-4. create-module example

```
(create-module "c:\\modules\\my-module" "01-00" :my-application nil)
 ;; This will create the directory "c:\modules\my-module-01-00"
(create-module "c:\\modules\\my-other-module" "01-00" '(:my-application :my-
other-app) '(2 4) :documentation "A wing design application")
```

compile-system-file [Function]

This function will allow compilation of a single file in a system without compiling a complete system. The source file will be read and the compiled binary file will be placed into the binaries subdirectory of newest version of the system. The file is also copied into the archive sources of the version subdirectory.

Arguments:

- **system-name**

  The name of the system that contains the file to compile. The system name must be an entry that is found in the logical.pth file.

- **file-name**

The name of the file that appears in the files list of the define-system of the system to compile.

get-command-line-arguments [Function]

This function returns a list of strings which are the command-line arguments when AML starts.

load [Function]

Loads a source code file or a compiled binary file. Using AML functionality that was loaded from a compiled binary file is much more efficient than if its source code was loaded.

Arguments:

- **filename**

  File path-name string to load.

See Also: compile-file

load-module [Function]

This function loads a module. If no path is provided, the module is assumed to be located in the directory returned by the call to "(logical-path :additional-modules)".

Arguments:

- **module-name**

  The name of the module as a string (can include the full path to directory where the module will be saved, otherwise module will be saved in current directory).

Keyword Arguments:

- **:version**

  String. This keyword specified the desired version of the module. If it is nil, AML loads the latest version (A string comparison is made on the module directory names to figure out the latest version).

  Default Value: nil

- **:force?**

  When set to t, will force reloading the module even if it is already loaded.

  Default Value: nil

- **:path**

  String that defines the location of the module directory. When nil, the module is assumed to be located in the directory returned by the call to "(logical-path :additional-modules)".

  Default Value: nil

- **:init-function**

  This is a symbol that specifies the name of the function to be automatically called right after the module is loaded. When :init-function is t, it calls function "module-name-init-function" if that function is defined.

  Default Value: nil

loaded-modules [Function]

This function returns a list of two items : The first item is a list of modules saved in the current AML image file, and the second item is a list of additional modules that were loaded after the current AML image file was started.

load-system [Function]

This function loads a compiled system. When a system is loaded, the platform dependent binary files that were created during the last compilation are loaded if no version number is supplied. When a version number is supplied the binaries of the specified version subdirectory are loaded. This allows versions to be in production and newer versions to be under development. If the source code is changed the changes will not be loaded until a compile system is performed. A system also tracks the version of the binary files that are loaded so that successive loading of the same system will not occur if not needed. If a system has already been compiled on a different platform and an exact version of that compilation needs to be loaded on the current platform, use the port-system function described below to port the compilation of the desired version.

Arguments:

- **system-name**

  The name of a system to load. The system name must be an entry that is found in the logical.pth file.

Keyword Arguments:

- **:forget?**

  This keyword controls the redefinition of a system. The default will read the system.def of the last version subdirectory. If forget? is supplied as t the original system.def file will be read. Note: Changes made to a system.def file of a loaded system will not be recognized unless :forget is t.

  Default Value: nil

- **:version**

  This keyword controls the version (version subdirectory suffix) of the system to be loaded. If not specified, the version with the highest number will be loaded.

Example 12-5. Load-system example

```
(load-system :my-aml-application)
```

platform-name [Function]

Platform-name returns the current OS(Operation System). The possible options are : "windows", "hp", "solaris", "aix", "sgi" or "linux"

port-system [Function]

This function is useful when a system has already been compiled on a different platform, and a corresponding compilation is needed on the current platform. Port-system compiles the source code stored in the version subdirectory sources of the specified version. It creates/updates the binary directory

corresponding to the current platform under the desired version subdirectory. By default, the latest available version is ported.

Arguments:

- **system-name**

  The name of a system to port. The system name must be an entry that is found in the logical.pth file.

Keyword Arguments:

- **:force?**

  Default only compiles files that have changed since the last compile. A t value will compile all files in the correspoding sources archive.

  Default Value: nil

- **:forget?**

  This keyword controls the redefinition of a system. The default will not read the original system.def but will only read the one under the corresponding archive directory.

  Default Value: nil

- **:version**

  Version (archive directory suffix) of the system to port. Default will port that latest archived version.

  Default Value: nil

- **:all?**

  When non nil, all systems required by system-name as defined in define-system are ported prior to porting system-name unless they are defined as load-only systems. When nil, required systems are only loaded.

  Default Value: nil

system-bitmap [Function]

Given a system name and a bitmap-name, this function returns the full path of a bitmap file. The bitmap files are expected to be located in a directory named "bitmaps" located in the system directory. (i.e. the "bitmaps" directory is a sibling to the "sources" directory)

Arguments:

- **system-name**

  Name of an AML system

- **bitmap-name**

  Name (string) of a bitmap file (must not include the dot and the extension).

Keyword Arguments:

- **bitmap-file-extension**

  Extension (string) of the bitmap-name file.

  Default Value: "bmp" for WINDOWS and "xpm" for MOTIF

1. No need to have compiler directives between WINDOWS and UNIX for the file extension: The default value for the bitmap-file-extension keyword will be used if the keyword is omitted. 2. Modules: This function will automatically find the bitmap file when the system is packed into a module. A "bitmaps" directory is automatically created in the module directory. Note: When used with modules, the function system-bitmap does not require the existence of a ":system-name" entry in the logical path.

Example 12-6. System-bitmap example

```
;; Assuming the file bitmap1.bmp (or bitmap1.xpm) exists in the directory
returned by (logical-path :sketcher "bitmaps"), then call the function
system-bitmap as follows:
(system-bitmap :sketcher "bitmap1")
```

System Patching

A software patch is a file that fixes or enhances a compile AML system. System management in AML incorporates patches for automatic loading. Patching a system allows incremental minor enhancements to an AML system without the need to recompile a system.

Creating a patch file:

To create a patch for an existing system, the source code of the patch must be placed in the "patches\sources" directory of a system's version directory as follows:

```
     c:\
        apps\
            my-aml-application\
                system.def       File containing the system definition.
                sources\
                   file1.aml     Source code file.
                   file2.aml     Source code file.
                   file3.aml     Source code file.
                my-aml-application-1\        Archive directory of version 1
(This was created automatically by compile-system)
                        system.def
                        sources\
                                file1.aml        Archived source code file.
                                file2.aml        Archived source code file.
                                file3.aml        Archived source code file.
                        patches\
                                sources\
                                        0001-patch1.aml
                        bins\
                                file1.btc        binary file.
                                file2.btc        binary file.
                                file3.btc        binary file.
```

Remarks:

1. A system must be compiled at least once before it is patched (see compile-system).
2. The name of a patch file must have a four digit prefix.
3. The patch must be placed in a "patches\sources\" directory as shown above.
4. To compile a patch:
    i. Load the version of that system using "load-system".
    ii. Call the function port-patches on the loaded system.
5. This will compile the file into a platform-dependent binary directory. The binary directory is a sibling to the "patches\sources" directory. The names of the binary directory follows the same convention as binary directories created by the function compile-system.
6. Patches are loaded in numerical order when a system is loaded.
7. In order to make future versions of a system reflect the fixes and enhancements of the patches to the current version, the source code of a patch file must be a copy of a modification or addition made to the original source code files of the system. Otherwise, fixes and enhancements made by the patches will not be reflected next time a compile-system is done.

See Also: compile-system, define-system, load-system, port-system.

Functions

patch-system [Function]

Once patch files have been created the files will be loaded automatically when a system is loaded. If the system is already loaded and a new patch is created it can be loaded without loading the complete system by using patch-system. Patch-system will load patches of the system version that is currently loaded into memory.

Arguments:

- **system-name**

  The name of a system to patch. The system name must be an entry that is found in the logical.pth file.

Example 12-7. Patch-system example

```
> (patch-system :system)
;; Loading file "system\SYSTEM-1\patches\bins\patch-0001.btc"
```

Package Definition

Packages allow the user to define code in multiple name-spaces. Typically, most development will be done in the :aml package. Therefore, the user needs to state (in-package :aml) at the beginning of all AML source code files because you will be using functionality from the :aml package. When using functionality between packages, you must specify the package name as a prefix. This can be seen when calling functions of the Virtual Geometry Layer package from within normal AML code for example.

Functions

define-package [Function]

Arguments:

- **package-name**

  The name of the package being defined.

Keyword Arguments:

- **:use**

  A list of package names whose exported values should be accessible by the package being defined.

- **:nicknames**

  A list of alternative names for the package.

Example 12-8. Define-package example

```
(define-package "My-Extension-To-AML"
    :nicknames '("My-AML" M-AML"
                 "my-adaptive-modeling-language")
        )
```

AML Images

An image is the main underlying file from which AML starts. An image holds all the AML classes, methods and functions definitions. The user can create a new AML "image" file that includes user defined classes, methods and functions. This image file can replace the default ".img" file the current AML version is using. Saving an AML image file should be done before calling the function aml or its equivalents (i.e. pressing F6 on UNIX platforms). After saving an AML image, the user must exit AML. Typically, to save a new image, an AML developper starts with the default AML image file, loads the desired systems or modules, saves a new image file, then exits AML.

Note: To run AML with an image file new-aml.img: Run "aml.exe new-aml.img" (aml.exe is located under the AML directory).

Functions

save-aml-image [Function]

Saves an aml image file.

Arguments:

- **image-file**

  File name string of the image file.

Keyword Arguments:

- **:serial-number**

You can add new license requirements to an AML image as follows: (save-aml-image ... :serial-number "<ADD/name FEATURE=version/> ..." . name is any symbol. . FEATURE is the feature (in the license file) that must be checked out when AML starts. . version is a float number. The user can call (root::get-aml-image-serial-number) to see the serial number in the current image.

- **:runtime?**

  This keyword can accept 4 values : nil (Default) : Save a development AML image. t : Save a runtime AML image that starts without a console window and without an AML prompt. 1 : Save a runtime AML image that starts with a console window. 2 : Save a runtime AML image that restarts with a console window and with an AML prompt.

- **:restart-function**

  This a symbol which defines a function with no arguments, it will be appended to the end of the startup functions list.

Chapter 13. Tables

Tables

Classes and Methods

This section describes an efficient and easy data management capability in AML, especially when dealing with dynamic , relatively small amounts of data, allowing also the storage and retrieval to/from ASCII files.

record-table-object [class]

The record-table-object is a table that organizes its data in records. Each record is constituted of a number of fields. The record-table-object is extremely fast in random record access and update.It is not optimized for sequential searches throughout the table. A record-table-object uses a primary key to identify its record. Each record will have a unique primary key value, and the primary-key-field property (see below) is used to specify what field of the record-table-object's records will be used to represent the primary key. A primary key value can be any AML value (number,list,symbol,object instance,...). Note: A record list is a list of the field values of a record.

Inheritance: object

Properties:

- **size**

  An integer specifing the maximum number of records expected. If the actual number of record exceeds the size specified, the record-table-object instance will stay completely functional but will loose some table manipulation efficiency.

  Default Formula: 100

- **file**

  File path string specifying the file that the methods loadtable and save-table will use during data lookup/storage. The file does not have to exist initially , it will be created by the recordtable- object instance whenever the first call to save-table is triggered.

  Default Formula: nil

- **primary-key-field**

  Index to the record field used as the primary key for the record-table-object. A primary-key-field value of 0 means that all the records in the record-table-object will have their first field as their primary key value.

  Default Formula: 0

add-record [Method]

Defined on Classes:

record-table-object

Adds a record to the instance of record-table-object passed.

Arguments:

- **record-table-object**

  Instance of the record-table-object being added to.

- **record**

  List of values representing the added record fields values.

clear-table [Method]

Defined on Classes:

record-table-object

Clears a record-table-object instance of all entries.

retrieve-record [Method]

Defined on Classes:

record-table-object

Retrieves a record from a record-table-object and returns a list of the retrieved record fields values.Uses the primary key value to identify the record to be retrieved.

Arguments:

- **record-table-object**

  Instance of the record-table-object.

- **primary-key-value**

  Value that should match the primary key field value of the record to be retrieved.

delete-record [Method]

Defined on Classes:

record-table-object

Deletes a record from a record-table-object. Uses the primary key value to identify the record to be deleted.

Arguments:

- **record-table-object**

  Instance of the record-table-object.

- **primary-key-value**

  Value that should match the primary key field value of the record to be deleted.

update-record [Method]

Defined on Classes:

record-table-object

Updates an existing record in a record-table-object. Uses the primary key value to identify the record to be updated.If no record matching the primary key value exists, update-record will behave like add-record.

Arguments:

- **record-table-object**

  Instance of the record-table-object.

- **primary-key-value**

  Value that should match the primary key field value of the record to be deleted.

- **record**

  List of values representing the new record fields values (Including the primary key field).

select-table [Method]

Defined on Classes:

record-table-object

Queries a record-table-object and returns a list of records matching the query. (i.e. a list of lists).

Arguments:

- **record-table-object**

  Instance of the record-table-object being queried.

Keyword Arguments:

- **:fields**

  A list of indices specifying what fields of the matching records are returned within a record list. For example if fields = '(0 3), only the first and fourth fields of matching records are returned; therefore select-table returns a list of couples where each couple represents the first and fourth field value of the correponding record.When field is nil (default), all fields are returned.

  Default Value: nil

- **:record-test**

  Name of a user defined function meant to be called on each record list of the record-table-object. Only records that yield to a non Nil return value when passed to record-test are returned by select-table. A record-test function should be defined with two arguments , first argument being the record list in question and second argument specified by test-param (see below).

  Default Value: t

- **:test-param**

  Value passed as the second argument to the record-test function.

  Default Value: nil

Example 13-1. Select-table example

```
Let's assume that we have an employee record-table-object with its records
having only three fields for simplicity: social security number, name and
age. Let us refer to the table instance by table.

> (defun Older-than (record-list age)
(> (nth 2 record-list) 25))
;; The following query will return the social security number and name of all
employees older than 25 currently stored in table.
>(select-table table :field '(0 1) :record-test 'older-than :test-param 25)
;; returns :
((282-34-5543 "Eddie Pope")
(254-45-1111 "Alexi Lalas")
...etc)
```

load-table [Method]

Defined on Classes:

record-table-object

Resets a record-table-object by loading the data in the file specified by the property file in order to populate the table. The table must have been be saved previously using the method savetable (see below)

Arguments:

- **record-table-object**

  Instance of the record-table-object to be reseting.

Optional Arguments:

- **file-path-string**

  Optional argument. When provided with a valid file path string, the file is used instead of the file pointed to by the property file.

save-table [Method]

Defined on Classes:

record-table-object

Saves the current state of a record-table-object into the file specified by the property file.

Arguments:

- **record-table-object**

  Instance of the record-table-object to be saved.

Optional Arguments:

- **file-path-string**

  Optional argument. When provided with a valid file path string, the file is used instead of the file pointed to by the property file.

flat-file-record-table-class [class]

Inheritance: record-table-object

Properties:

- **file:**

  Path-name string of flat formatted flat file.

  Default Formula: nil

- **field-delimiter**

  Character delimiting the fields of a record. Values allowed are: :space, :tab, and :comma.

  Default Formula: :tab

- **record-format**

  List of the desired field type/format. The 4 Types/formats allowed are: :integer, '(:real i) where i is the number of decimal points, :string and :symbol When an integer field is expected and a real number is read, the real number is truncated. When record-format is nil, all fields are read as a string.

  Default Formula: nil

The table is reset whenever either of the properties file or record-format is smashed. The flat file wil be reread in that case.

insert-record [Method]

Defined on Classes:

flat-file-record-table-class

This methods inserts a record to the flat-file-record-table-class at the specified position.The insertion is equivalent to an add-record; the position specified is only reflected when the table is saved back to the flat file or in a call to select-table.

Arguments:

- **record-list**

  A list of the record field values.

- **position**

  Integer specifying the insertion position index. The record is always inserted before the existing record with the given index. Indices start at 0.

Units and Unit Conversions

The AML model-manager manages a units table used for unit conversion. The units table requires the existence of the file "unit-conversion.dat". This file should be located on the directory specified by the AML_PATH environment variable on UNIX platforms or by "(logical-path :aml)" on WINDOWS platforms.

Definitions

Arguments:

- **Unit-expression**

  A unit expression is defined as: 1 - A symbol representing a unit in the unit conversion table. (Example: inch, cm,...) Or 2 - A list representing a composite unit based on available units in the unit conversion table. The format of the list shows the mathematical relation among the individual units forming the composite unit. Example : An acceleration unit can be represented by '(m (s -2). A force unit can be represented by '(kg m (s -2)). In other words a unit-expression can be: 1 - A unit symbol (unit should be available in the table). 2 - A list of unit symbols. 3 - A list of couples , each couple representing a unit symbol and an exponent.

Functions

add-unit [Function]

Adds a unit to the unit conversion table and specifies its relationship with (an) existing unit(s). The new unit is specified as a factor of a valid unit expression.

Arguments:

- **new-unit-symbol**

  Symbol to be used for new unit.

- **factor**

  Real number specifying: 1 new-unit-symbol = factor * unit-expression

- **unit-expression**

  Valid unit expression.

Example 13-2. Add-unit example

```
> (add-unit 'Pa 1.0 '(N (m -2))
```

convert-units [Function]

Convert a value or a list of values from one unit to the other. Both units should be valid unit-expressions.

Arguments:

- **value**

  Real number value or list of values to convert.

- **from-unit**

  Unit expression to convert from.

- **to-unit**

  Unit expression to convert to.

Example 13-3. Convert-units example

```
> (convert-units 1.0 'm 'in)
;; returns : 39.37007874015748
> (convert-units 1 'N 'lb)
;; returns : 0.2248089236553391
> (convert-units 1 'lb '(kg m (s -2)))
;; returns : 4.448222
```

convert-units-temperature [Function]

Converts a temperature value or a list of values from one unit to the other. Both units should be valid unit-expressions such as degk, degc, degf and degr.

Arguments:

- **value**

  Real number value or list of values to convert.

- **from-unit**

  Unit expression to convert from.

- **to-unit**

  Unit expression to convert to.

Example 13-4. Convert-units-temperature example

```
> (convert-units-temperature  45 'degf 'degc)
;; returns : 7.2222222222222285
> (convert-units-temperature 245 'degk 'degf)
;; returns :-18.670000000000016
> (convert-units-temperature (list 32 25 14 5 0 -12 -25 -40) 'degc 'degf)
;; returns :(89.59999999999997 76.99999999999994 57.19999999999999
40.99999999999994 31.999999999999943 10.399999999999977 -13.000000000000057 -
40.00000000000006)
```

dimensionally-similar-units-list [Function]

This function takes a unit-expression and returns a list of the unit symbols in the table that are dimensionally similar to the unit-expression provided. When the unit expression provided equals t, this function returns all available unit symbols in the table.

Chapter 14. Translators

Geometry Exchange

This section describes functionality that allows importing and exporting geometry into and out of AML in DXF, IGES, and STEP format. The geometry exchange functionality expects the path :geometry-exchange-exec to exist in the logical path of the current AML session and to point to the directory where the AML translator executable files are located. By default, this path is automatically set while installing AML.

Export functionality

aml-to-dxf [Function]

This function is used to translate the geometry of AML graphic object instances into a DXF file.

Keyword Arguments:

- **:objects**

  A list of graphic object instances to be translated into the DXF format.

- **:out-file**

  The full path and name string of the DXF file to be created.

- **:approx-curve?**

  If the user is exporting a surface geometry that maps to a curve bounded surface (CBS) then the user can control the type of curves that make up the CBS loops. When this keyword is non-nil, the curves will be mapped to nurb curves. Otherwise, the system checks for the curve types (line, arc, polygon, ..) and generates the required definitions accordingly. It is highly recommended not to set this keyword to nil, because most CAD systems can deal with nurbs.

  Default Value: t

- **approx-surf?**

  If the user is exporting a surface geometry that maps to a curve bounded surface (CBS) then the user can control the type of surface that defines the underlying CBS surface. When this keyword is non-nil, the underlying surfaces will be mapped to nurb surfaces. Otherwise, the system checks for the surface type (sheet, cylinder, disc, ..) and generates the required definition accordingly. It is highly recommended not to set this keyword to nil because most CAD systems can deal with nubs, but may have trouble dealing with cylindrical or disk surfaces...

  Default Value: t

- **:model-units**

  Model-units takes a unit. To get a list of possible values call the function geometry-exchange-possible-model-units-list. Default value is unitless.

  Default Value: 'data-transfer

- **:generate-solids?**

  At this time this argument is ignored in this function

  Default Value: nil

- **:user-model-tolerance**

  The geometric tolerance value.

  Default Value: "(vgl::get-tol-value)"

- **:canonical?**

  if set to t then the dxf is generated to reflect the canonical geometries.

  Default Value: nil

- **:join-loop-edges?**

  When t, the loop edges are approximated by a single nurb curve. This keyword was added to interface to AutoShip and should only be used when needed.

  Default Value: nil

- **:tag-function-name**

  This could be a function or a method.

- **:tag-function-argument-list**

  A list, typically the first element in the list is an instance. As the code goes through and generates the data for the different subgeoms, each subgeom is appended to the list and the function/method specified in tag-function-name is called.

See Also: geometry-exchange-possible-model-units-list

aml-to-iges [Function]

This function is used to translate the geometry of AML graphic object instances into an IGES file.

Keyword Arguments:

- **:objects**

  A list of graphic object instances to be translated into the IGES format.

- **:out-file**

  The full path and name string of the IGES file to be created.

- **:approx-curve?**

  If the user is exporting a surface geometry that maps to a curve bounded surface (CBS) then the user can control the type of curves that make up the CBS loops. When this keyword is non-nil, the curves will be mapped to nurb curves. Otherwise, the system checks for the curve types (line, arc, polygon, ..) and generates the required definitions accordingly. It is highly recommended not to set this keyword to nil, because most CAD systems can deal with nurbs.

  Default Value: t

- **:approx-surf?**

If the user is exporting a surface geometry that maps to a curve bounded surface (CBS) then the user can control the type of surface that defines the underlying CBS surface. When this keyword is non-nil, the underlying surfaces will be mapped to nurb surfaces. Otherwise, the system checks for the surface type (sheet, cylinder, disc, ..) and generates the required definition accordingly. It is highly recommended not to set this keyword to nil because most CAD systems can deal with nubs, but may have trouble dealing with cylindrical or disk surfaces...

Default Value: t

- **:iges-143?**

  When t then CBS are mapped to 143 instead of 144.

  Default Value: nil (i.e. 144)

- **:model-units**

  Takes a unit. To get a list of possible values call the function geometry-exchange-possible-model-units-list. Default value is unitless.

  Default Value: 'data-transfer

- **:generate-solids?**

  At this time this argument is ignored in this function.

  Default Value: nil

- **:user-model-tolerance**

  The geometric tolerance value.

  Default Value: "(vgl::get-tol-value)"

- **:canonical?**

  If set to t then the Iges is generated to reflect the canonical geometries.

  Default Value: nil

- **:join-loop-edges?**

  When t, the loop edges are approximated by a single nurb curve. This keyword was added to interface to AutoShip and should only be used when needed.

  Default Value: nil

- **:tag-function-name**

  This could be a function or a method.

- **:tag-function-argument-list**

  A list, typically the first element in the list is an instance. As the code goes through and generates the data for the different subgeoms, each subgeom is appended to the list and the function/method specified in tag-function-name is called.

See Also: geometry-exchange-possible-model-units-list

aml-to-step [Function]

This function is used to translate the geometry of AML graphic object instances into a STEP file.

Keyword Arguments:

- **:objects**

  A list of graphic object instances to be translated into the STEP format.

- **:out-file**

  The full path and name string of the STEP file to be created.

- **:approx-curve?**

  If the user is exporting a surface geometry that maps to a curve bounded surface (CBS) then the user can control the type of curves that make up the CBS loops. When this keyword is non-nil, the curves will be mapped to nurb curves. Otherwise, the system checks for the curve types (line, arc, polygon, ..) and generates the required definitions accordingly. It is highly recommended not to set this keyword to nil, because most CAD systems can deal with nurbs.

  Default Value: t

- **approx-surf?**

  If the user is exporting a surface geometry that maps to a curve bounded surface (CBS) then the user can control the type of surface that defines the underlying CBS surface. When this keyword is non-nil, the underlying surfaces will be mapped to nurb surfaces. Otherwise, the system checks for the surface type (sheet, cylinder, disc, ..) and generates the required definition accordingly. It is highly recommended not to set this keyword to nil because most CAD systems can deal with nubs, but may have trouble dealing with cylindrical or disk surfaces...

  Default Value: t

- **:model-units**

  Model-units, takes a unit. To get a list of possible values call the function geometry-exchange-possible-model-units-list. Default value is unitless.

  Default Value: 'data-transfer

- **:generate-solids?**

  When t, generates solid geometry.

  Default Value: nil

- **:user-model-tolerance**

  The geometric tolerance value.

  Default Value: "(vgl::get-tol-value)"

- **:canonical?**

  If set to t then the step is generated to reflect the canonical geometries.

  Default Value: nil

- **:join-loop-edges?**

  When t, the loop edges are approximated by a single nurb curve. This keyword was added to interface to AutoShip and should only be used when needed.

  Default Value: nil

- **:tag-function-name**

This could be a function or a method.

- **:tag-function-argument-list**

  A list, typically the first element in the list is an instance. As the code goes through and generates the data for the different subgeoms, each subgeom is appended to the list and the function/method specified in tag-function-name is called.

- **:map-faces-to-shells?**

  Some systems have issues with shells, in that case this keyword should be set to nil. Otherwise it should be left alone.

  Default Value: t

See Also: geometry-exchange-possible-model-units-list

dxf-out-multiview-port [Function]

This function will allow the user to generate dimension entities for DXF. Note: Before calling this function, the user needs to undraw all the objects in main-view-port-id, then pass the list to the function. Result: An DXF with facetted geometry and DXF dimension entities.

Arguments:

- **file-name**

  DXF file name

- **main-view-port-id**

  The main view port ID (the viewport id where the dimensions are drawn)

Keyword Arguments:

- **:dimension-objects**

  A list of the dimension objects drawn in main-view-port-id

- **:user-model-tolerance**

  Model tolerance. Default value uses the system's default tolerance.

  Default Value: nil

- **:model-units**

  Model measurement unit. The function geometry-exchange-possible-model-units-list will allow user to list all acceptable units.

  Default Value: 'data-transfer

See Also: geometry-exchange-possible-model-units-list

geometry-exchange-possible-model-units-list [Function]

This function lists all acceptable model units that can be used with the functions aml-to-iges, aml-to-step and aml-to-dxf.

iges-out-multiview-port [Function]

This function will allow the user to generate dimension entities for IGES. Note: Before calling this function, the user needs to undraw all the objects in main-view-port-id, then pass the list to the function. Result: An IGES with facetted geometry and IGES dimension entities.

Arguments:

- **file-name**

  IGES file name

- **main-view-port-id**

  The main view port ID (the viewport id where the dimensions are drawn)

Keyword Arguments:

- **:dimension-objects**

  A list of the dimension objects drawn in main-view-port-id

- **:user-model-tolerance**

  Model tolerance. Default value uses the system's default tolerance.

  Default Value: nil

- **:model-units**

  Model measurement unit. The function geometry-exchange-possible-model-units-list will allow user to list all acceptable units.

  Default Value: 'data-transfer

See Also: geometry-exchange-possible-model-units-list

set-edge-approximation-tolerance [Function]

This function is used to set the tolerance value for generating nurb data for an edge. If never called the default tolerance is 10 to the power -5.

Arguments:

- **tolerance**

  Desired tolerance value (double float precision)

set-surface-approximation-tolerance [Function]

This function is used to set the tolerance value for generating nurb data for a surface. If never called the default tolerance is 10 to the power -5.

Arguments:

- **tolerance**

  Desired tolerance value (double float precision)

Import functionality

The following described functionality that allows importing DXF, IGES and STEP geometry into AML. Once a geometry is imported into AML it is treated like a native AML geometry that may be queried, modified, scaled, and used in boolean geometric operations.

aml-face-object [class]

This class is used by the import-geometry-class and the different geometry import functions. Instances of this class are automatically generated by the import-geometry-class object or the different import-geometry functions (e.g. iges-to-aml). This class represents the mapping of a curve bounded surface. This class should not be instantiated by the user, but contains certain propeties that the user should be aware of.

Inheritance: name-generator

Properties:

- **face-tolerance**

  When using the trimming option to generate the face geometry this is the trimming tolerance.

- **edge-tolerance**

  When the face geometry is generated using the topology, this tolerance is used for repairing the edges (projecting the edge to the corresponding face).

- **force-face-topology?**

  If set to t then the system will not perform a geom-check on the geometry that was generated using the topology, else a geom-check is performed and if problems are detected with the face then the generate-face-using-trimming? property will be set to t, and the system will attempt to generate the geometry using trimming.

- **generate-face-using-trimming?**

  If set to t then the face geometry is generated using trimming, else the topology data is used to generate the geometry.

  Default Formula: (default nil)

aml-solid-object [class]

Instances of this class are automatically generated by the import-geometry-class object or the different import-geometry functions (e.g. iges-to-aml). This class represents the mapping of a solid entity to an AML-SOLID-OBJECT instance. This class should not be instantiated by the user, but contains propeties that the user should be aware of. The solid is made up of aml-shell-object instance(s), and each shell is made up of a set of aml-face-object instances.

Inheritance: imported-geometry-class, geom-object, name-generator

Properties:

- **face-tolerance**

  When using the trimming option to generate the face geometry this is the trimming tolerance.

- **edge-tolerance**

Used with the face instances that make up the shell(s). When the face geometry is generated using the topology, this tolerance is used for repairing the edges (projecting the edge to the corresponding face).

- **tolerance**

  When the solid geometry is generated by sewing, this property defines the sewing tolerance.

- **force-face-topology?**

  If set to t then the system will not perform a geom-check on the geometry that was generated using the topology, else a geom-check is performed and if problems are detected with the face then the generate-face-using-trimming? property will be set to t, and the system will attempt to generate the geometry using trimming.

- **force-solid-topology?**

  If set to t then the system will not perform a geom-check on the geometry that was generated using the topology, else a geom-check is performed and if problems are detected with the face then the generate-solid-using-sewing? property will be set to t, and the system will attempt to generate the geometry using sewing.

- **generate-face-using-trimming?**

  If set to t then the face geometry is generated using trimming, else the topology data is used to generate the geometry.

  Default Formula: (default nil)

- **generate-solid-by-sewing?**

  If set to t then the solid geometry is generated using sewing, else the topology data is used to generate the geometry.

import-geometry-class [class]

This class is used to translate geometry files in DXF, IGES, or STEP format into AML geometry objects. This class allows control over some of the translation aspects. When instantiated, this class also serves as the parent object; all new translated/imported instances are added as subobjects of this instance.

Inheritance: name-generator, coordinate-system-class

Properties:

- **filename**

  File path-name string of the geometry file to be imported.

- **geometry-type**

  Format of geometry to import. Values allowed are 'iges, 'step, and 'dxf.

  Default Formula: 'iges

- **set-tolerance?**

  When t, this property sets the tolerance of the translated AML instances based on the tolerance specified in the imported geometry file.

  Default Formula: nil

- **map-to-web-flg**

If set to 1 the curve bounded surfaces are mapped to webs. If 0 curve bounded surfaces come in as AML faces.

Default Formula: 0

- **map-to-web-tolerance**

  Specifies the maximum deviation between the original curve bounded faces and the generated webs.

  Default Formula: 1.0

- **map-to-uniform-flg**

  If set to 1 the nurb surfaces are mapped to uniform surfaces.

  Default Formula: 0

- **map-to-uniform-tolerance**

  Specifies the maximum deviation between the original nurb surface and the mapped nurb surface.

  Default Formula: 0.1

- **model-unit**

  If the model unit from the imported file is "Data Transfer", setting the desired modelunit has no effect.

- **map-to-reduce-flg**

  If set to 1 the the system tries to reduce the number of control points used to define the nurb surfaces.

  Default Formula: 0

- **map-to-reduce-tolerance**

  Specifies the maximum deviation tolerance between the original surface and the mapped surface.

  Default Formula: 0.1

- **scale-factor**

  Scale factor to allow user to scale the imported geometries.

  Default Formula: 1.0

- **imported-objects**

  The list of object instances resulting from importing the geometries. The imported objects are children of the import-geometry-class instance. Changing any of the input properties will lead to the automatic deletion of the imported objects. The "reference-coordinate-system" property of all imported objects points to the import-geometry-class instance that generated them. Note that this property must be demanded in order to trigger the procedure that performs the translation of the geometry and the creation of the imported subobjects.

  Default Formula: (import-geometry (the superior))

- **layers-to-draw-list**

  Can be 'all or a list of layer numbers. Only sub-objects whose layer is in the layers-to-draw-list are drawn.

  Default Formula: (default 'all)

- **generate-face-using-trimming?**

If set to t, face is generated by trimming underlying surface. Recommended value nil.

Default Formula: (default nil)

- **generate-solid-by-sewing?**

  If set to t, solid is generated by sewing the faces. Recommended value is nil.

  Default Formula: (default nil)

- **generate-shell-by-sewing?**

  If set to t, shell is generated by sewing the faces. Recommended value is nil.

  Default Formula: (default nil)

- **input-unit**

  This is the model unit specified in the imported geometry file. This property is set for the user after the model is retrieved. The user should not set this property.

- **force-face-topology?**

  If set to t, the system does not try to generate the surface geometry by sewing, in case the geom check fails.

- **force-solid-topology?**

  If set to t, the system does not try to generate the solid geometry by sewing, in case the geom check fails.

Remark: Simply Instantiating this class does not trigger the method that performs the translation of the geometry and the creation of the imported subobjects. The property imported-objects must be demanded for the procedure run. Remarks about model-unit, input-unit and scale-factor: The model-unit property allows the user to set the unit of the imported model. Internally the input-unit and model-unit are used to compute a scale factor. Setting the model-unit to 'data-transfer results in an internally computed scale-factor of 1 regardless of the input-unit. For example if the input unit is feet and the model-unit is inches a scale-factor of twelve is applied to data. If the scale-factor property is set to a valid number then that value will be used to scale the imported objects ignoring the input-unit and model-unit.

dxf-to-aml [Function]

This function is used to create AML geometry objects translated from the geometry defined in a DXF file. The use of the class import-geometry-class instead of this function is recommended.

Keyword Arguments:

- **:addto**

  Object instance where the new translated DXF instances will be added.

- **:infile**

  Name string of the DXF file to be translated. Can also specify the full path string in case :path is nil.

- **:path**

  The path string of the directory that contains the DXF file. The default means that the :infile is expected to specify the full path of the file.

  Default Value: nil

- **show-underlying-surface-objects?**

  When set to t, the underlying surface defining a face is added as a subobject to the face instead of being added as a property.

iges-to-aml [Function]

This function is used to create AML geometry objects translated from the geometry defined in an IGES file. The use of the class import-geometry-class instead of this function is recommended.

Keyword Arguments:

- **:addto**

  Object instance where the new translated IGES instances will be added.

- **:infile**

  Name string of the IGES file to be translated. Can also specify the full path string in case :path is nil.

- **:path**

  The path string of the directory that contains the IGES file. The default means that the :infile is expected to specify the full path of the file.

  Default Value: nil

- **show-underlying-surface-objects?**

  When set to t, the underlying surface defining a face is added as a subobject to the face instead of being added as a property.

Example 14-1. Iges-to-aml example

```
> (create-model 'name-generator)
;; creates a new name generator model
> (IGES-TO-AML :addto (the) :infile "file.igs")
;; returns :
/d/usr1/technosoft/iges/sunbin/iges-aml ~/file.igs ~/.amlout
Initializing the models
Setting the models
GDX Model Initialized
Initialized models
IGES Model Initialized
AML Model Initialized
Setting the Mappings
Parsing /users/mike/file.igs
...
```

step-to-aml [Function]

This function is used to create AML geometry objects translated from the geometry defined in a STEP file. The use of the class import-geometry-class instead of this function is recommended.

Keyword Arguments:

- **:addto**

Object instance where the new translated STEP instances will be added.

- **:infile**

  Name string of the STEP file to be translated. Can also specify the full path string in case :path is nil.

- **:path**

  The path string of the directory that contains the STEP file. The default means that the :infile is expected to specify the full path of the file.

  Default Value: nil

- **show-underlying-surface-objects?**

  When set to t, the underlying surface defining a face is added as a subobject to the face instead of being added as a property.

Chapter 15. Tagging

Classes and Methods

tagging-object [class]

The tagging-object implements geometry attribute tagging and tag propagation. All geometry classes that inherit from this class will be tagged. Those tags will be propagated through geometric operations to allow resultant geometry to refer back to the original geometry tags.

Inheritance: object

Properties:

- **source-object**

  The object that is being tagged. Tagging object will usually be a superclass of the instance being created so that the instance will be its own source-object.

  Default Formula: the graphic-object

- **id-tag**

  An identifier used to associate geometry with the object. This value is set by the system and should be only queried not set at this point. The value is set after the geom property has been demanded. To insure the value exists a reference can be made to (the geom).

  Default Formula: nil

- **tag-dimensions**

  This determines which entities of the geometry is to be tagged. The default means a solid object tags the solid, a surface tags the surfaces, a wire tags the edges, and points tags points. If the value is a list '(0 1 2 3) all points, edges, surfaces, solids associated with a geometry will be tagged.

  Default Formula: nil

- **tag-attributes**

  Attribute list associated with the object. At present, these attributes are used in the context of meshing. The list includes in order: maximum edge size, minimum edge size, curvature refinement value (0 for off 2 for on), curvature approximation error, segment value (0 for off 1 for on), segment size, and entity tolerance. By default, these attributes will be applied to all tagged points, edges, faces, and solids associated with the geometry.

  Default Formula: '(0.25 0.0625 0 0.1 0 10.0 1e-5).

- **0d-tags**

  Attribute list for tagged points associated with the geometry. It is specified as a list of lists with the first element in each sublist indicating the type of attribute and the second, the value of that attribute. For example, '((:max-e-size 0.5) (:min-e-size 0.2) (:curvflag 2) (:curv-value 0.1) (:seg-flag 1) (:seg-value 10.0)). Any attributes specified here will be applied to tagged points and those which are not will take their value from the tag-attributes property.

  Default Formula: (default nil)

- **1d-tags**

  Specified exactly like 0d-tags. Attributes specified here are applied to tagged edges of the geometry.

  Default Formula: (default nil)

- **2d-tags**

  Specified exactly like 0d-tags. Attributes specified here are applied to tagged faces of the geometry.

  Default Formula: (default nil)

- **3d-tags**

  Specified exactly like 0d-tags. Attributes specified here are applied to tagged solids of the geometry.

  Default Formula: (default nil)

- **sub-geoms-indices-list**

  This property is used to tag specific sub-geoms of a geom of specified tag-dimensions instead of the earlier behavior of tagging every sub-geom of the specified tag-dimensions. The value of the property must either be nil or a list corresponding to the tag-dimensions property. If the value is non nil, it is a list of lists, where each list is a set of indices of sub-geoms of dimension corresponding to the tag-dimensions list.

- **overwrite-other-tags?**

  If overwrite-other-tags? is t, and the tagging-object results in new sub-geoms, every sub-geom (of dimension specified by the tag-dimensions property) will be tagged with a new id. If overwrite-other-tags? is nil, and the tagging-object results in new sub-geoms, every sub-geom (of dimension specified by the tag-dimensions property) will retain the id that was propagated into it (or will be tagged with a new id ONLY if one does not exist already). If overwrite-other-tags? is a list, it is expected to be a sequence of t and nil values, corresponding to the tag-dimensions property. So, if tag-dimensions is '(0 1 2 3), then overwrite-other-tags? can be t, nil, or a list of 4 values, each being t or nil. In the case of it being a list, each item corresponds to the item in the the tag-dimensions property and determines if sub-geoms of that dimension will be tagged with a new id or will get tags propagated to it (or will be tagged with new ids ONLY if one does not exist already).

  Default Formula: nil

Note: All system geometric objects have a tagged counterpart such as tagged-box-object, tagged-cylinder-object, tagged-difference-object, tagged-extrusion-object, etc. Assume the following model is created :

```
model
  tagged-box
     tag-dimensions '(0 1 2 3)
     solid? t
  tagged-cylinder
     tag-dimensions '(0 1 2 3)
     solid? t
  tagged-difference
     object-list (list ^^tagged-box ^^tagged-cylinder)
     tag-dimensions '(0 1 2)
     overwrite-other-tags '(t nil t)
```

In this case every vertex or face (dimension 0/2) of the difference will get a new tag. Every edge of the difference that had an originaltag will retain it and NOT overwrite it and any newly created edges will be the only ones that will get a new tag. Nothing will be affected on the solid tag(s) of the difference, they will keep any propagated ids and will not attempt to tag anything new because tag-dimensions does not include 3. In case tag-dimensions is nil on the difference, the overwrite-other-tags can be t or nil or a list of a single t or nil which will be interpreted as the overwrite flag for the dimension of the difference (3).

Example 15-1. Sub-geoms-indices-list example

```
In the example below, look at the following 4 cases:
a) In object box, only 4 faces will be tagged, specifically those with sub-
geom
indices 0,1,2,3

b) In object retagged-box, the box from case (a) is being copied and moved,
and
two of its faces are being retagged.

c) In object box-2, only edges 0,1,2,3 will be tagged but all the faces will
be
tagged.

d) In object retagged-box-2, all edges except 0,1,2,3 will be tagged and the
first face will be retagged.


Example:
(define-class my-test
  :inherit-from (object)
  :subobjects (
               (box :class '(tagging-object box-object)
                        tag-dimensions '(2)
                        sub-geoms-indices-list  '((0 1 2 3))
                        )
               (retagged-box :class '(tagging-object geom-copy-object)
                        tag-dimensions          '(2)
                        source-object           ^^box
                        orientation             (list (translate '(2 0 0)))
                        sub-geoms-indices-list  '((0 1))
                        overwrite-other-tags?   t
                        )
               (box-2 :class '(tagging-object box-object)
                        tag-dimensions '(1 2)
                        sub-geoms-indices-list  '((0 1 2 3) nil)
                        orientation             (list (translate '(4 0 0)))
                        )
               (retagged-box-2 :class '(tagging-object geom-copy-object)
                        tag-dimensions          '(1 2)
                        source-object           ^^box-2
                        orientation             (list (translate '(2 0 0)))
                        sub-geoms-indices-list  '(nil (0))
```

```
                        overwrite-other-tags?   '(nil t)
                        )
                )
  )
```

get-tags [Method]

This method returns a list of tag attributes for all entities tagged on the geometry.

Arguments:

- **object**

  Instance of type tagging-object.

given-tag [Method]

This method returns the geom id associated with a specific id tag.

Arguments:

- **object**

  Instance of type tagging-object.

Example 15-2. Given tag example

```
(define-class TAGGED-OBJECT-EXAMPLE
  :inherit-from (object)
  :properties (
               min-edge-size 0.05
               max-edge-size 0.1
               mesh-object ^mesh
               )
  :subobjects (
               (box :class 'tagged-box-object
                    tag-dimensions '(0 1 2 3)
                    tag-attributes (list ^^max-edge-size ^^min-edge-size
                                         0 0.1 0 20.0 1.0e-5)
                    (mesh :class 'mesh-object
                          object-to-mesh ^^box
                          )
                    )
               )
  )
```

Chapter 16. Graphical User Interface : Primitives

Overview :

Chapter 16 describes base widget classes and functionality used to build a graphical user interface (GUI) for AML models.

Naming convention

Unless specified otherwise:

1- A class with a "-class" name suffix is a class that can be instantiated or inherited from directly by the user.

2- A class with a "-mixin" name suffix is typically mixed (multiple inheritance) with another instantiable class in the context of defining a user class.

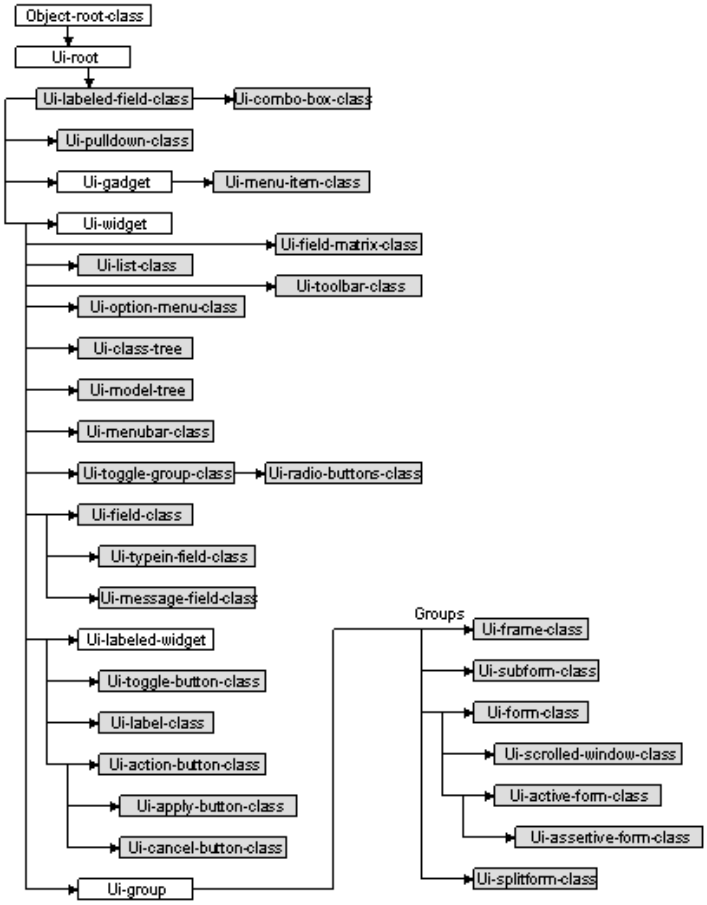
3- A class with a "-superclass" name suffix is a parent class that a group of instantiable classes inherit from. A super-class typically provides the properties and functionality that are shared by a group of sub-classes. For example: A box class is a super-class of a cube class. The user can inherit from a super-class.

Other classes described should not be instantiated or directly inherited from.

Class Hierarchy Tree

The figure below shows the hierarchy of the GUI base classes. The filled boxes represent classes that can be directly instantiated by the user.

Figure 16-1. Class Hierarchy Tree.



Root Classes

Classes and Methods

Root classes are not instantiated by the user. They provide generic functionality to instantiable classes that inherit from them.

ui-root [class]

Super-class of all GUI Base classes. Provides all GUI classes with the capability to point to the external structure holding the native user interface entities (I.e. Motif or WINDOW SDK), and manages its creation and memory freeing on deletion. Properties defined by ui-root are not input properties; the user should not redefine their formulas or set their values.

Inheritance: object-root-class

Properties:

- **Tsi-entity**

Holds the C structure pointer. Should not be changed by the user.

- **Update?**

  This property can be referenced to update the display of a user interface entity according to the current property values of its corresponding AML object instance. When referenced, the property update? calls the update method only if an update is needed.

  Default Formula: (update (the superior))

display [Method]

Defined on Classes:

ui-root

Displays a GUI entity on the screen.

See Also: hide, displayed?

displayed? [Method]

Defined on Classes:

ui-root

Returns t if the GUI entity corresponding to the ui-root instance specified is displayed on the screen. Returns nil otherwise.

See Also: display, hide

drop-tree-item-method [Method]

Defined on Classes:

ui-root

Any GUI tree item can be dragged with the left mouse button and dropped on any AML GUI widget. When the item is dropped, the system automatically calls drop-tree-item-method. By default this method does not execute anything. The user can override the method on his/her own widget class in order to define the widget's behavior when a tree item is dropped on it.

Arguments:

- **item**

  Item that has been dropped on the widget. In the case of an AML model tree item, this is the object instance.

- **x**

  X coordinate (relative to the left side of the widget), in pixels, of the location where the item was dropped.

- **y**

  Y coordinate (relative to the top side of the widget), in pixels, of the location where the item was dropped.

Example 16-1. Drop-tree-item-method example

```
(define-class my-form-class
  :inherit-from(ui-form-class)
  )

(define-method drop-tree-item-method my-form-class (object x y)
  (when (typep object 'aml-class)
    (message-box "Message" (format nil "The object ~a has been dropped on the
form at location ~a ~a." (find-tree object) x y)))
 )
```

get-value [Method]

This method is used with most user interface widgets that accept/use input. It returns the current status/value of the widget. The type of return value varies from widget class to the other. Please refer to the individual widget classes later in this chapter for more information on get-value.

grayed? [Method]

Defined on Classes:

ui-root

Returns t if the widget corresponding to the ui-root instance specified is disabled (grayed out). Returns nil otherwise.

See Also: gray-widget, ungray-widget

has-been-created? [Method]

Defined on Classes:

ui-root

Returns t when the native Motif/Windows entity corresponding to the ui-root instance has been created. Returns nil otherwise.

hide [Method]

Defined on Classes:

ui-root

Hides/Closes a GUI entity. The entity can be redisplayed using the method DISPLAY.

See Also: display, displayed?

update [Method]

Defined on Classes:

ui-root

Updates the attributes of an entity according to the current property values of its corresponding AML object instance. Update needs to be called only when any property value controlling a display attribute of a widget has been changed or smashed.

ui-widget [class]

Ui-widget is the super-class of all GUI widgets. A GUI widget is a GUI entity that can control its own size/position and appearance attributes. All subclasses of ui-widget base their general appearance (position, size, and color...) on the properties defined by ui-widget. Unless stated otherwise, all widgets are instantiated as subobjects of a ui-group instance. Note: Some of the properties defined for ui-widget are irrelevant for specific widgets. For all color properties, refer to the AML color names appendix.

Inheritance: ui-root

Properties:

- **gray?**

  When t, widget is disabled and appears grayed out.

  Default Formula: nil

- **font**

  Takes a string specifying the X font used for the widget text. This property is effective on Unix Platforms only.

  Default Formula: (default)

- **light-slope-color**

  Specifies the color of the light shadow area of the widget. This property exists on UNIX platforms only.

  Default Formula: (default)

- **dark-slope-color**

  Defaults to (default). Specifies the color of the dark shadow area of the widget. This property exists on UNIX platforms only.

  Default Formula: (default)

- **selected-color**

  Specifies the color of the widget when selected/pressed/toggled...etc. This property exists on UNIX platforms only.

  Default Formula: (default)

- **background-color**

  Specifies the background color of the widget.

  Default Formula: (default)

- **foreground-color**

  Specifies the text color of the widget.

  Default Formula: (default)

- **measurement**

Can take a value of 'pixels or 'percentage. When 'pixels, the x-offset, y-offset, width and height of the widget represent pixel values. When 'percentage, the x-offset, y-offset, width and height of the widget represent a percentage value of the width/height of the superior of the current widget instance. Also, when a percentage measurement is specified, the widget is attached to its parent, i.e. the widget grows and shrinks with its parent window when that window is resized with the mouse.

Default Formula: (default 'percentage)

- **attachemnt-info-list**

  This property sets the location of the widget with respect to its parent widget. It is made up of list of the following: ;;; Top side of widget related to the parent widget (top or bottom) ;;; Bottom side of widget related to the parent widget (top or bottom) ;;; Left side of widget related to the parent widget (left or right) ;;; Right side of widget related to the parent widget (left or right) The "attachment-info-list" property is only relevant when the corresponding "measurement" property is 'pixels. This property is not effective when the superior/parent object of the widget it belongs to is an instance of ui-scrolled-window-class (or any other scrollable window class).

- **x-offset**

  Integer representing the offset of the widget from the left side of its parent.

  Default Formula: 0

- **y-offset**

  Integer representing the offset of the widget from the top side of its parent.

  Default Formula: 0

- **width**

  Integer specifying the width of the widget

  Default Formula: 10

- **height**

  Integer specifying the height of the widget

  Default Formula: 10

- **tooltip**

  Tooltip string. A tooltip is a "help" string that pops up when a button gains the focus of the mouse cursor and the cursor is not moved for a certain time. The tooltip functionality is defined with most widgets. Some exceptions are: ui-field-class, ui-option-menu-class and their descendant classes.

See Also: default

get-window-coordinates [Method]

Defined on Classes:

ui-widget

Returns the current global pixel coordinates (W.R.T the screen) as an '(x y width height) list. The coordinate origin is always the top left corner of the screen.

gray-widget [Method]

Defined on Classes:

ui-widget

Disables a widget (grays it out) making it unelectable. The method also changes the value of the gray? property to t. This method is irrelevant on certain types of widgets like forms.

See Also: update, ungray-widget, grayed?

set-availability [Method]

Defined on Classes:

ui-widget

Updates the enabled/disabled status of the widget according to the gray? property value.

See Also: update

ungray-widget [Method]

Defined on Classes:

ui-widget

Enables a widget (making it selectable). The method also changes the value of the gray? property to nil. This method is irrelevant on certain types of widgets like forms.

See Also: gray-widget, grayed?

update-colors [Method]

Defined on Classes:

ui-widget

Updates the color display of the widget according to the foreground-color and background-color property values.

See Also: update

set-focus [Method]

Defined on Classes:

ui-widget

This method simply sets the keyboard focus on the given ui-widget instance. It takes no other parameters.

ui-labeled-widget [class]

A ui-labeled-widget is a ui-widget that has the capability of having a label or an image displayed on it.

Inheritance: ui-widget

Properties:

- **label**

String specifying the label text displayed on the widget.

Default Formula: (write-to-string (object-name (the superior)))

- **image**

  File path string of a bitmap image file to be displayed on the widget. An XBM or XPM format is required on UNIX platforms. A BMP format is required on WINDOWS platforms. When a valid image file is specified, the label is ignored.

  Default Formula: ""

- **label-align**

  Keyword specifying the text justification of the widget label. It can also take the values of :left or :right.

  Default Formula: :center

- **transparent-image-color**

  This property defines the transparent color of the image (bmp file) associated with the widget. The transparent color of an image is the color that is replaced by the background-color of the widget it is displayed on. This property is effective on WINDOWS platforms only.

  Default Formula: (default "RGB(192,192,192)")

set-transparent-image-color [Method]

Defined on Classes:

ui-labeled-widget

This method allows to turn a color used by the image of a widget to become transparent.

Arguments:

- **color**

  The color parameter should be a color name (string) or an RGB specification string (Example: "RGB(192,192,192)").

Example 16-2. Set-transparent-image-color example

```
        (set-transparent-image-color (the interface forms form1 button1)
"gray")
```

update-label [Method]

Defined on Classes:

ui-labeled-widget

Updates the display of the label/image on a widget according to the current values of the label, image, and label-align properties. This method is automatically called when the method update is called on a ui-labeled-widget instance. However, for efficiency reasons, it can be used individually. In other words, if the AML user knows that only the label or image property values have changed, calling update-label on the widget instance would be more efficient than calling update.

See Also: update

visible? [Method]

Defined on Classes:

ui-labeled-widget

This method returns t if the given widget is currently visible or at least partially exposed. It returns nil otherwise.

ui-gadget [class]

Ui-gadget is the super-class of all GUI gadgets. A GUI gadget is a GUI entity that derives its position, size and most appearance attributes from its parent.

Inheritance: ui-root

Properties:

- **font**

  Takes a string specifying the X font used for the widget text. This property is effective on Unix Platforms only.

  Default Formula: (default)

- **label**

  String specifying the label text displayed on the widget.

  Default Formula: (write-to-string (object-name (the superior)))

- **gray?**

  When t, widget is disabled and appears grayed out

  Default Formula: nil

Callbacks Management

Classes

apply-cancel-mixin [class]

Apply-cancel-mixin is inherited into widgets classes to enable the apply and cancel callback methods of the parent form object of these widgets.

Inheritance: object

Properties:

- **apply-action**

  Quoted expression to be evaluated when the apply-form-method of the superior form object is called

  Default Formula: nil

- **cancel-action**

  Quoted expression to be evaluated when the cancel-form-method of the superior form object is called.

Default Formula: nil

See Also: apply-form-method, cancel-form-method source code sample G

ui-callback [class]

Ui-callback is inherited into widget classes that require callback functionality

Inheritance: object

See Also: get-value

Group Widget Classes

Classes and Methods

This section describes the classes of ui-widgets used to group and manage other ui-widgets. Typically, a group widget class is instantiated as a parent (i.e. superior) object to other ui-widgets.

ui-group [class]

Ui-group is the super-class of all form classes. It is not directly instantiated by the user. Typically, a class that ui-group inherits into is instantiated as a superior object (i.e. an instance of ui-form-class) to other widgets and can manage their position, size, appearance, property values and callbacks. A group can also be a subobject of another group instance.

Inheritance: ui-widget, apply-cancel-mixin

See Also: Refer to the main subclasses of ui-group: ui-frame-class, ui-form-class, ui-scrolled-window-class, ui-subform-class

lower [Method]

Defined on Classes:

ui-group

Sends a Top-level form to the background of the screen. Only the parts of the form that no other window is overlapping stay visible.

See Also: top-level-form?

raise [Method]

Defined on Classes:

ui-group

Sends a Top-level form to the front of the screen making it totally visible.

See Also: top-level-form?

top-level-form? [Method]

Defined on Classes:

ui-group

Returns t when called on a ui-group instance that is not a subobject of another ui-group, returns nil otherwise. Top level forms are created inside a stand-alone window with a border and a title bar. They also have the capability of being individually moved, resized, and minimized with the mouse.

apply-form-method [Method]

Defined on Classes:

ui-group

This method simply loops in the immediate apply-cancel-mixin children of the group and executes their apply-action.

See Also: cancel-form-method, ui-apply-button-class, apply-cancel-mixin

cancel-form-method [Method]

Defined on Classes:

ui-group

This method simply loops in the immediate apply-cancel-mixin children of the group and executes their cancel-action.

Arguments:

- **instance**

  instance of ui-group

Keyword Arguments:

- **:close?**

  When t, the form is hidden at the end of the method execution.

  Default Value: nil

- **:update-form?**

  When t, the form is updated (by calling update) at the end of the method execution.

  Default Value: nil

See Also: apply-form-method, ui-cancel-button-class, apply-cancel-mixin.

ui-form-class [class]

A ui-form-class is a ui-group that can be instantiated both as a top-level-form? or as a subobject of another ui-group. A ui-form-class is typically instantiated as a top-level form. When displayed, it is automatically created inside a stand-alone window with a border and a title bar. Note: A ui-form-class instance, when instantiated as a top-level form, ignores the value of its measurement property: Its x-offset, y-offset, width and height properties are pixel values. However, setting the measurement property of a top-level form is very useful since the measurement property value of its children widgets is directly derived from their parent form.

Inheritance: ui-group

Properties:

- **label**

String specifying the label appearing on the title bar of the form. This property is ignored when the form does not have a title bar, i.e. when it is a subobject of another form.

Default Formula: (write-to-string (object-name (the superior)))

- **resize?**

  When nil, the form window cannot be resized with the mouse, otherwise the window is resizable.

  Default Formula: t

- **shrink?**

  A value of t allows the user to minimize the form individually into an icon. A value of nil will disable this feature.

  Default Formula: Defaults to t on WINDOWS platforms and nil on UNIX platforms.

- **image**

  File path string specifying the icon image when the form is minimized. On WINDOWS platforms, this image is also visible on the title bar of the form even when the form is displayed. An ico icon file should be used on WINDOWS, and an xbm or xpm should be used on UNIX.

  Default Formula: t

- **button3-action**

  This property expects a quoted AML expression that is evaluated when the user right-clicks on an empty area of the form. An empty area is an area that is not occupied by a child widget.

  Default Formula: nil

- **close-action**

  This property expects a quoted AML expression that is evaluated when the user close s the form from its title bar (x button on WINDOWS)..

  Default Formula: '(hide !superior)

See Also: top-level-form?, source code sample A, source code sample E

Example 16-3. Ui-form-class example

```
(define-class ui-callback-form-class
  :inherit-from(ui-form-class)
  :properties(
              label "Testing a Form's Callbacks"
              x-offset 40
              y-offset 40
              width    300
              height   300

              button3-action '(let ((choice (pop-up-menu '("1" "2" "3")))
                                    )
                                (when choice
                                  (message-box
                                   "Message"
                                   (format nil "Option ~a has been selected"
choice))
```

```
                                    ))

              close-action '(when (string= (pop-up-message
                                            "Are you sure you want to close
the form ?"
                                            :done-label "Yes"
                                            :cancel-label "No")
                                           "Yes")
                                  (hide !superior)
                                  )
              )
  )
```

border-width [Method]

Defined on Classes:

ui-form-class

This method returns the border width of a top-level form. This value is platform dependent. The border-width can be defined as the number of pixels between the x-offset of a top-level form and the position where a child widget whose x-offset is 0 is located.

set-window-coordinates [Method]

Defined on Classes:

ui-form-class

Positions and resizes a top-level ui-form-class instance according to its current x-offset, y-offset, width and height properties.

title-bar-height [Method]

Defined on Classes:

ui-form-class

The method returns the height in pixels of the title bar of a top-level form. This is a platform dependent value. On WINDOWS, the returned value includes the menu bar height in case the top-level form specified contains a menu bar.

ui-frame-class [class]

Ui-frame-class is typically instantiated as a parent of other widgets. It provides a decorative frame around its subobjects. A ui-frame-class should be a subobject of another group widget. An instance of ui-frame-class is equivalent to an instance of ui-subform-class with the frame? property set to t.

Inheritance: ui-group

Properties:

- **display?**

Allows the user to control whether the frame (and its components) needs to be displayed or not. When t, the frame is displayed when its parent form is updated. When nil, the frame is hidden when its parent form is updated

Default Formula: t

- **x-margin**

  Specifies the horizontal margin affecting the available area for its widget subobjects.

  Default Formula: 2

- **y-margin**

  Specifies the vertical margin affecting the available area for its widget subobjects. This property is ignored when frame? is nil.

  Default Formula: 2

- **shadow-type**

  Specified the style of the frame border. Possible values are :inset, :outset, :double-inset, :double-outset.

  Default Formula: :inset

See Also: ui-subform-class

ui-active-form-class [class]

A ui-active-form-class is a ui-form-class that features the following additional behavior: When the method display is called on a ui-active-form-class instance, the method does not return until the user hides the form. A ui-active-form-class has to be instantiated as a top-level-form?. The AML programmer should provide a button on the form that hides it when clicked.

Inheritance: ui-form-class

ui-assertive-form-class [class]

A ui-assertive-form-class is a ui-active-form-class that features the following additional behavior: When displayed, it disables (or grays out) any other AML window until it is hidden. A ui-assertive-form-class has to be instantiated as a top-level-form?. The AML programmer should provide a button on the form that hides it when clicked.

Inheritance: ui-active-form-class

ui-scrolled-window-class [class]

A ui-scrolled-window-class is a ui-form-class that provides a scrolling area capable of containing a bigger form. A ui-scrolled-window-class can be instantiated as a top-level form or as a subobject of another form.

Inheritance: ui-form-class

See Also: source code sample F

ui-splitform-class [class]

This class manages two resizable panes. These 2 panes are instantiated as subobjects of the split form. They divide it horizontally or vertically occupying its whole area. The user will be able to drag the divider bar with the mouse in order to interactively resize the panes. This class cannot be instantiated as a top-level form and must always have two subobjects of class ui-resizable-pane-mixin.

Inheritance: ui-group

Properties:

- **Divider-bar-thickness**

  Thickness of the divider bar in pixels.

  Default Formula: 6

- **Divider-offset**

  Initial position of the divider bar as a percentage of the split form's height or width (depending on the orient property).

  Default Formula: 50

- **Orient**

  vertical for up and down resizing (horizontal divider bar) or :horizontal for left-right resizing (vertical divider bar).

  Default Formula: :vertical

See Also: ui-resizable-pane-mixin

ui-resizable-pane-mixin [class]

This class is instantiated only as a subobject of a ui-splitform-class. It can be "mixed in" with another predefined form/subform class as shown in the example below.

Inheritance: ui-root

Properties:

- **Background-color**

  Background color string of the pane.

  Default Formula: (default)

- **height**

  Integer specifying the height in pixels of the total scrollable area of the pane. This is effective only when scrollable? is non-nil.

- **width**

  Integer specifying the width in pixels of the total scrollable area of the pane. This is effective only when scrollable? is non-nil.

- **scrollable?**

  When non-nil, creates scroll bars if the total size of the pane does not fit within its part of the parent split form.

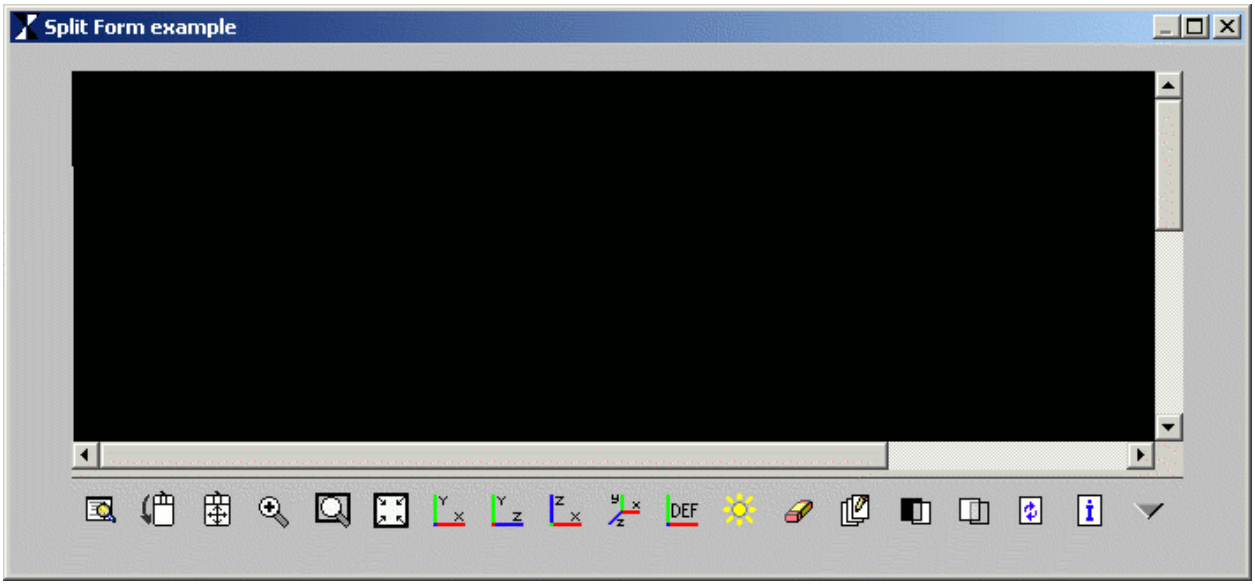
Default Formula: nil

- **Scroll-bar-color**

  Color string of the scroll bars. This is effective on Unix platforms only.

  Default Formula: (default "gray70")

Figure 16-2. Split form example



Example 16-4. Ui-resizable-pane-mixin example

```
(in-package :aml)

(define-class pane1-example-class
  :inherit-from(ui-resizable-pane-mixin)
  :properties(
              width 800
              height 500
              )
  :subobjects(
              (canvas :class 'ui-canvas-class
                      x-offset 0 y-offset 0 width ^^width height ^^height
                      measurement 'pixels
                      )
              )
  )

(define-class pane2-example-class
  :inherit-from(ui-resizable-pane-mixin ui-graphic-control-toolbar-class)
  :properties(
              canvas-object (default nil)
              )
  )
```

```
(in-package :aml)

(define-class split-form-area-class
  :inherit-from(ui-splitform-class)
  :properties(
              divider-bar-thickness 5
              orient                :vertical
              divider-offset        84
              canvas-object         (the superior pane1 canvas)
              )
  :subobjects(
              (pane1 :class 'pane1-example-class
                     scrollable? t
                     )
              (pane2 :class 'pane2-example-class
                     scrollable? nil
                     )
              )
  )

(define-class split-form-example-class
  :inherit-from(ui-form-class)
  :properties(
              label "Split Form example"
              x-offset 20
              y-offset 20
              width 700
              height 300
              )
  :subobjects(
              (split-area :class 'split-form-area-class
                          x-offset 5
                          y-offset 5
                          width 90
                          height 90
                          )
              )
  )

(define-method display split-form-example-class ()
  (call-next-method)
  (activate-display (the split-area canvas-object))
  )
```

ui-subform-class [class]

A ui-subform-class is a ui-group that can only be instantiated as a subobject of another ui-group. It cannot be a top-level form. A ui-subform-class can be instantiated with a decorative frame around it.

Inheritance: ui-group

Properties:

- **display?**

Allows the user to control whether the sub-form needs to be displayed or not. When t, the sub-form is displayed when its parent form is updated. When nil, the sub-form is hidden when its parent form is updated.

Default Formula: t

- **frame?**

  When t, the subform is created with a decorative frame around it.

  Default Formula: nil

- **x-margin**

  When the subform is framed, this property specifies the horizontal margin affecting the available area for its widget subobjects. This property is ignored when frame? is nil.

  Default Formula: 2

- **y-margin**

  When the subform is framed, this property specifies the vertical margin affecting the available area for its widget subobjects. This property is ignored when frame? is nil.

  Default Formula: 2

- **shadow-type**

  Specified the style of the frame border. This property is ignored when frame? is nil. Possible values are :inset, :outset, :double-inset, :double-outset.

  Default Formula: :inset

See Also: ui-form-class, ui-frame-class

Component Widget Classes

Main Component Widgets

ui-canvas-class [class]

A Ui-canvas-class provides a drawing window for AML graphic objects. When an instance of ui-canvas-class is created and activated (see activate-display below) all graphical functions called from AML will operate on that canvas until another canvas is activated. The active canvas is sometimes referred to as the current display window. A canvas can only be instantiated as a subobject of a ui-group (i.e. form or subform).

Inheritance: ui-root

Properties:

- **Measurement**

  Can take a value of 'pixels or 'percentage. When 'pixels, the x-offset, y-offset, width and height of the canvas will represent pixel values. When 'percentage, the x-offset, y-offset, width and height of the widget will represent a percentage value of the width and height of the parent object of the ui-widget instance. Also, when a percentage measurement is specified, the canvas is attached to its parent, i.e. the canvas grows and shrinks with its parent window when that window is resized with the mouse.

Default Formula: (default 'percentage)

- **X-offset**

  Integer representing the offset of the canvas from the left side of its parent.

  Default Formula: Integer representing the offset of the canvas from the left side of its parent.0

- **Y-offset**

  Integer representing the offset of the canvas from the top side of its parent.

  Default Formula: 0

- **Width**

  Integer specifying the width of the canvas.

  Default Formula: 200

- **Height**

  Integer specifying the height of the canvas.

  Default Formula: 200

- **Camera-field**

  Initial dimensions of the view area rectangle covered by the canvas. This property is not updated when the view is modified.

  Default Formula: '(1.0 1.0)

- **Camera-target**

  Coordinates of the camera center target position. This property is not updated when the view is modified.

  Default Formula: '(0.0 0.0 0.0)

- **Camera-eye-vector**

  Coordinates of the normal vector to the viewing plane, in the direction of the "user's eyes". This property is not updated when the view is modified.

  Default Formula: '(0.0 0.0 1.0)

- **Camera-up-vector**

  Coordinates of the vector in the view plane determining the camera's upward direction. This property is not updated when the view is modified

  Default Formula: '(0.0 1.0 0.0)

- **Render**

  This property can take the values 'boundary, 'shaded, or 'isoline to enable a permanent rendering method of graphic objects on the canvas regardless of the render property value of the objects being drawn

  Default Formula: nil

See Also: current-eye-vector, current-up-vector, get-view, view-set

activate-display [Method]

Defined on Classes:

ui-canvas-class

Activates a ui-canvas-class instance. When an instance of ui-canvas-class is activated (see activate-display below), all graphical functions called from AML will operate on that instance. Only one canvas can be active at a time. Activating a canvas deactivates the previously active canvas. The function (current-display) always returns the active canvas object instance. (The active canvas is also referred to as the "current display"). A canvas must be displayed before it gets activated.

See Also: current-display

get-window-coordinates [Method]

Defined on Classes:

ui-canvas-class

See Also: get-window-coordinates defined on ui-widget.

validate-drawn-objects-table [Method]

Defined on Classes:

ui-canvas-class

This method can be called on an instance of ui-canvas-class in order to removed from its drawn-objects table all invalid (or deleted) objects. It is needed when objects that are drawn on the specified canvas have been deleted while the canvas was not active.

set-current-display-background-color [Function]

This function sets the background color of the current display (i.e. the active canvas).

Arguments:

- **Color**

  List of RGB values (between 0 and 1 inclusive) or a color name (Refer to the color names appendix in the AML reference manual)

ui-combo-box-class [class]

A ui-combo-box-class is a labeled field widget (see ui-labeled-field-class) that also features a list of menu accessible options for the content of the field. The menu is popped up by clicking on the arrow located at the right side of the field.

Inheritance: ui-labeled-field-class

Properties:

- **Options-list**

  The list of available options. When the content property is a string, options-list is a list of strings. The value nil cannot be a member of the options-list.

- **Options-button-width**

Value representing a percentage of the total width of the widget occupied by the options menu pop-up button.

Default Formula: (Default 10)

- **Label-width**

  Value representing a percentage of the total width of the widget occupied by the label area.

  Default Formula: 40

- **Button1-action**

  Quoted expression that is evaluated after a new option is selected from the option menu.

  Default Formula: nil

See Also: ui-labeled-field-class, ui-option-menu-class

ui-field-class [class]

Ui-field-class provides a framed text area used mainly to display text. The text in a ui-field-class can be editable; however, in the case of editable text (input text), it is recommended to use the ui-typein-field-class described later because it features error trapping and callback capabilities.

Inheritance: ui-widget

Properties:

- **content**

  This property indicates the text content of the field. It also defines the data type returned by the method get-value when get-value is called on an instance of ui-field-class. If the value of the content property is a string, the return value of get-value will be a string. If the value of the content property is a number or symbol, the return value of get-value will be a number or symbol. For example: If the value of the content property is 52 (a number), and the user types 64 in the field widget, the next time the method get-value is called on this field widget instance, it will return 64. On the other hand, if the value of the content property is "52" (a string), and the user types 64 in the field widget, the method get-value will then return "64". Please note: In case the programmer intends to have a blank field, he/she can set the content property to "" (empty string) or nil. If the content property is nil, it indicates that the field values that the user will enter will be returned as numbers or symbols. If it is "", these values will be returned as strings.

  Default Formula: nil

- **editable?**

  When non-nil, the text in the field can be edited.

  Default Formula: nil

- **input-length**

  Specifies the maximum number of characters allowed in the field. A value of 0 specifies an unlimited number of characters. If this property smashes/changes at runtime, it will lead to the destruction and recreation of the widget; therefore, this practice is not recommended.

  Default Formula: 1000

- **input-rows**

When greater then 1, the field allows a multi-line text area. If this property smashes/changes at runtime, it will lead to the destruction and the recreation of the widget; therefore, this practice is not recommended.

Default Formula: 1

- **scroll-bars?**

  When t, the field forces the existence of vertical and horizontal scroll-bars useful when the field text exceeds the size of the widget.

  Default Formula: nil

- **text-align**

  It allows setting the alignment of the content of a text widget (left, center or right) in a similar manner that the label of a label widget currently works. It can take one of 3 possible values: :left, :center or :right.

  Default Formula: :left

This class includes a method get-value, which returns the current text in the field widget. When the content property is specified as a string, get-value returns a string; otherwise, get-value returns the text in the form of a number, symbol or keyword.

See Also: get-value, ui-typein-field-class

field-set-editable [Method]

Defined on Classes:

ui-field-class

Sets the editable/non-editable status of the field widget according to the value of the editable? keyword specified.

Arguments:

- **Instance**

  Instance of ui-field-class

Keyword Arguments:

- **:editable?**

  Optional keyword. Defaults to t to make the field widget editable. When nil, the field widget is becomes non-editable.

  Default Value: t

add-text [Method]

Defined on Classes:

ui-field-class

Appends the text specified to the current text in the field widget and updates the content property accordingly.

Arguments:

- **Instance**

  Instance of ui-field-class.

- **Text**

  String to be appended to the current text in the field widget.

insert-text [Method]

Defined on Classes:

ui-field-class

Inserts text into the field widget at the specified location within the existing content of the field.

Arguments:

- **Instance**

  Instance of ui-field-class

- **Text**

  String to insert

- **Index**

  Location within the existing string where the new text must start. Index starts at 0. When Index = -1, the text is inserted at the current cursor position. When index = -2. the text is appended to the current text.

replace-text [Method]

Defined on Classes:

ui-field-class

Replaces the current text in the field widget with the text specified and updates the content property accordingly.

Arguments:

- **Instance**

  Instance of ui-field-class.

- **Text**

  String to replace the current text in the field widget.

set-scroll-bar [Method]

Defined on Classes:

ui-field-class

Sets the scroll bar position of a field widget.

Arguments:

- **Instance**

  Instance of ui-field-class.

- **Indext**

  Value of 0 or 1: 0 sets the horizontal scroll bar, 1 sets the vertical scroll bar.

- **Ratio**

  A value between 0 and 1 specifying the position desired. A value of 0.0 moves the scroll bar to its start position, while a value of 1.0 moves it to its end position.

See Also: get-scroll-bar-position.

get-scroll-bar-position [Method]

Defined on Classes:

ui-field-class

Arguments:

- **Instance**

  Instance of ui-field-class.

- **Index**

  Value of 0 or 1: 0 specifies the horizontal scroll bar, 1 specifies the vertical scroll bar.

See Also: set-scroll-bar

ui-field-matrix-class [class]

Ui-field-matrix-class provides a matrix of field widgets with labels widget heading the rows and columns. Rows labels appear at the left side of the matrix, and column labels on top. The label property inherited from ui-widget specifies the label of the top-left corner area. This class requires the property measurement to be set to 'pixels.

Inheritance: ui-widget, ui-callback, apply-cancel-mixin

Properties:

- **Rows-labels-list**

  List of strings specifying the row labels.

  Default Formula: nil

- **Columns-labels-list**

  List of strings specifying the column labels.

  Default Formula: nil

- **Content**

  List of list of values, each list representing the contents of the fields in a row.

- **Num-rows**

  Integer specifying the number of rows excluding labels.

  Default Formula: 10

- **Num-columns**

  Integer specifying the number of columns excluding labels.

Default Formula: 10

- **Maximum-size**

  Maximum number of fields.

  Default Formula: 200

- **Field-width**

  Width of a field in pixels. The user has to keep in mind that there is a 3-pixel clearance between fields; the values given to field-width and width should anticipate this clearance.

  Default Formula: 60

- **Field-height**

  Height of a field in pixels. The user has to keep in mind that there is a 3-pixel clearance between fields; the values given to field-height and height should anticipate this clearance

  Default Formula: 60

- **Label-align**

  Text justification of the label area. Can be :left, :center or :right.

  Default Formula: :left

- **Labels-font**

  . Font used for the columns/rows labels. It is not recommended to change this property at runtime since it will cause the widget to be destroyed and recreated.

  Default Formula: ^font

ui-label-class [class]

This class provides a plain label area used to display text or image.

Inheritance: ui-labeled-widget

ui-labeled-field-class [class]

This class is a compound widget consisting of a label and a field area. Its field area functionality is similar to that of a ui-typein-field-class. The label area is created on the left of the field area. Note: All methods documented with ui-field-class can also be called on instances of ui-labeled-field-class and operate on the field of the ui-labeled-field-class object.

Inheritance: ui-root, apply-cancel-mixin

Label area properties

Properties:

- **label**

  String specifying the label displayed in the label area of the widget.

  Default Formula: ""

- **label-width**

  Specifies the width of the label area as a percentage of the total width of the widget.

Default Formula: 50

- **label-align**

  Text justification of the label area. Can be :left, :center or :right.

  Default Formula: :left

- **label-background-color**

  Specifies the background color of the label area.

  Default Formula: ^background-color

- **label-foreground-color**

  Specifies the text color of the label area.

  Default Formula: ^foreground-color

Field area properties

Properties:

- **accept-content-action**

  Refer to ui-typein-field-class

  Default Formula: nil

- **content**

  Refer to ui-field-class

  Default Formula: nil

- **input-length**

  Refer to ui-field-class

  Default Formula: 1000

- **input-rows**

  Refer to ui-field-class

  Default Formula: 1

- **scroll-bars?**

  Refer to ui-field-class

  Default Formula: nil

- **error-message**

  Refer to ui-typein-field-class

  Default Formula: "Error"

- **editable?**

  Refer to ui-field-class

  Default Formula: t

- **focusin-action**

Takes a quoted AML expression that will be evaluated everytime the field gains keyboard focus.

Default Formula: nil

- **focusout-validation?**

  Refer to ui-typein-field-class

  Default Formula: nil

- **validation-function**

  Refer to ui-typein-field-class

  Default Formula: nil

- **button3-action**

  Provides right click callback functionality to the text field of the class. It expects the same format as the standard button3-action property found in other UI base classes.

  Default Formula: nil

- **text-align**

  It allows setting the alignment of the content of a text widget (left, center or right) in a similar manner that the label of a label widget currently works. It can take one of 3 possible values: :left, :center or :right.

  Default Formula: :left

See Also: ui-label-class, ui-field-class, ui-typein-field-class

ui-list-class [class]

This class provides a scrollable list of selectable elements. Only one item can be selected at a time. Ui-list-class provides callback capability on a left mouse click.

Inheritance: ui-widget, ui-callback, apply-cancel-mixin

Properties:

- **Button1-action**

  Quoted expression that is evaluated when the user left-clicks on the button with mouse.

  Default Formula: nil

- **Items-list**

  List of values representing the items available

  Default Formula: nil

- **Labels-list**

  List of labels associated with the items available. Labels-list and Items-list have the same number of elements. When the list widget is displayed, every item is represented by its corresponding label in the labels-list.

  Default Formula: nil

- **Selected-item**

Specifies a value from Items-list to be the initially selected (highlighted) item. This property is updated every tiem an item is selected either programmatically (using select-item) or with a left mouse click.

Default Formula: nil

- **Selected-index**

  Holds the index of the selected item in the items-list. The index of the first element of the list is 0. The user should not redefine the formula or set the value of this property. This index is nil until the user clicks on an item in the list for the first time.

- **multiple-selection?**

  If "multiple-selection?" is 1: More than one item of the list can be selected by left-clicking on an item. At all time, selecting an item in the list does not unselect the previously selected item(s). If "multiple-selection?" is 2: More than one item of the list can be selected by left-clicking on an item while the control key is down. The user can also automatically select several consecutive items in two clicks: Left-click on the first one, then left-click on the last one while the shift key is down. Left-clicking on an item without having the control or the shift key down will unselect all previously selected items.

  Default Formula: nil

This class includes a method get-value, which returns the selected item (or item list). Items are elements of the property items-list.

See Also: get-value

set-top-item [Method]

Defined on Classes:

ui-list-class

This method is useful when some the list items are hidden and scroll bars are used for navigation through the list widget. Set-top-item moves the scroll bars in order to position the item whose index is specified as the first (top most) of the visible items.

Arguments:

- **Instance**

  Instance of ui-list-class.

- **index**

  Index of the item to be positioned as the top most visible item. The index of the first element in the Items-list is 0.

See Also: get-top-item-index

get-top-item-index [Method]

Defined on Classes:

ui-list-class

Returns the index of the current first (top most) visible item of the list widget. Indices start at 0.

See Also: set-top-item.

select-item [Method]

Defined on Classes:

ui-list-class

Programmatically selects and highlights the item whose index is specified. Indices start at 0.

Arguments:

- **Instance**

  Instance of ui-list-class.

- **Index**

  Index of item to be selected. When index=-1, all items in the list widget are deselected.

- **notify?**

  Flag that triggers the button1-action callback when non-nil.

ui-message-field-class [class]

Ui-message-field-class provides a permanent message pane area for an AML application. It is a non-editable ui-field-class.

Inheritance: ui-field-class

For more runtime message functionality, see also: pop-up-message, message-box

show-message [Method]

Defined on Classes:

ui-message-field-class

Directs the string specified into the message pane. It can replace the existing message or append to it.

Arguments:

- **Instance**

  Instance of ui-message-field-class.

- **Message**

  Message string to be displayed on instance.

Keyword Arguments:

- **:append?**

  Optional keyword. When append? is non-nil, a carriage return and the new message are appended to the existing message in the message pane; otherwise, the new message replaces the existing one.

  Default Value: nil

ui-option-menu-class [class]

This class provides an option selection widget with a button that pops up a menu of available options. One option is selected at a time. Unless the button is pressed, the widget shows the selected option only. On WINDOWS platforms, this widget is referred to sometimes as a non-editable combo box. The

options menu is activated by a left mouse click. Notes : 1. An option menu does not provide a label area on its left side (even though the standard MOTIF/UNIX widget does) in order to be compatible with the WINDOWS corresponding combo box widget. 2. The height property is ignored and will not affect the actual height of the widget. The height is platform dependent and is automatically computed. Allow the equivalent of 32 pixels vertically for each option menu button. 3. On UNIX platforms, the button of the option menu widget will not appear exactly at the x-offset specified; there will always be a clearance of about 15 pixels.

Inheritance: ui-widget

Properties:

- **Options-list**

  List of the available options. The value nil cannot be a member of the options-list.

  Default Formula: nil

- **Labels-list**

  List of strings. One string per option is needed. A label is the representation of an option on the widget.

  Default Formula: nil

- **Selected-option**

  Initial option selected. The selected-option should be an element of the options-list. The value of this property is updated every time a new option is selected. This property will always hold the currently selected option.

  Default Formula: nil

- **Button1-action**

  Quoted expression that is evaluated after a new option is selected from the option menu.

  Default Formula: nil

This class includes a method get-value

See Also: get-value

select-option [Method]

Defined on Classes:

ui-list-class

Programmatically selects the option whose index is specified. An index of 0 denotes the first option of the options-list.

Arguments:

- **Instance**

  An instance of ui-option-menu-class

- **Index**

  Index of the desired option.

- **notify?**

When non nil, invokes the button1-action callback.

ui-radio-buttons-class [class]

Ui-radio-buttons-class provides an exclusive toggle group, where only one button can be selected at a time. The buttons visual representation is different from that of a standard ui-toggle-group-class.

Inheritance: ui-toggle-group-class

Properties:

- **Status**

  Integer specifying the index of the button selected. It is updated every time the user selects a new button with a left mouse click.

  Default Formula: 0

This class includes a method get-value, which returns the index of the currently selected button. Indices start at 0.

See Also: get-value, ui-toggle-group-class

ui-richtext-file-form-class [class]

Ui-richtext-file-form-class implements a basic form that has a rich text widget. This class can read from a file.

Inheritance: ui-richtext-form-class

Properties:

- **data-file-path**

  The path of the file whose contents are to be displayed.

See Also: display-file-richtext

display-file-richtext [Function]

This function displays text.

Keyword Arguments:

- **file-path**

  File to get contents from.

- **editable?**

  When t, enables the editing of the text.

- **scroll-bars?**

  When t, the scroll-bars will appear when the text goes outside the bounds of the control.

  Default Value: t

- **centered?**

  When t, the widget is centered on the screen.

- **x**

The horizontal offset (in pixels) of the widget on the screen.

- **y**

  The vertical offset (in pixels) of the widget on the screen.

- **width**

  The width of the widget.

- **height**

  The height of the widget.

- **label**

  The label of the widget.

  Default Value: "Richtext"

- **font**

  The font of the text.

  Default Value: "8x13"

- **text-foreground-color**

  The color of the text.

  Default Value: "black"

- **text-background-color**

  The text-background-color.

  Default Value: "white"

- **topmost?**

  When t, richtext widget will stay on top of all other forms.

  Default Value: nil

- **new?**

  When t, the form is deleted and a new object instance is created. The widget is reset with the default values.

See Also: display-richtext

ui-richtext-form-class [class]

Ui-richtext-form-class implements a basic form that has a rich text widget.

Inheritance: ui-form-class

Properties:

- **content**

  The data that is inserted into the field when it is displayed.

- **editable?**

  Currently not used.

- **scroll-bars?**

  When t, the scroll-bars will show up when the text goes outside the bounds of the control.

- **text-background-color**

  The color of the background color of the text.

- **text-foreground-color**

  The color of the text.

See Also: richtext-insert, richtext-clear, richtext-highlight, richtext-unhighlight, richtext-set-text-color, richtext-set-background-color, richtext-set-text-font, richtext-set-style, display-richtext

richtext-clear [Method]

Defined on Classes:

ui-richtext-form-class

This method allows the user to clear the text that is currently in the form's widget.

See Also: richtext-insert, richtext-clear, richtext-highlight, richtext-unhighlight, richtext-set-text-color, richtext-set-background-color, richtext-set-text-font, richtext-set-style, display-richtext

richtext-highlight [Method]

Defined on Classes:

ui-richtext-form-class

This method allows the user to highlight a region of text that is currently in the form's widget.

Arguments:

- **start-index**

  The index of where the highlighting will begin(0 based indexing inclusive).

- **end-index**

  The index where the highlighting will end (0 based indexing exclusive).

Keyword Arguments:

- **color**

  The color to highlight the text with. If nil, the corresponding property is used.

See Also: richtext-insert, richtext-clear, richtext-unhighlight, richtext-set-text-color, richtext-set-background-color, richtext-set-text-font, richtext-set-style, display-richtext

richtext-insert [Method]

Defined on Classes:

ui-richtext-form-class

This method allows the user to insert text into the form's widget.

Arguments:

- **text**

The text to insert.

- **position**

  The index where the text will be inserted (0 based indexing inclusive).

Keyword Arguments:

- **font**

  The font of the added text. If nil, the corresponding property is used.

- **background-color**

  The background color of the added text. If nil, the corresponding property is used.

- **foreground-color**

  The foreground color of the added text. If nil, the corresponding property is used.

See Also: richtext-clear, richtext-highlight, richtext-unhighlight, richtext-set-text-color, richtext-set-background-color, richtext-set-text-font, richtext-set-style, display-richtext

richtext-set-background-color [Method]

Defined on Classes:

ui-richtext-form-class

This method allows the user to set the background color of the text that is currently in the form's widget.

Arguments:

- **start-index**

  The index of where the background color will begin to change (0 based indexing inclusive).

- **end-index**

  The index where the background color changing will end (0 based indexing exclusive).

Keyword Arguments:

- **color**

  The color to change the background to. If nil, the corresponding property is used.

See Also: richtext-insert, richtext-clear, richtext-highlight, richtext-unhighlight, richtext-set-text-color, richtext-set-text-font, richtext-set-style, display-richtext

richtext-set-style [Method]

Defined on Classes:

ui-richtext-form-class

This method allows the user to set the style of the text that is currently in the form's widget.

Arguments:

- **start-index**

  The index of where the text style will begin to change (0 based indexing inclusive).

- **end-index**

The index where the text style changing will end (0 based indexing exclusive).

Keyword Arguments:

- **font**

  The font o change the text to. If nil, the corresponding property is used.

- **background-color**

  The color to change the background to. If nil, the corresponding property is used.

- **foreground-color**

  The color to change the foreground to. If nil, the corresponding property is used.

See Also: richtext-insert, richtext-clear, richtext-highlight, richtext-unhighlight, richtext-set-text-color, richtext-set-background-color, richtext-set-text-font, display-richtext

richtext-set-text-color [Method]

Defined on Classes:

ui-richtext-form-class

This method allows the user to set the text color in the form's widget.

Arguments:

- **start-index**

  The index of where the text color will begin to change (0 based indexing inclusive).

- **end-index**

  The index where the text color changing will end (0 based indexing exclusive).

Keyword Arguments:

- **color**

  The color to change the text to. If nil, the corresponding property is used.

See Also: richtext-insert, richtext-clear, richtext-highlight, richtext-unhighlight, richtext-set-background-color, richtext-set-text-font, richtext-set-style, display-richtext

richtext-set-text-font [Method]

Defined on Classes:

ui-richtext-form-class

This method allows the user to set the font of the text that is currently in the form's widget.

Arguments:

- **start-index**

  The index of where the text font will begin to change (0 based indexing inclusive).

- **end-index**

  The index where the text font changing will end (0 based indexing exclusive).

Keyword Arguments:

- **font**

  The font to change the text to. If nil, the corresponding property is used.

See Also: richtext-insert, richtext-clear, richtext-highlight, richtext-unhighlight, richtext-set-text-color, richtext-set-background-color, richtext-set-style, display-richtext

richtext-unhighlight [Method]

Defined on Classes:

ui-richtext-form-class

This method allows the user to unhighlight a region of text that is currently in the form's widget.

Arguments:

- **start-index**

  The index of where the unhighlighting will begin(0 based indexing inclusive).

- **end-index**

  The index where the unhighlighting will end (0 based indexing exclusive).

See Also: richtext-insert, richtext-clear, richtext-highlight, richtext-set-text-color, richtext-set-background-color, richtext-set-text-font, richtext-set-style, display-richtext

display-richtext [Function]

This function displays text.

Keyword Arguments:

- **initial-data**

  Initial data to be displayed.

- **editable?**

  When t, enables the editing of the text.

- **scroll-bars?**

  When t, the scroll-bars will appear when the text goes outside the bounds of the control.

  Default Value: t

- **centered?**

  When t, the widget is centered on the screen.

- **x**

  The horizontal offset (in pixels) of the widget on the screen.

- **y**

  The vertical offset (in pixels) of the widget on the screen.

- **width**

  The width of the widget.

- **height**

The height of the widget.

- **label**

  The label of the widget.

  Default Value: "Richtext"

- **font**

  The font of the text.

  Default Value: "8x13"

- **text-foreground-color**

  The color of the text.

  Default Value: "black"

- **text-background-color**

  The text-background-color.

  Default Value: "white"

- **topmost?**

  When t, richtext widget will stay on top of all other forms.

  Default Value: nil

- **new?**

  When t, the form is deleted and a new object instance is created. The widget is reset with the default values.

See Also: richtext-insert, richtext-clear, richtext-highlight, richtext-unhighlight, richtext-set-text-color, richtext-set-background-color, richtext-set-text-font, richtext-set-style

ui-slider-class [class]

Inheritance: ui-widget, ui-callback, apply-cancel-mixin

Properties:

- **min-value**

  Integer specifying the minimum value

  Default Formula: 0

- **max-value**

  Integer specifying the maximum value

  Default Formula: 10

- **current-value**

  Integer specifying the current value, this property is update when the user interactively changes the slider value

  Default Formula: 0

- **orient**

Can be :vertical or :horizontal.

Default Formula: :horizontal

- **major-tickspace**

  Number of values between major tickmarks

  Default Formula: 5

- **minor-tickspace**

  Number of values between minor tickmarks

  Default Formula: 1

- **show-ticks?**

  Flag to show/hide tickmarks

  Default Formula: t

- **value-change-action**

  Quoted AML expression that is evaluated when the slider value changes.

  Default Formula: nil

Example 16-5. Ui-slider-class Example

```
(define-class test-slider-form
  :inherit-from(ui-form-class)
  :properties(
              x-offset 100
              y-offset 100
              width    500
              height   500
              label "Slider"
              )
  :subobjects(
              (slider1 :class 'ui-slider-class
                       measurement 'pixels
                       x-offset 0
                       y-offset 0
                       width    500
                       height   50
                       min-value 0
                       max-value 100
                       current-value 50
                       value-change-action '(format t "Current Value is:
~a~%" ^current-value)
                       attachment-info-list '(top top left right)
                       )
              )
  )
```

ui-spreadsheet-class [class]

This class provides the user with a spreadsheet-like interface to a table of values. Cells can be editable or read only.

Inheritance: ui-widget, ui-callback.

Properties:

- **column-labels**

  These are the column labels.

  Default Formula: '("Column1" "Column2")

- **row-labels**

  These are the row labels.

  Default Formula: '("Row1" "Row2")

- **cell-values-labels**

  These are the cell values in row-major order.

  Default Formula: '(("00" "01") ("10" "11"))

- **column-width**

  Start width of the cells in the spreadsheet.

  Default Formula: 100

- **row-height**

  Start height of the cells in the spreadsheet.

  Default Formula: 20

- **editable?**

  If t, the cells are editable.

  Default Formula: t

- **button1-action**

  This is the code that is evaluated when a row or column is selected. The "header-selection-index" property will reflect what was clicked on.

  Default Formula: nil

- **default-data-type**

  This property specifies the data type of cells that the user does not provide a value for. The code will use this property to decide what to do with the actual text. Should AML call "read-from-string" on it or not. This may have an effect on how data is written out.

  Default Formula: 'string

Example 16-6. Ui-spreadsheet-class Example

```
(define-class ui-test-spreadsheet-form-class
  :inherit-from(ui-form-class)
```

```
:properties(
            label "Test SpreadSheet"
            width  500
            height 500
            x-offset 20
            y-offset 20
            )
:subobjects(
            (sheet :class 'ui-spreadsheet-class
                   measurement 'pixels
                   x-offset 5
                   y-offset 5
                   width    480
                   height   480
                   column-labels (list "C1" "C2" "C3" "C4")
                   row-labels '("R1" "R2" "R3" "R4")
                   cell-values '(("01" "02")
                                 ("11" "12" "13" "14")
                                 ("21" "22" "23")
                                 )
                   row-height   25
                   column-width 60
                   attachment-info-list '(top bottom left right)
                   editable? nil
                   )
            )
)
```

refresh-cell [Method]

Defined on Classes:

ui-spreadsheet-class

Updates the value in a cell.

Arguments:

- **row**

  The row index of desired cell.

- **column**

  The column index of desired cell.

set-cell-value [Method]

Defined on Classes:

ui-spreadsheet-class

Sets the content of a cell.

Arguments:

- **row**

The row index of desired cell.

- **column**

  The column index of desired cell.

- **value**

  The value of desired cell.

Keyword Arguments:

- **refresh?**

  When t, it automatically refreshes the spreadsheet to reflect the new sheet.

  Default Value: t

get-cell-value [Method]

Defined on Classes:

ui-spreadsheet-class

Returns the current content of a given cell as a string.

Arguments:

- **row**

  The row index of desired cell.

- **column**

  The column index of desired cell.

set-cell-color [Method]

Defined on Classes:

ui-spreadsheet-class

Sets the color of a cell.

Arguments:

- **row**

  The row index of desired cell.

- **column**

  The column index of desired cell.

- **bg**

  The background color string.

- **fg**

  The foreground color string.

Keyword Arguments:

- **refresh?**

When t, it automatically refreshes the spreadsheet to reflect the new sheet.

Default Value: t

get-selected-cells [Method]

Defined on Classes:

ui-spreadsheet-class

Returns a list of pairs where each pair is a list of '(row column) indices of the selected cells.

set-dimensions [Method]

Defined on Classes:

ui-spreadsheet-class

Sets the number of rows and columns.

Arguments:

- **nrows**

  The number of rows.

- **ncolumns**

  The number of columns.

Keyword Arguments:

- **refresh?**

  When t, it automatically refreshes the spreadsheet to reflect the new sheet.

  Default Value: t

select-cells [Method]

Defined on Classes:

ui-spreadsheet-class

Programmatically selects a list of cells.

Arguments:

- **row-column-info-list**

  List of pairs where each pair is a list of '(row column) indices of the cells to select.

Keyword Arguments:

- **refresh?**

  When t, it automatically refreshes the spreadsheet to reflect the new sheet.

  Default Value: t

unselect-cells [Method]

Defined on Classes:

ui-spreadsheet-class

Programmatically unselects a list of cells.

Arguments:

- **row-column-info-list**

  List of pairs where each pair is a list of '(row column) indices of the cells to unselect.

Keyword Arguments:

- **refresh?**

  When t, it automatically refreshes the spreadsheet to reflect the new sheet.

  Default Value: t

ui-typein-field-class [class]

The user has the choice now of defining a validation-function for ui-typein-field-class that takes two arguments. The first argument will be the object instance of the field. The second argument will be the current content of the field (Return value of get-value). For backward compatibility, the validation-function can still take only one argument. In this case, this argument will always be the current content of the field (Return value of get-value).

Inheritance: ui-field-class, ui-callback, apply-cancel-mixin

Properties:

- **Accept-content-action**

  Quoted expression. This expression is evaluated every time the validation-function executes and returns a non-nil value.

  Default Formula: nil

- **Editable?**

  Refer to ui-field-class

  Default Formula: t

- **Validation-function**

  Specifies a function that expects one argument. The argument expected is in the format returned by the method get-value defined on ui-field-class. The validation function specified is triggered when: 1- Input-rows is 1 and the return key is pressed while the cursor is inside the field widget. The return key callback is not supported on WINDOWS platforms. Or 2- Focusout-validation? is non-nil and the field widget loses focus after the user made a change to the text. When the validation function returns a non-nil value, the text in the field is considered valid and the content property is set to the current text. When the validation function returns a nil value, the error message specified in the error-message property appears in a message box.

  Default Formula: nil (disabled)

- **Error-message**

  String specifying the error message to be displayed whenever the validation function returns a nil value.

  Default Formula: "Error"

- **Focusin-action**

Takes a quoted AML expression that will be evaluated everytime the field gains keyboard focus.

Default Formula: nil

- **Focusout-validation?**

  When non-nil, causes the validation-function to be called in case both of the following occur: The user modifies the text in the field then the field widget loses the mouse focus.

  Default Formula: nil

- **button3-action**

  Provides right-click callback functionality to the text field of the class. It expects the same format as the standard button3-action property found in other UI base classes.

  Default Formula: nil

See Also: ui-field-class, get-value, source code sample B, source code sample H

ui-file-text-field-class [class]

It is a text field that allows viewing, editing and saving a text file.

Inheritance: ui-typein-field-class

Properties:

- **File-name**

  Path and name string of file.

There are 3 methods along with this class - display-file-text - provide file name and key, and keyword property force?. Save-to-file - the file name is optional. Select-text - provide the text, &key and two keyword properties - start-position and ignore-case.

ui-toggle-button-class [class]

Ui-toggle-button-class is a ui-widget that provides a check/toggle button that can be toggled on and off by the user. The button is always either in a selected/checked or a released/unchecked status. This type of widgets is referred to as a check button also.

Inheritance: ui-labeled-widget, ui-callback, apply-cancel-mixin

Properties:

- **Release-action**

  Quoted expression that is evaluated when the user releases/unchecks the button with the mouse.

  Default Formula: nil

- **Select-action**

  Quoted expression that is evaluated when the user selects/checks the button with the mouse.

  Default Formula: nil

- **Status**

  Specifies the initial status of the button. When nil, the button appears released/unchecked, otherwise the button appears selected/checked. The value of this property is changed every time the user clicks on the button; therefore, it can be accessed to examine the status of the button at runtime.

Default Formula: nil

This class includes a method get-value, which returns t when the button is selected/checked, nil otherwise. Calling this method on a ui-toggle-button-class instance is equivalent to accessing the property status.

See Also: get-value, ui-radio-buttons-class, ui-toggle-group-class

get-toggle-state [Method]

Defined on Classes:

ui-list-class

Returns t when the button is selected/checked, nil otherwise. Calling this method on a ui-toggle-button-class instance is equivalent to accessing the property status.

ui-toggle-group-class [class]

Ui-toggle-group-class provides a group of toggle buttons (check boxes). It is a non-exclusive group, which means more than one button can be selected/checked at the same time.

Inheritance: ui-widget, ui-callback, apply-cancel-mixin

Properties:

- **Button1-action**

  Quoted expression that is evaluated when a button in the group is selected or released with a left mouse click.

  Default Formula: nil

- **Labels-list**

  List of strings specifying the labels of the toggle buttons. The length and order of this list specifies the number and order of buttons respectively.

- **Status**

  List of flags (t or nil) representing the status of each toggle button. This list should have the same length as the labels-list. A flag of t specifies a selected/checked button, while a flag of nil specifies a released/unchecked button. This list is updated every time the user toggles on/off any of the buttons with a left mouse click.

- **Orient**

  Specifies the orientation used to position the buttons. A :horizontal orientation means that the buttons will be positioned by filling rows going from left to right starting at the topmost row. A :vertical orientation means that the buttons will be positioned by filling columns going from top to bottom starting at the leftmost column.

  Default Formula: :horizontal

- **Wrap-count**

  Wrap-count is interpreted as the number of rows when orient is :horizontal, and as the number of columns when orient is :vertical.

  Default Formula: 1

This class includes a method get-value, which returns a list of flags (t or nil) representing the current status of each toggle button. This list should have the same length as the labels-list. A flag of t specifies a selected/checked button, while a flag of nil specifies a released/unchecked button.

See Also: get-value, ui-radio-buttons-class

ui-unit-conversion-field-class [class]

This class provides a ui-labeled-field-class with a unit selection menu. It is typically used when the application requires input fields with units that are different from the main unit used in the actual AML model. The content property value must be a number. Unit expressions and unit conversions are based on the unit table/functions described in chapter 14 of the AML reference manual.

Inheritance: ui-labeled-field-class

Properties:

- **Label-width**

  Number specifying a percentage of the width of the widget that the label area occupies. (Label-width is inherited from ui-labeled-field-class)

  Default Formula: 30

- **Field-width**

  Number specifying a percentage of the width of the widget that the text field occupies.

  Default Formula: 40

- **Unit-in**

  Valid unit expression specifying the input unit, must be a member of the available-units-list.

- **Unit**

  Valid unit expression specifying the unit of the converted value.

- **Available-units-list**

  A list of unit expressions specifying the available input units.

  Default Formula: (dimensionally-similar-units-list ^unit)

- **Units-labels-list**

  List of strings specifying a label for each available unit. These labels are reflected in the input unit selection menu of the widget.

- **Converted-value**

  This is the output property of the widget. It holds the converted value of the content.

  Default Formula: (convert-units ^content ^unit-in ^unit)

See Also: ui-labeled-field-class, add-unit, convert-units, dimensionally-similar-units-list

converted-value [Method]

Defined on Classes:

ui-list-class

Converts the current contents of the field from unit-in to unit and returns the result. It is recommended to use this method to get the current converted value (v/s accessing the property converted-value) because the property converted-value is not updated unless content is. (Content depends on focusout-validation? and the validation-function ... see ui-typein-field-class)

Action Widgets

This section describes component widgets whose purpose is to trigger a user specified action.

ui-action-button-class [class]

Ui-action-button-class is a ui-widget that provides a visual push button with the capability of a left-click and a right-click callback.

Inheritance: ui-labeled-widget, ui-callback

Properties:

- **Button1-action**

  Quoted expression that is evaluated when the user left-clicks on the button with mouse.

  Default Formula: nil

- **Button3-action**

  Quoted expression that is evaluated when the user right-clicks on the button with the mouse.

  Default Formula: nil

set-accelerator [Method]

Defined on Classes:

ui-action-button-class

This method enables a keyboard shortcut to trigger the left-click callback of a GUI button. The shotcut is defined by a set of keyboard keys referred to as "accelerator" of the button in question. All keys must be down at the same time for the button callback to be triggered.

Arguments:

- **accelerator**

  Default Value: list of keys, keys can be either of the following: "ctrl", "shift", "alt", "f1", "f2", "f3", ... "f12". Any single-character string specifying a key on the keyboard. When accelerator is nil, the button's accelerator is disabled.

Example 16-7. Set-accelerator Example

```
> (set-accelerator (the interface forms form1 button1) '("ctrl" "alt" "d"))

> (set-accelerator (the interface forms form1 button2) '("f5" "e"))

> (set-accelerator (the interface forms form1 button3) nil)
```

ui-apply-button-class [class]

Ui-apply-button-class is a ui-action-button-class whose button1-action is set to call apply-form-method on the form specified by the property apply-form.

Inheritance: ui-action-button-class

Properties:

- **Apply-form**

  Points to a ui-group instance.

  Default Formula: (the superior superior ui-form-class)

See Also: ui-group apply-form-method

ui-cancel-button-class [class]

Ui-cancel-button-class is a ui-action-button-class whose button1-action is set to call cancel-form-method on the form specified by the property cancel-form.

Inheritance: ui-action-button-class

Properties:

- **Cancel-form**

  Points to a ui-group instance.

  Default Formula: (the superior superior ui-form-class)

- **Close?**

  When t, the cancel-form instance is closed (hidden) at the end of the cancel-form-method execution.

  Default Formula: nil

- **Update-form?**

  When t, updates the form pointed to by cancel-form before returning from cancel-form-method.

  Default Formula: nil

See Also: ui-group, cancel-form-method

ui-toolbar-class [class]

This class allows the creation of several adjacently placed buttons referred to as a toolbar. The toolbar buttons can be placed vertically or horizontally. This class is instantiated as a subobject of any ui-group class (form, subform...)

Inheritance: ui-widget, ui-callback

Properties:

- **Availability-list**

  List of t or nil specifying (respectively) the availability of the buttons on the toolbar. When nil is specified for a button, it is disabled (grayed out). When this property is nil, all buttons are available.

  Default Formula: nil

- **Button1-action-list**

List of quoted AML expressions (see button1-action of ui-action-button-class) representing (in order) the left mouse click callbacks of the buttons of the toolbar.

Default Formula: nil

- **Button3-action-list**

  List of quoted AML expressions (see button3-action of ui-action-button-class) representing (respectively) the right mouse click callbacks of the buttons of the toolbar.

  Default Formula: nil

- **Images-list**

  List of strings specifying the full path name of bitmap files to be displayed on the toolbar buttons. An empty string or nil can be specified within the list to specify no image (bmp files are used on WINDOWS and xbm or xpm files are used on UNIX). When this property is nil, no images are used.

  Default Formula: nil

- **Labels-list**

  List of strings specifying the label to be displayed on the toolbar buttons. An empty string or nil can be specified within the list to specify no label. When this property is nil, no labels are used.

  Default Formula: nil

- **Number-of-buttons**

  Integer specifying the number of buttons on the toolbar.

  Default Formula: 1

- **Orient**

  Orientation of the buttons. Can be :horizontal or :vertical.

  Default Formula: :horizontal

- **Tooltips-list**

  List of strings specifying the tooltips of the toolbar buttons. A tooltip is a "help" string that pops up when a button gains the focus of the mouse cursor and the cursor is not moved for a certain time. An empty string or nil can be specified within the list to specify no tooltip. When this property is nil, no tooltips are used.

  Default Formula: nil

set-toolbar-accelerator [Method]

Defined on Classes:

ui-toolbar-class

This method enables a keyboard shortcut to trigger the left-click callback of a button on a GUI toolbar. The shortcut is defined by a set of keyboard keys referred to as "accelerator" of the button in question. All keys must be down at the same time for the button callback to be triggered.

Arguments:

- **button-index**

  Zero-based index of the desired button on the toolbar

- **accelerator**

  List of keys, keys can be either of the following: "ctrl", "shift", "alt", "f1", "f2", "f3", ... "f12". Any single-character string specifying a key on the keyboard. When accelerator is nil, the button's accelerator is disabled.

Example 16-8. Set-toolbar-accelerator Example

```
> (set-toolbar-accelerator (the interface forms form1 toolbar1) 0 '("ctrl"
"alt" "d"))

> (set-toolbar-accelerator (the interface forms form1 toolbar1) 1 '("f5"
"e"))

> (set-toolbar-accelerator (the interface forms form1 toolbar1) 2 nil)
```

ui-work-area-action-button-class [class]

This class enables the AML developer to include an action button in the work-area of an object. This action button will appear when the object is being edited. Contrary to other work area widgets, this class does not need to be associated with a property instance from the object being edited. When the user left-clicks on the work-area button, the method named work-area-button1-action is called on the object being edited. The value of the button1-parameters property is passed as a parameter to the work-area-button1-action method. Similarly, when the user right-clicks on the work-area button, the method named work-area-button3-action is called on the object being edited. The value of the button3-parameters property is passed as a parameter to the work-area-button3-action method.

Inheritance: ui-grid-action-button-class

Properties:

- **button1-parameters**

  The value of this property is passed as a parameter to the work-area-button1-action method. The method work-area-button1-action is called when the button is left-clicked. The format of the button1-parameters value is free and depends on the application.

  Default Formula: nil

- **button3-parameters**

  The value of this property is passed as a parameter to the work-area-button3-action method. The method work-area-button3-action is called when the button is right-clicked. The format of the button3-parameters value is free and depends on the application.

  Default Formula: nil

Example 16-9. Ui-work-area-action-button-class examples

```
;;Example1:
;;Usage with the method property-classification-list

(define-class my-box-class
  :inherit-from(box-object)
```

```
  :properties(
              )
  )

(define-method work-area-button1-action my-box-class (params)
  (case params
    (:draw
     (draw self :draw-subobjects? nil)
     )
    (:undraw
     (undraw self :subobjects? nil)
     )
    )
  )

(define-method work-area-button3-action my-box-class (params)
  (case params
    (:draw
     (draw self :draw-subobjects? t)
     )
    (:undraw
     (undraw self :subobjects? t)
     )
    )
  )

(define-method property-classification-list my-box-class ()
  '(
    ("Geometry" (
                 depth
                 height
                 width
                 ("Draw..." (button1-parameters :draw button3-parameters
:draw)
ui-work-area-action-button-class)
                 ("Undraw..." (button1-parameters :undraw button3-parameters
:undraw)
ui-work-area-action-button-class)
                 ))
    )
  )

;;Example2:
;;Usage with the property property-objects-list

(define-class my-box2-class
  :inherit-from(box-object)
  :properties(
              property-objects-list
              (list
               (the depth self)
               (the height self)
               (the width self)
```

```
               '("Draw..." (button1-parameters :draw button3-parameters
:draw)
ui-work-area-action-button-class)
               '("Undraw..." (button1-parameters :undraw button3-parameters
:undraw)
ui-work-area-action-button-class)
               )
              )
  )

(define-method work-area-button1-action my-box2-class (params)
  (case params
    (:draw
     (draw self :draw-subobjects? nil)
     )
    (:undraw
     (undraw self :subobjects? nil)
     )
    )
  )

(define-method work-area-button3-action my-box2-class (params)
  (case params
    (:draw
     (draw self :draw-subobjects? t)
     )
    (:undraw
     (undraw self :subobjects? t)
     )
    )
  )
```

Tree Widgets

ui-class-tree [class]

Ui-class-tree provides a graphical tree widget representing the AML class hierarchy starting at the class specified by the property root-class. The tree is navigated using the left mouse button by clicking on the arrow of a class node. A class can be selected by left clicking on the class name. **Note :** This class can be directly instantiated even though it breaks the "class" suffix naming convention.

Inheritance: ui-tree-class (ui-tree-class inherits from ui-widget)

Properties:

- **Root-class**

  Class name symbol of the desired root class

- **Selected-class**

  Class name symbol of the currently selected class in the tree widget. The user should not redefine the formula or set the value of this property.

See Also: ui-model-tree, source code sample C

ui-model-tree [class]

Ui-model-tree provides a graphical tree widget representing the AML model hierarchy starting at the object specified by the property root-object. The tree is navigated using the left mouse button by clicking on the arrow of an object node. An object can be selected by left clicking on it. **Note :** This class breaks the "class" suffix naming convention; however, it can be directly instantiated.

Inheritance: ui-tree-class (ui-tree-class inherits from ui-widget)

Properties:

- **Button1-action**

  Quoted expression that is evaluated when an item in the group is selected with a left mouse click.

  Default Formula: nil

- **Button3-action**

  Quoted expression that is evaluated when an item in the group is selected with a right mouse click.

  Default Formula: nil

- **Root-object**

  Object instance of the desired root of the tree widget.

- **Selected-item**

  Object instance of the currently selected object in the tree widget. The user should not redefine the formula or set the value of this property.

- **Object-class**

  Class name symbol of the object class allowed in the tree.

  Default Formula: 'object-root-class

- **Object-test**

  Takes a quoted expression that is evaluated on every object instance. Test should return a non-nil value when called on an object in order for that object to be allowed in the tree widget. The Format of object-test follows the format the test keyword of the select-object function.

  Default Formula: t

See Also: ui-class-tree, source code sample D

ui-tree-class [class]

The ui-tree-class is a widget that enables the organization and visualization of an indefinite number of items in a hierarchical tree form. Items in a tree are identified by unique item-IDs.

Inheritance: ui-widget

Properties:

- **Button1-action:**

  Quoted AML expression that is evaluated when the user left-clicks on a tree item.

Default Formula: nil

- **Button3-action:**

  Quoted AML expression that is evaluated when the user right-clicks on a tree item.

  Default Formula: nil

- **selected-item**

  This property always holds the item ID of the selected item in the tree.

  Default Formula: (tsi::get-selected-item (the superior))

add-tree-item [Method]

Defined on Classes:

ui-tree-class

This method adds a new item to the tree widget.

Arguments:

- **instance**

  Instance of ui-tree-class.

- **itemID**

  ID of new item.

- **parentID**

  ID of item that the new item will be added to. When this parameter is nil, the new item is added as a root item of the tree. More than one item can be added as root items.

- **branch**

  This parameter is ignored.

Keyword Arguments:

- **:font**

  This is an optional argument that allows giving a font specific to the new item. This must be a value returned by the function make-native-font or select-font-dialog on WINDOWS platforms, or a font string on UNIX platforms. If this keyword is not specified (or if it is specified as nil), the font used is the font specified with the ui-tree-class instance.

- **:image**

  This is an optional argument that allows giving a bitmap specific to the new item (the bitmap appears on the left of the item).This is a file path string (bmp on WINDOWS, xbm/xpm on UNIX). If this keyword is not specified (or if it is specified as nil), the default image is assigned to the item depending on its status: Open branch, closed branch or leaf item.

- **:foreground-color**

  This is an optional argument that allows giving a text color specific to the new item. This must be a string (see the AML color names appendix in the AML reference manual). If this keyword is not specified (or if it is specified as nil), the text color used is the one specified with the ui-tree- class instance.

select-tree-item [Method]

Defined on Classes:

ui-tree-class

This method allows to programmatically select/highlight a node in a tree widge. This is equivalent to left-clicking on that node.

Arguments:

- **item**

  Item from tree.

Keyword Arguments:

- **:notify?**

  When t, the left-click callback is triggered after the item is selected. When nil, the item is simply selected/highlighted without triggering any callback.

  Default Value: t

update-tree-item [Method]

Defined on Classes:

ui-tree-class

This method updates the display of an existing item in the tree widget

Arguments:

- **tree**

  Instance of ui-tree-class

- **item**

  Item to be updated (object instance in the case of ui-model-tree).

Keyword Arguments:

- **:label**

  This is an optional argument that allows a label string.

  Default Value: nil

- **:Font**

  This is an optional argument that allows giving a font specific to the new item. This must be a value returned by the function make-native-font or select-font-dialog on WINDOWS platforms, or a font string on UNIX platforms. If this keyword is not specified (or if it is specified as nil), the font used is the font specified with the ui-tree-class instance.

  Default Value: nil

- **:Image**

  This is an optional argument that allows giving a bitmap specific to the new item (the bitmap appears on the left of the item).This is a file path string (bmp on WINDOWS, xbm/xpm on UNIX). If this

keyword is not specified (or if it is specified as nil), the default image is assigned to the item depending on its status: Open branch, closed branch or leaf item.

Default Value: nil

- **:Foreground-color**

  This is an optional argument that allows giving a text color specific to the new item. This must be a string (see the AML color names appendix in the AML reference manual). If this keyword is not specified (or if it is specified as nil), the text color used is the one specified with the ui-tree- class instance.

  Default Value: nil

- **:available?**

  This is an optional argument. When it is nil, it grays out the tree item.

delete-tree-items [Method]

Defined on Classes:

ui-tree-class

This method deletes a list of items from the tree widget.

Arguments:

- **Instance**

  Instance of ui-tree-class.

- **Item-ID-list**

  List of itemIDs to be deleted.

find-tree-item [Method]

Defined on Classes:

ui-tree-class

This function returns non-nil if the item-ID given exists in the tree.

Arguments:

- **ItemID**

  ID of an item.

reset-tree-item [Method]

Defined on Classes:

ui-tree-class

This method undoes any effect on a tree item caused by the optional keywords of add-tree-item or by the method update-tree-item.

Arguments:

- **Instance**

Instance of ui-tree-class

- **ItemID**

  ID of item to reset.

tree-item-children [Method]

Defined on Classes:

aml-class

This method can be redefined by the user on a specific class, it must return a list of object instances. This method defines the order in which subobjects of a certain class must appear in a model tree widget. When a model tree widget (of class ui-model-tree) is being populated, it uses the method tree-item-children in order to get an ordered list of the subobjects at every node in the tree. Tree-item-children calls the method - (children self :expand? nil) by default. If the user redefines it to return nil, no tree item is added under an instance of the class of the redefined method.

tree-item-display [Method]

Defined on Classes:

ui-model-tree

This method is called while an object is being added to the ui-model-tree. It should return a string representing the display of an object on the tree widget. By default, tree-item-display is defined to return the value that the method tree-display returns when called on the object instance in question.

```
(define-method tree-item-display ui-model-tree (object)
        (write-to-string (tree-display object))
        )
```

The user can redefine this method on ui-model-tree or on any user class that inherits from ui-model-tree in order to control the display of objects in his/her model tree widget. When tree-item-display returns nil, the object in question does not show on the tree widget. Note that no dependency is established between the ui-model-tree properties and any properties used in tree-item-display.

Example 16-10. Tree-item-display example

```
;; user-model-tree-class inherits from ui-model-tree
(define-method tree-item-display user-model-tree-class (object)
               <user code for object display control; must return a string
representing object>)
```

See Also: tree-display

open-branch [Method]

Defined on Classes:

ui-tree-class

Programmatically opens the branch of a specific item. Open branch is ignored if the branch specified does not contain children. In other words, open-branch must be called only after child items have been added to the specified parent item or if the item already contains child items.

Arguments:

- **tree**

  Instance of ui-tree-class

- **item**

  Item in question (object instance in the case of ui-model-tree)

Keyword Arguments:

- **all?:**

  optinal keyword, when t, all branches belonging to the subtree of the item specified are opened. When ":all?" is non-nil, the method also open all closed item in the path leading to the desired item. This is similar to the behavior of the method make-item-visible.

  Default Value: nil

close-branch [Method]

Defined on Classes:

ui-tree-class

Programmatically closes the branch of a specific item.

Arguments:

- **tree**

  Instance of ui-tree-class

- **item**

  Item in question (object instance in the case of ui-model-tree)

make-item-visible [Method]

Defined on Classes:

ui-tree-class

Opens the needed branches and moves the vertical scroll bar accordingly in order to make an specific tree item visible.

Arguments:

- **tree**

  Instance of ui-tree-class

- **item**

  Item in question (object instance in the case of ui-model-tree)

Keyword Arguments:

- **scroll-horizontal?:**

Optional keyword. When t, also moves the horizontal scroll bar if needed.

Default Value: nil

highlight-items [Method]

Defined on Classes:

ui-tree-class

This method allows the programmer to mark an item in a tree widget (instance of ui-tree-class) by setting its text color and font.

Arguments:

- **items-list**

  List of items to highlight/mark.

- **foreground-color**

  Color string (see color appendix).

Keyword Arguments:

- **font**

  Font string. The default value is nil, which means the keyword is ignored.

  Default Value: nil

See Also: unhighlight-items, make-item-visible

unhighlight-items [Method]

Defined on Classes:

ui-tree-class

This methods "unmarks" items that were marked using the method highlight-items.

Arguments:

- **items-list**

  List of items.

See Also: highlight-items, make-item-visible

pre-edit-node-method [Method]

Defined on Classes:

aml-class

This method is called when the user selects to "Edit" an object right before the properties of that object are displayed in the work-area. The method does nothing by default and it can be defined by the user on any class to customize the "Edit" behavior in an application.

See Also: post-edit-node-method

post-edit-node-method [Method]

Defined on Classes:

aml-class

This method is called when the user selects to "Edit" an object right after the properties of that object are displayed in the work-area. The method does nothing by default and it can be defined by the user on any class to customize the "Edit" behavior in an application.

See Also: pre-edit-node-method

Menus

Classes

Menus are GUI entities that provide standard menu-bar, pull-down and cascading menu functionality for the interface. They are typically activated by a left mouse click.

ui-menubar-class [class]

Ui-menubar-class provides a widget responsible of managing a set of pull-down menus. It is instantiated as a parent object to a set of ui-pulldown-class objects to provide standard menu bar functionality to the AML application. Pull-down menus can be arranged either vertically or horizontally. A ui-menubar-class instance is always a subobject of a top-level form.

Inheritance: ui-widget

Properties:

- **Orient**

  Can also be :vertical. Under Windows NT platforms a menu-bar is always horizontal because of OS standards.

  Default Formula: :horizontal

Note : 1. Windows OS always places a menu-bar in the top left corner of a window with its width equal to the window width. This OS constraint ignores any x-offset, y-offset, width and height values that are given to a menu-bar under Windows, while these properties are required under Unix platforms in order to position the menu bar. It is recommended, for portability issues, to always position a menu bar following the Windows standard, except if the code is not intended to run on Windows platforms at all. 2. WINDOWS does not consider a menubar a standard component that occupies an area of a form, but rather treats it as a feature of the form that is inserted between the title bar and the component's area. Therefore, a child widget of that form with a y-offset of 0 will be positined right below the menubar. On the other hand, a menubar on a UNIX platform is treated as a regular component of the form, which means that if a menubar is positioned at the topmost of the form, a child widget of that form with a y-offset of 0 will overlap the menubar. **Example:** See example for ui-pulldown-class

ui-menu-item-class [class]

Ui-menu-item-class is only instantiated as a subobject of a ui-pulldown-class instance. It created the actual item in the pulldown menu from which an action is triggered. A menu item is activated (i.e. its action is triggered) by either a left mouse click, accelerator keys, or a mnemonic.

Inheritance: ui-gadget

Properties:

- **Label**

  String that specifies the label displayed on the menu item.

  Default Formula: (write-to-string (object-name (the superior)))

- **Image**

  Specifies a bitmap file path string for the to be displayed on the menu item. Under UNIX systems, the bitmap should be in xbm or xpm format. Under Windows systems, the bitmap should be in bmp format. When a valid bitmap is specified, the label property is disregarded. Until AML3.2, this property is disabled; it will be enabled in future AML releases.

  Default Formula: ""

- **Button1-action**

  Quoted expression that is evaluated when the item is activated with a left mouse click.

  Default Formula: ""

- **Accelerator**

  Shortcut keys combination used to activate the menu item without having to open its parent pulldown menu at all. It should be a list of the form `(:control-key :alt-key #\<character>). The :control-key or :alt-key or both can be omitted but cannot be switched. Some accelerator examples : `(:control-key #\A) `(:alt-key #\D) `(#\C) `(:control-key :alt-key #\F)

  Default Formula: nil

- **Accel-text**

  String to be displayed next to the menu item label used to show the user what accelerator keys are defined on that button. This property is ignored on WINDOWS platforms; a default accelerator text is displayed whenever a valid accelerator is specified.

  Default Formula: ""

- **Mnemonic**

  Letter symbol used to activate the menu item when its parent pulldown menu is open. It is typically one of the letters of the menu item label.

  Default Formula: nil

- **toggle?**

  When this property is t, the menu item behaves like a flag with a check mark that becomes visible/invisible when the item is toggled on/off.

  Default Formula: nil

- **init-status**

  This property specifies the initial status of a menu item. When t, the menu item is checked on. This property is ignored when "toggle?" is nil.

  Default Formula: nil

Example: See example for ui-pulldown-class

ui-pulldown-class [class]

Ui-pulldown-class provides pull-down menus and cascading menus functionality. A ui-pulldown-class is instantiated as a subobject of a ui-menubar-class or of another ui-pulldown-class object (to provide sub-menus or cascading menus). It is used as a superior object to a set of ui-menu-item-class and other ui-pulldown-class instances. It is created as a "button" on its parent menu-bar or as an item within its parent pull-down menu. It opens a menu when activated by a left mouse click. Pull-down menus are placed on a menu-bar starting from the left or top in the order they were created depending on whether the orient property of their superior ui-menubar-class is :horizontal or :vertical.

Inheritance: ui-root

Properties:

- **label**

  String specifying the title of the pulldown menu.

  Default Formula: (write-to-string (object-name (the superior)))

- **image**

  File path string of a bitmap image file to be displayed on the menu. An XBM or XPM format is required on UNIX platforms. A BMP format is required on WINDOWS platforms. When a valid image file is specified, the label is ignored.

  Default Formula: ""

- **label-align**

  Keyword specified the text justification of the menu label. It can also take the values of :left or :right.

  Default Formula: :center

- **gray?**

  When t, the pulldown menu is disabled and appears grayed out.

  Default Formula: nil

- **font**

  Takes a string specifying the X font used for the pulldown menu text. This property is effective on Unix Platforms only.

  Default Formula: (default)

- **background-color**

  Specifies the background color of the pulldown menu. This property is effective on Unix platforms only. The colors of a menu follow their menubar colors on WINDOWS.

  Default Formula: (default)

- **foreground-color**

  Specifies the text color of the pulldown menu. This property is effective on Unix platforms only. The colors of a menu follow their menubar colors on WINDOWS.

  Default Formula: (default)

- **help?**

When t, places the pulldown menu on the right/bottom end of the menubar. Every menubar can only have only one help pulldown menu. This property is ignored on Windows platforms.

Default Formula: nil

- **separator-index-list**

  When non-nil, it is a list of integer indices specifying the position of separators on the pulldown menu.

  Default Formula: nil

Example 16-11. Ui-pulldown-class example

```
(in-package :aml)

(define-class display-control-menu
  :inherit-from(ui-pulldown-class)
  :properties(label "Display"
              )
  :subobjects(
              (draw-item :class `ui-menu-item-class
                         label "Draw/Undraw"
                         button1-action `(if (displayed? (root-object))
(undraw (root-object) (draw (root-object))))
                         accelerator `(:control-key #\D)
                         mnemonic `D)
                         (shade-item :class `ui-menu-item-class label
"Shade/Unshade"
                                     button1-action `(shade-unshade-function
(root-object))
                                     accelerator `(:control-key #\S)
                                     mnemonic `S)
                         )
  )

(define-class models-menu
  :inherit-from(ui-pulldown-class)
  :properties(label "Model")
  :subobjects(
              (new-model :class `ui-menu-item-class
                         label "New"
                         button1-action `(create-model `model-class)
                         accelerator
                         mnemonic `N)
              (select-model :class `ui-menu-item-class
                            label "Select"
                            button1-action `(select-model-function)
                            accelerator mnemonic `S)
              ;; sub-menu
              (draw :class `display-control-menu)
              )
  )

(define-class test-menubar
```

```
:inherit-from(ui-menubar-class)
:properties(
            x-offset 0 y-offset 0
            width 100 height 100
            measurement `percentage
            )
:subobjects(
            (menu1 :class `models-menu)
            ;; A menubar can have several other menus
            )
)
```

Composite Widgets

Classes

A composite widget is a combination of several widget classes into one class. A composite widget is treated as a single widget and its constituent widgets are invisible to the user. Below is the list of existing composite widgets starting by their super-class ui-composite-widget. A user can define his/her own composite widget by inheriting from the ui-composite-widget; however, this practice is not recommended for non-experienced AML developers.

ui-composite-widget [class]

This is the super-class of all composite widgets. It should not be instantiated. It can be inherited into when creating a user-defined composite widget.

Inheritance: ui-root, apply-cancel-mixin

Properties:

- **Gray?, font, light-slope-color, dark-slope-color, slope-thickness, selected-color, background-color, foreground-color, measurement, x-offset, y-offset, width, height, label, label-align, image**

  Refer to the description of ui-widget, and ui-labeled-widget

- **Widgets**

  This property holds a list of the widget instances that constitute the composite widget. Unless the user is defining a new composite widget, the formula of this property should not be redefined. These widgets are typically properties/subobjects of the composite widget. Note that a composite widget does not have its own WINDOWS/Motif entity, it is only an AML manager/place-holder for its constituent widgets.

The following methods apply to this class : **display :** Calls display on the members of the widgets list. **hide :** Calls hide on the members of the widgets list. **update :** Updates (Calls the method update) the members of the widgets list when needed.

See Also: ui-widget, ui-labeled-widget

ui-action-field-pair-class [class]

This is a composite widget constituted of an action button horizontally adjacent to a field widget (ui-field-class).The action button is always on the left side. The label string of the widget appears on the action button.

Inheritance: ui-composite-widget

Properties:

- **Accept-content-action**

  Refer to ui-typein-field-class.

- **Content, input-rows, scroll-bars?**

  Refer to ui-field-class.

- **Button1-action, button3-action**

  Refer to ui-action-button-class.

- **Editable?**

  When non-nil, the text in the field can be edited.

  Default Formula: nil

- **Label-width**

  Number specifying a percentage of the total width of the widget that the action button occupies.

  Default Formula: 50

- **Field-background-color**

  String specifying the background color of the field widget.

  Default Formula: ^background-color

- **Field-foreground-color**

  String specifying the text color of the field widget

  Default Formula: ^foreground-color

The following methods apply to this class : **add-text :** Refer to description of ui-field-class **field-set-editable :** Refer to description of ui-field-class **get-scroll-bar-position :** Refer to description of ui-field-class **get-value :** Refer to description of ui-field-class **get-window-coordinates :** Refer to description of ui-widget **replace-text :** Refer to description of ui-field-class **set-scroll-bar :** Refer to description of ui-field-class

See Also: ui-typein-field-class, ui-field-class, ui-action-button-class

ui-labeled-option-menu-class [class]

This is a composite widget constituted of a label widget horizontally adjacent to an option menu widget. The label is always on the left side.

Inheritance: ui-composite-widget

Properties:

- **Button1-action, options-list, labels-list, selected-option**

Refer to the description of ui-option-menu-class

- **Label-align**

  Alignment of the label text. Refer to the description of ui-labeled-widget.

  Default Formula: :left

- **Label-width**

  Number specifying a percentage of the width of the composite widget that the label area occupies

  Default Formula: 50

- **Label-background-color**

  String specifying the background color of the label area.

  Default Formula: ^background-color

- **Label-foreground-color**

  String specifying the text color of the label.

  Default Formula: ^foreground-color

The following methods apply to this class : **get-value :** Refer to description of ui-option-menu-class. **get-window-coordinates :** Refer to description of ui-widget. **select-option :** Refer to description of ui-option-menu-class.

See Also: ui-option-menu-class, ui-labeled-widget

ui-labeled-radio-buttons-class [class]

This is a composite widget constituted of a label widget horizontally adjacent to a radio-buttons widget. The label is always on the left side.

Inheritance: ui-composite-widget

Properties:

- **Orient, wrap-count, labels-list, status, button1-action**

  Refer to the description of ui-radio-buttons-class.

- **Label-align**

  Alignment of the label text. Refer to the description of ui-labeled-widget.

  Default Formula: :left

- **Label-width**

  Number specifying a percentage of the width of the composite widget that the label area occupies.

  Default Formula: 50

- **Label-background-color**

  String specifying the background color of the label area.

  Default Formula: ^background-color

- **Label-foreground-color**

  String specifying the text color of the label.

Default Formula: ^foreground-color

See Also: ui-radio-buttons-class, ui-labeled-widget

Grid Form and Components

A widget group (i.e a form, a subform...) can be partitioned into a virtual grid to allow easy positioning and sizing of component widgets. The user defines a unit-height and a unit-width value on a grid form. The grid component subobjects of that form are positioned using their coordinates property. The coordinates '(x y) denote an x-offset position of x unit-width(s) and a y-offset position of y unit-height. A grid component is sized based on its size property. The size property is a '(w h) pair where w denotes a width equal to w times unit-width and h denotes a height equal to h times unit-height.

The classes ui-grid-form-mixin and ui-grid-component-mixin are described below. Typically a user form/subform class should be defined to inherit from both ui-grid-form-mixin and ui-form-class/ui-subform-class to provide itself with grid capabilities. Similarly a grid component widget should be defined to inherit from both a ui-grid-component-mixin and the desired widget class in order to add grid component capability to that widget class. Several grid component widgets are already defined and are listed below. Note that a ui-group class can also be a grid component when used as a subobject of another ui-group.

The unit-height and unit-width properties are closely tied to the measurement property. In other words, a unit-height of 5 with a measurement property set to 'percentage means 5 percent of the height of the grid form, while a unit-height of 5 and a measurement property set to 'pixels means 5 pixels. The measurement property value of sibling grid component and of their superior grid form should be equal.

When using grid forms and grid components, the AML programmer does not have to deal with any of the x-offset, y-offset, width, and height properties of the grid components (since they are all derived); however, he/she can always overwrite them.

Classes

ui-grid-form-mixin [class]

Class to be mixed with a ui-group class to provide grid capabilities to the group.

Inheritance: Object

Properties:

- **Unit-height**

  Value of 1 vertical unit of the grid. This value is a pixel value if the measurement property is equal to 'pixels and is a percentage of the height of the form if the measurement property is equal to 'percentage.

  Default Formula: 100

- **Unit-width**

  Value (in percentage or pixels) of 1 horizontal unit of the grid. This value is a pixel value if the measurement property is equal to 'pixels and is a percentage of the width of the form if the measurement property is equal to 'percentage.

  Default Formula: 100

See Also: source code sample E and F

ui-grid-component-mixin [class]

Class to be mixed with a component widget and instantiated as a subobject of a ui-grid-form-mixin in order to provide the widget with grid capabilities. (See Above)

Inheritance: object

Properties:

- **Coordinates**

  Specifies the position of the widget as a factor of the unit-heigh t and unit-width (respectively) properties of the superior grid form.

  Default Formula: '(0 0)

- **Size**

  Specifies the size (height, width) of the widget as a factor of the unit-height and unit-width (respectively) properties of the superior grid form.

  Default Formula: '(1 1)

See Also: source code sample E and F, ui-grid-action-button-class

Example 16-12. Ui-grid-component-mixin example

Example of a grid form with action buttons grid components. Refer to the description of ui-grid-action-button-class later.

```
(in-package :aml)

(define-class grid-widgets-form
  :inherit-from(ui-grid-form-mixin ui-form-class)
  :properties(
              label "Row Column Form"
              x-offset 10
              y-offset 10
              width 200
              height 500
              rows 10
              columns 4
              unit-height (/ 100.0 ^rows)
              unit-width (/ 100.0 ^columns)
              )
  :subobjects(
              (b1 :class 'ui-grid-action-button-class
                  coordinates '(0 0)
                  )
              (b2 :class 'ui-grid-action-button-class
                  coordinates '(0 1)
                  )
              (b3 :class 'ui-grid-action-button-class
                  coordinates '(1 0)
                  )
              (b4 :class 'ui-grid-action-button-class
```

```
                    coordinates '(1 1)
                    )
                (b5 :class 'ui-grid-action-button-class
                    coordinates '(2 0)
                    size '(1 2)
                    )
                (b6 :class 'ui-grid-action-button-class
                    coordinates '(2 2)
                    size '(1 2)
                    )
                (b7 :class 'ui-grid-action-button-class
                    coordinates '(3 0)
                    size '(1 1.5)
                    )
                (b8 :class 'ui-grid-action-button-class
                    coordinates '(3 1.5)
                    size '(1 1.5)
                    )
                )
  )
```

ui-grid-action-button-class [class]

Action button with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-action-button-class

ui-grid-apply-button-class [class]

Apply button with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-apply-button-class

ui-grid-cancel-button-class [class]

Cancel button with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-cancel-button-class

ui-grid-class-tree [class]

Class tree widget with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-class-tree

ui-grid-combo-box-class [class]

Combo box with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-combo-box-class

ui-grid-field-class [class]

Field widget with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-field-class

ui-grid-label-class [class]

Label widget with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-label-class

ui-grid-labeled-field-class [class]

Labeled field widget with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-labeled-field-class

ui-grid-labeled-option-menu-class [class]

Labeled option menu with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-labeled-option-menu-class

ui-grid-list-class [class]

List widget with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-list-class

ui-grid-message-field-class [class]

Message field widget with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-message-field-class

ui-grid-model-tree [class]

Model tree widget with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-model-tree

ui-grid-option-menu-class [class]

Option menu widget with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-option-menu-class

ui-grid-radio-buttons-class [class]

Inheritance: ui-grid-component-mixin, ui-radio-buttons-class

ui-grid-toggle-button-class [class]

Toggle button widget with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-toggle-button-class

ui-grid-toggle-group-class [class]

Toggle group widget with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-toggle-group-class

ui-grid-typein-field-class [class]

Ui-typein field-class widget with grid component capabilities.

Inheritance: ui-grid-component-mixin, ui-typein-field-class

Utility Functions

Functions

This section describes utility functions to be used in developing AML graphical user interfaces. Most functions described can always be obtained by building AML objects from ui-root classes; however these function should be used anytime the existence of an object is not necessary.

Note : The functions arguments below that appear between brackets are optional arguments or keywords. Optional arguments and keywords will typically appear with a default value specified after them.

add-application-menu [Function]

This function adds an instance of a user pulldown-down menu class to the "Applications" menu of AML's main menubar. If the "Applications" menu does not exist, it is first added. This function will typically be used to provide startup menus for loaded modules/systems. The user is expected to define a menu class (inheriting from ui-pulldown-class) for his/her module/system and then add it to the AML menubar using add-application-menu. It is recommended to start using this function instead of add-header-menu as it allows more than one menu to be all confined within the "Applications" menu.

Arguments:

- **menu-name**

  Name of menu object to add (symbol).

- **menu-class-name**

  Class name (symbol) of the menu to add, must inherit from ui-pulldown-class.

Keyword Arguments:

- **label**

  Label of menu (string).

- **display?**

  When t, the AML menubar is displayed automatically in case it was hidden.

  Default Value: t

Example 16-13. Add-application-menu Example

```
(add-application-menu 'my-app-menu 'my-app-menu-class :label "My App")
;; returns : Checking out license AML_FEATURE_2_1_127_2
```

add-form [Function]

This function behaves exactly like display-form. The only difference is that it will not display the form added.

Arguments:

- **Name**

  The name of the form.

Keyword Arguments:

- **:main-branch**

  Object instance pointing to the branch of the tree 1 level under model-manager.

  Default Value: (the interface)

- **:branch-name**

  Object name of the branch under main-branch where the form will be searched for or added. If no object of name form-name exists under main-branch , an object of class ui-layout-class and of name branch-name is created.

  Default Value: 'forms

- **:class**

  Class name of the form.

  Default Value: The form-name

- **:x**

  The x-offset of the form. It can now accecpt the value :cursor, which when used will display the form at an x position that is equal to the current mouse cursor x position.

  Default Value: The x-offset property formula of the form class.

- **:y**

  The y-offset of the form. It can now accecpt the value :cursor, which when used will display the form at a y position that is equal to the current mouse cursor y position.

  Default Value: The y-offset property formula of the form class.

- **:width**

  The width of the form.

  Default Value: The width property formula of class.

- **:height**

  The height of the form.

  Default Value: The height property formula of class.

- **:centered?**

  When this keyword is t, the x and y keywords are ignored and the form is displayed in the center of the screen.

  Default Value: nil

- **:init-form**

  The list of property names and values to give to the form.

- **:new?**

  If a form with name form-name already exists under branch-name, and new? is set to t, the form is deleted and a new object instance is created.

add-header-menu [Function]

This function is used to add a pull-down menu (instance of ui-pulldown-class) to the AML menu bar. If a pull-down menu of the specified name already exists, the function call is ignored. This function is typically called on application startup from the aml-init-function.

Arguments:

- **menu-name**

  The object-name of the button to be added.

Keyword Arguments:

- **:class**

  The class name of the pulldown menu being added. It can be any class that inherits from ui-pulldown-class.

  Default Value: ui-pulldown-class

- **:label**

  The string label of the menu.

  Default Value: (write-to-string menu-name)

- **:menu-items**

  List of entries of the form : (label-string quoted-action)

  Default Value: nil

- **menu-item-class**

  Can be of any class that inherits from ui-menu-item-class.

  Default Value: ui-menu-item-class

Note: For every entry in menu-items, an object of class menu-item-class is added to the existing menu items of the pull-down menu class. Initially, the default :class has no menu items under it.

Example 16-14. Add-header-menu example

```
(define-class example-pulldown-menu
        :inherit-from(ui-pulldown-class)
        :properties(label "Menu2")
        :subobjects(
                (item21 :class 'ui-menu-item-class
                        label "Item21"
                        button1-action '(print 'item21-selected)
                        )
```

```
                )
        )
(add-header-menu 'added-menu1 :label "Menu1" :menu-items'(("Item11" '(print
'item11-selected)) ("Item12" '(print 'item12-selected)))
(add-header-menu 'added-menu2 :label "Menu2" :class 'example-pulldown-menu)
```

aml-assist-function [Function]

This function opens a pop-up menu that allows access to several "assist" functionality options that can generate formulas/values for AML properties. The function returns the generated formula or value. This functionality is used by the Assist button of the inspection form and the formula editor form.

Optional Arguments:

- **trace-from**

  When specified it must be an object or property instance. Formulas generated by the function will trace from this instance (When applicable).

The default assist functionality options that AML provides are listed below: (The assist-functionality-ID corresponding to each option is also listed. Its use is described in the "Customized behavior" section below)

1. **"Get Object" - assist-get-object**

   Allows the user to select an object or a property and returns the corresponding "the" reference statement. The "the" reference statement returned traces from the trace-from instance provided. If no trace-from is provided, tracing starts from the model-manager. The object selection mechanism set from the AML options is used for object selection. (Refer to the documentation of get-object for the different object selection mechanisms)

2. **"Get Object List" - assist-get-object-list**

   Similar to "Get Object" but allows the user to keep select more than one object and returns a list of "the" reference statements. If the object selection mechanism is graphical, the user must right click to abort selection. If the object selection mechanism uses a tree menu, the user should left-click on all the desired objects than hit Ok. (Refer to the documentation of get-object for the different object selection mechanisms)

3. **"Get Object List of Class" - assist-get-object-list-of-class**

   Similar to "Get Object List", but also prompts for a class name to filter object selection to instance of the specified class.

4. **"Get Coordinate Value" - assist-get-coordinate-value**

   Allows selection of a point and returns its coordinate value list. Uses the select-a-point-method function. This option is used if no dependency on the selected point is needed.

5. **"Get Coordinate Formula" - assist-get-coordinate-formula**

   Allows selection of a point and returns a formula that can be used to generate the coordinate value list. Uses the select-a-point-method function. This option is used if a dependency (reflected in the formula returned) on the selected point is needed.

6. **"Get Coordinate Value List" - assist-get-coordinate-value-list**

Uses the select-points-method to select a list of points and returns a list of coordinate value list of the points selected. This option is used if no dependency on the selected points is needed.

7. **"Get Coordinate Formula List" - assist-get-coordinate-formula-list**

   Uses the select-points-method to select a list of points and returns a list of formulas that can be used to generate the coordinate value lists of the selected points. This option is used if a dependency (reflected in the formulas returned) on the selected points is needed.

8. **"Get Distance Value Between Points" - assist-get-distance-value**

   Allows the selection of two points (using the select-a-point-method function) and returns the distance between the two points selected.

9. **"Get Distance Formula Between Points" - assist-get-distance-formula**

   Allows the selection of two points (using the select-a-point-method function) and returns a formula that can be used to generate the distance between the two points selected.

10. **"Get File" - assist-get-file**

    Opens the file selection dialog an returns the file selected (path string) or nil when no file is selected.

All formulas generated by the options above start their "the" reference from the trace-from instance provided. If no trace-from instance is provided, tracing starts at the model-manager. **Customized Behavior:** The AML user can customize the aml-assist-function by setting the available "assist" options that show up on the pop-up-menu. The method aml-assist-option-list should be used: Aml-assist-option-list is a method that can be defined by the AML developer on a certain class to determine the "assist" functionality that will be available to the user whenever aml-assist-function is called with its trace-from parameter being an instance of that class. The effect of the customization is global throughout the AML session including non user-defined buttons that call the aml-assist-function with a trace-from argument. The aml-assist-option-list method should return a list of couples with each couple containing as follows: '(label-string assist-functionality-ID) 1. The label-string is the label that will appear on the options pop-up menu when the user calls aml-assist-function. 2. The assist-functionality-ID defines what assist function/method will be triggered once its corresponding label is selected from the pop-up menu.(The available assist-functionality-ID that AML provides are listed above). The user can also return a user defined assist-functionality-ID. A user defined ID must be the name of a function that takes an instance of the property/object in question as argument. It can also be the name of a method defined on that class. A user-defined assist function/method should return the desired value/formula. In other words, the user can define his/her own function that takes a property-object instance argument (or a method defined on the desired class) and returns a formula/value. As long as the function/method name is specified as an assist-functionality-ID, it will be available on the pop-up-menu when aml-assist-function is called with the trace-from argument being an instance of that class.

Example 16-15. Aml-assist-function Example

```
(define-method aml-assist-option-list my-property-object ( )
'(
        ("Get Object" assist-get-object)   ;; assist-get-object is an ID
provided by AML
        ("Get Object List" assist-get-object-list) ;; assist-get-object-list
is an ID provided by AML
```

```
        ("Get Sphere Diameter" assist-get-sphere-diameter) ;; assist-get-
sphere-diameter is a user defined ID, which is a method name (See next)
))

(define-method assist-get-sphere-diameter my-property-object ()
        (let ((obj (get-object :class 'sphere-object))
                )
        (when obj `(the diameter (:from ,(find-the-trace obj :from self))))
        )
)
```

Notes: 1. If aml-assist-option-list returns nil, the default assist functionality options will be used. 2. When the user left-clicks the Assist Button of the inspection form and the formula editor form, the function aml-assist-function is called on the property-object instance being modified. Therefore, a user can customize the behavior of this button by using the aml-assist-option-list.

aml-save-dialog [Function]

This function is the GUI version of the method aml-save. It opens a dialog that allows setting the parameters and options available with the method aml-save. The user can then save the AML model/object by clicking on the "Save" button. When the "Save Loaded Objects?" toggle button is on, the model/object is saved with the "save-loaded-objects?" keyword set to true.

Keyword Arguments:

- **:root-object?**

  When t, it indicates that the user intends to save the current AML model. When nil, it indicates that the user intends to save a certain object from the current model. The user can change his/her mind when the dialog is displayed as there is a check box that allows to reset this value.

  Default Value: t

- **:object**

  This is the object instance that the user intends to save. This is effective only when the ":root-object?"keyword is nil.

  Default Value: nil

- **:filter**

  This specifies the file extension used for AML object/model files. It must be a string of the form "*.abc", where "abc" is the file extension. On WINDOWS platforms, the user can specify more than filter by separating them by a semi-colon, for example: "*.mdl;*.obj"

  Default Value: "*.mdl"

- **:model-version**

  This indicates the version of the object/model to save.

  Default Value: "1.0"

- **:description**

  This is a description string that will be saved in the object/model file.

Default Value: ""

- **:x**

  This is an integer specifying the x-offset of the dialog (in pixels) from the left of the screen. When nil, it follows the current mouse cursor position.

  Default Value: nil

- **:y**

  This is an integer specifying the y-offset of the dialog (in pixels) from the top of the screen. When nil, it follows the current mouse cursor position.

  Default Value: nil

Example 16-16. Aml-save-dialog Example

```
(aml-save-dialog
:root-object? nil
:object (the box1)
:model-version "1.0"
:description "This is a beta version of box 1"
)
```

click-message [Function]

This function displays five lines of information in the active message pane (see message) which, by default, describes mouse click information.

Optional Arguments:

- **Heading**

  String value to be displayed on the first line of the message pane.

  Default Value: "select"

- **Left**

  String value to be displayed on the second line of the message pane.

  Default Value: "Mouse Left: Select"

- **Middle**

  String value to be displayed on the third line of the message pane.

  Default Value: nil

- **Right**

  String value to be displayed on the fourth line of the message pane.

  Default Value: nil

- **Esc**

  String value to be displayed on the fifth line of the message pane.

  Default Value: "Esc: Finish Selection.".

Example:

```
(click-message "Heading" "Left" "Middle" "Right" "ESC")
```

display-canvas-form [Function]

This function creates and displays an instance of ui-canvas-form-class. The form is given the specified name and instantiated under (the interface forms). If an instance with the specified name already exists under (the interface forms), that instance is just displayed. A ui-canvas-form-class is a top-level form with a graphics canvas that occupies its whole area. Calling the method activate-display on the form activates the canvas.

Keyword Arguments:

- **:name**

  A name for the form

- **:x**

  The x coordinates for the starting point of the form.

- **:y**

  The y coordinates for the starting point of the form

- **:height**

  The height of the form.

- **:width**

  The width of the form.

- **:new?**

  When new? is t, the form is deleted and created again in case it exists.

display-dependency-tree-form [Function]

This function display a form showing the dependency tree of a given property object instance. It can be used in 2 modes: Showing the tree of objects/properties the given property depends on, or showing the tree of objects/properties the given property affects.

Arguments:

- **source-property-object**

  An instance of property-object.

Keyword Arguments:

- **i-affect?**

  When t, the "I affect" tree is shown instead of the "I depend on" tree.

  Default Value: nil

- **dependency-level**

  Level of dependency to follow. More level can be expanded once the tree is displayed by left-clicking on a tree node.

Default Value: 1

- **width**

  Width of the form in pixels.

  Default Value: 310

- **height**

  Height of the form in pixels.

  Default Value: 600

- **x**

  X position (from left) of the form in pixels. When nil, form is displayed at current mouse position.

  Default Value: nil

- **y**

  Y position (from top) of the form in pixels. When nil, form is displayed at current mouse position.

  Default Value: nil

- **object-image-method**

  Name of a method to be used to generate the icon of an object/property instance in the tree. This method must return a valid image file path string (bmp on windows, xpm on inix). This method can be defined on any object/property class and must not take any arguments. When nil, the default images are used.

  Default Value: nil

- **tree-widget-object**

  Instance of ui-model-tree used in the current application as the default model tree. This is used when the dependency tree needs to interact with the model tree.

  Default Value: nil

- **topmost?**

  When t, the dependency tree form will stay on top of all other forms.

  Default Value: nil

display-form [Function]

Controls the display and creation of top-level form objects.

Arguments:

- **Name**

  The name of the form.

Keyword Arguments:

- **:main-branch**

  Object instance pointing to the branch of the tree 1 level under model-manager.

  Default Value: (the interface)

- **:branch-name**

  Object name of the branch under main-branch where the form will be searched for or added. If no object of name form-name exists under main-branch , an object of class ui-layout-class and of name branch-name is created.

  Default Value: 'forms

- **:class**

  Class name of the form.

  Default Value: The form-name

- **:x**

  The x-offset of the form. It can now accecpt the value :cursor, which when used will display the form at an x position that is equal to the current mouse cursor x position.

  Default Value: The x-offset property formula of the form class.

- **:y**

  The y-offset of the form. It can now accecpt the value :cursor, which when used will display the form at a y position that is equal to the current mouse cursor y position.

  Default Value: The y-offset property formula of the form class.

- **:width**

  The width of the form.

  Default Value: The width property formula of class.

- **:height**

  The height of the form.

  Default Value: The height property formula of class.

- **:centered?**

  When this keyword is t, the x and y keywords are ignored and the form is displayed in the center of the screen.

  Default Value: nil

- **:init-form**

  The list of property names and values to give to the form.

- **:new?**

  If a form with name form-name already exists under branch-name, and new? is set to t, the form is deleted and a new object instance is created.

Description : If a form named form-name under branch-name exists, the form is displayed, and the properties specified in the function keywords are updated. If form-name does not exist, an object of name form-name and of class class is created. This object is added in the tree under branch-name with its property values as specified by either the function keywords or the class definition. **Note :** Supplying a property to the init-form keyword causes a change-formula on it, i.e. the property value will be smashed.

Example 16-17. Display-form example.

```
;;; Example 1
(display-form 'form1 :class 'my-form-class :centered? t)
;;; Example 2
(display-form 'form2 :class 'my-form-class :x :cursor :y 100)
;;; Example 3
(display-form 'form3 :class 'my-form-cladd :x :cursor :y :cursor)
```

display-text [Function]

This function opens a form and displays user text or the contents of a text file. The text can be editable. The function returns the instance of the text editor form. The method get-value can be called on that instance to return the content of the text area as a string.

Keyword Arguments:

- **:text**
- **:file**
- **:title**
- **:x**
- **:y**
- **:width**
- **:height**
- **:font**
- **:foreground**
- **:background**
- **:editable?**
- **:topmost?**

  When non-nil, the text/file form becomes a topmost form (it will remain on top of all other forms).

  Default Value: nil

- **:file-filter**

  This keyword specifies the default file name extension filter (a string) used when the user decides to open or save a file from the "File" menu of the text editor.

  Default Value: "*.*"

get-message [Function]

This function returns the string currently displayed in the active message pane.

See Also: message

get-message-pane [Function]

This function returns the current AML message pane (instance of ui-message-field-class). It does not take any arguments.

See Also: set-message-pane

get-object [Function]

This function is used to allow the user to interactively select an object from a tree menu or from the graphics display window. Selecting an object is achieved with a left mouse click. A right mouse click aborts the function and returns nil.

Keyword Arguments:

- **:from**

  Specifies the root node for the branch of the tree to be displayed.

  Default Value: the current model

- **:class**

  Class name: Only the objects that inherit from this class will be selectable. This keyword also accepts a list of classes: Objects that inherit from any of these classes will be selectable.

  Default Value: all AML objects

- **:test**

  Quoted AML expression to be evaluated on each object. Only the objects that pass this test will be selectable.

  Default Value: all AML objest will pass the test

- **:prompt**

  String specifying the user message to be shown on the message pane.

- **:mechanism**

  If mechanism is not supplied, the system will look for the get-object-mechanism property of the model manager. If mechanism is equal to 'menu, a tree menu will be displayed and a selection can be made by left-clicking on the desired object. If mechanism is equal to 'display the user will be expected to select from the currently active graphics display by left clicking on the highlighted object. Objects are highlighted one at a time. The highlighted object is the object whose geometrical center is closest to the mouse position. On a platform running OpenGL, this keyword can be set to 'edge to allow selecting the objects form the graphic display by clicking on any edge of the desired object without highlighting any object.

- **:expand?**

  When t, all object in the tree under :from are expanded. When nil, only objects that are already demanded are selectable.

- **:eval**

  Quoted expression to be evaluated on the selected object to specify the return value of the function.

  Default Value: '(the), which returns the selected object

Example 16-18. Get-object Example

```
> (get-object :class 'box-object)
> (get-object :class 'box-object :test '(equal (object-name (the)) 'table-
top))
> (get-object :class 'box-object :eval '(the width))
;; returns : 1.0
```

get-objects [Function]

This function accepts the same arguments as get-object but expects the interactive selection of a list of objects instead of only one object. The selection process is exited with a right mouse click (when selecting from the graphics display). The function returns the list of object instances selected or a list of whatever value the keyword :eval evaluates to for every object. If no objects are selected before exiting, the function returns nil. A new object selection method can be used with this function by setting the :mechanism keyword to 'window: This allows the user to click and drag to define a rectangle on the current display: Objects that are fully or partly contained by the rectangle will be the ones considered by the function. **Additional Arguments:** This function defines the following additional keywords that are not defined with get-object. These additional arguments are only effective when mechanism is set to 'display:

Keyword Arguments:

- **:on-select**

  This optional keyword allows the user to specify the behavior of the objects on the display window as they are being selected. Allowed values are: :undraw, :shade, and :highlight. When :highlight is specified, the selected objects colors are changed to the color specified by the :highlight-color keyword.

  Default Value: nil

- **:highlight-color**

  Optional keyword specifying the highlight color when :on-select is set to :highlight. A color is specified by name (see color appendix) or by a list of RGB value between 0 and 1 (inclusive)

- **:reset-on-done?**

  When non-nil, resets the objects to their original status after the object selection process is done. In other words, undoes the effect of the :on-select keyword.

  Default Value: t

See Also: get-object

get-property [Function]

This function is used to allow the user to interactively select a property object or a property value from a selected object. A property menu pops-up for the user to select a property. The property menu allows navigation to subobjects by clicking on the "Browse Subobjects" item and to the parent by clicking on the "Browse Superior" item.

Keyword Arguments:

- **:eval?**

  The default returns the property object. When t, returns the property value.

  Default Value: nil

- **:from**

  Object instance specifying the root of the tree menu that is displayed when the keyword :mechanism is set to 'menu.

- **:class**

  Class name: Only objects that inherit from this class will be selectable. The default is all AML objects. This keyword can also take a list of classes: Objects that inherit from any of these classes will be selectable.

- **:test**

  Quoted AML expression to be evaluated on each object. Only objects that pass this test will be selectable.

  Default Value: all AML objects will pass the test

- **:prompt**

  String specifying the user message to be shown on the message pane. (see the function message)

- **:mechanism**

  If mechanism is equal to 'menu, a tree menu will be displayed for selecting an object, otherwise the user will be expected to select from the display. When this keyword is not supplied, the system will look for the get-object-mechanism property from the model manager.

- **property-list-method**

  Name of a method that returns a list of property object instances. This keyword is used to customize the get-property method so only a list of selected properties of a certain class are available for selection. See example below.

  Default Value: 'property-objects

- **source-object**

  Object instance to get a property from. If a valid object is given, the step to select an object is skipped, a menu showing properties of source-object is displayed.

  Default Value: nil

- **:superior-selection?**

  When nil, the "Superior" option does not appear on the property selection pop-up menu.

  Default Value: t

- **:self-selection?**

  When nil, the "Self" option does not appear on the property selection pop-up menu.

  Default Value: t

- **:browse?**

When nil, the "Browse Superior..." and "Browse Subobjects..." options do not appear on the property selection pop-up menu

Default Value: t

See Also: get-object

Example 16-19. Get-property Example

```
;; Getting a property using the default property list method
(get-property :property-list-method 'property-objects :source-object (the
part2 box1))

;; Customizing the property-list-method:

;; General behavior of the user method visible-prop-objects defined on the
class object
(define-method visible-prop-objects object ( )
   (property-objects self)
 )

;; Specific behavior of the user method visible-prop-objects defined on box-
object and cylinder-object
(define-method visible-prop-objects box-object ( )
   (list (the depth self) (the height self) (the width self))
 )

(define-method visible-prop-objects cylinder-object ( )
   (list (the diameter self) (the height self))
 )

;; Using the method visible-prop-object with get-property
(get-property :property-list-method 'visible-prop-objects)
```

get-screen-size [Function]

This function returns a list containing the screen width and height (respectively) in pixels. This function takes no arguments.

form-shell-width [Function]

Returns the platform-dependent width (in pixels) of the window border (shell) of top-level forms This function does not take any arguments.

form-title-bar-height [Function]

Returns the platform-dependent height of the title bar of top-level forms. This function does not take any arguments.

interactive-delete [Function]

This function allows the interactive deletion of objects.

interactive-shade-objects [Function]

This function allows the user to interactively select several objects to be shaded, un-shaded or faceted.

Optional Arguments:

- **shade?**

  Can take the any of the following three values: - 'facet to change the selected objects rendering to 'facet. - t to change the selected objects rendering to 'shaded. - nil to change the selected objects rendering to 'boundary.

  Default Value: t

interactive-undraw [Function]

The interactive-undraw function is called by the undraw button on the graphic-palette form.

loop-while-waiting-for-mouse-click [Function]

This function repeats the body of code until a mouse-click is detected. It returns a list specifying the mouse click, the x location in pixels and the y-location in pixels. A mouse click of 0 represents the left mouse button, 1 represents the middle mouse button, and 2 represents the right mouse button.

Arguments:

- **Window**

  The window (window handle) to monitor for the mouse-click.

- **Body**

  The body of code to perform while waiting for the mouse-click.

Example 16-20. Loop-while-waiting-for-mouse-click example

```
> (setf mouse-button (first (loop-while-waiting-for-mouse-click (current-
display-id) (print "Waiting"))))
"Waiting"
"Waiting"
"Waiting"
"Waiting"
0 ;; Result from clicking the left mouse button
```

See Also: waiting-for-mouse-click

message [Function]

This function will cause its string parameter to be displayed in the active message pane. The message is displayed on the AML command line window if no active message pane is found. The active message pane can be set by calling the function set-message-pane.

Arguments:

- **String**

  A message in the form of a string to be displayed in the message

Keyword Arguments:

- **:append?**

  Optional keyword. When :append? is t, it appends to the message already present on the message pane (message form). When :append? is nil , the existing message(s) in the message pane is (are) deleted and the new message is displayed.

  Default Value: nil

See Also: get-message, set-message-pane

Example 16-21. Message example

```
> (message "This is a test")
;; returns : "This is a test" in the message pane erasing all previous
messages.
> (message "This is also a test" :apppend? t)
```

message-box [Function]

This function displays a dialog form with the specified message and heading and allows different option for user feedback. The dialog closes and returns after the user clicks on any of the buttons on the dialog. The default return values of the function are described with the mode keyword below.

Arguments:

- **Heading**

  String displayed on the title bar of the dialog.

- **Message**

  Message string.

Keyword Arguments:

- **:mode**

  This keyword can take the following values and affect the return values of the function: :OK Allows only one option for user feedback by showing only one button on the dialog. (Ok only) Returns t. :OK-CANCEL Allows two options for user feedback by showing two buttons on the dialog. (Ok and Cancel) Returns t on Ok, and nil on Cancel. :YES-NO-CANCEL Allows three options for user feedback by showing three buttons on the dialog. (Yes, No, and Cancel) Returns 1 on Yes, 2 on No, and nil on Cancel.

- **:centered?**

  When non nil, positions the form at the center of the screen.

  Default Value: nil

- **:x**

  Position in pixels of the dialog form from left of screen. It is ignored when centered? is non nil.

- **:y**

  Position in pixels of the dialog form from top of screen. It is ignored when centered? is non nil.

- **:width**

  Width (in pixels) of the dialog form.

- **:height**

  Height (in pixels) of the dialog form.

- **:labels**

  List of strings that overwrite, when provided, the default labels of the buttons.

- **:return-values**

  When non-nil, this keyword overrides the return value behavior caused by the mode keyword. It expects a list of values that will constitute the available return values of the function. The first value would be the return value if the leftmost button was clicked, the second value would be that of the second button...

  Default Value: nil

Example 16-22. Message-box example

```
(message-box "heading test" "message test" :mode :ok-cancel :labels
'("Accept" "Abort"))
t
;;Returned T after user clicks on "Accept"
```

pop-up-menu [Function]

Opens a menu with a list of selectable items. The function returns when the user left clicks on an item or outside the menu window. When an item is selected, the value returned depends on the choice-list argument described below. If the user clicks outside the menu, the function returns nil. On Unix platforms, the menu can be opened in two styles: standard or framed. A framed menu opens within a window shell that includes a title bar. The style of the menu is specified by the frame? keyword described below. When a menu item's label is equal to an empty string, a separator widget is created instead of a regular menu item.

Arguments:

- **choice-list**

  List of strings or pairs with each pair comprised of a string and a return value. Each string/pair corresponds to a selectable item on the menu. If an item is specified with a string, that string is used to represent the item on the menu and it is returned if the item is selected. If an item is specified with a '(string value) pair, string is used to represent the item on the menu and value is returned if the item is selected.

Keyword Arguments:

- **:background-color**

  String specifying the background color of the menu. This keyword is ignored on WINDOWS platforms.

- **:foreground-color**

String specifying the text color of the menu. This keyword is ignored on WINDOWS platforms.

- **:frame?**

  This keyword is only effective on UNIX platforms. When non nil, the menu is opened as a list widget inside a window shell that includes a title bar. This keyword is ignored on WINDOWS platforms.

- **:label**

  String specifying the title of the menu window. This keyword is ignored when frame? is nil. This keyword is ignored on WINDOWS platforms.

- **:width**

  Width of the menu window in pixels. This keyword is ignored when frame? is nil. This keyword is ignored on WINDOWS platforms.

- **:max-length**

  When frame? is non nil, this keyword specifies the maximum number of visible items on the list widget used as the menu. Any remaining elements are available through a vertical scroll bar. This keyword is ignored on WINDOWS platforms.

- **:x-offset**

  Integer specifying the offset of the menu window from the left of the screen. When nil, the menu's x offset is set to the current x offset of the mouse cursor.

- **:y-offset**

  Integer specifying the offset of the menu window from the top of the screen. When nil, the menu's y offset is set to the current y offset of the mouse cursor.

- **:font**

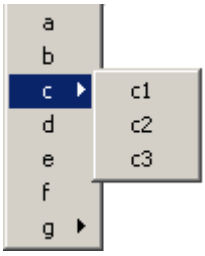
  Font of the menu text. This keyword is ignored on WINDOWS platforms.

The function pop-up-menu has been enhanced in order to allow cascading menus. This is done by the :menu feature that has to be added in the front of each menu option that has to be cascaded. The example below best illustrates its usage.

Example 16-23. Pop-up-menu Example

```
(pop-up-menu
   '(("a" 1)
     ("b" 2)
     (:menu "c" (("c1" 31) ("c2" 32) ("c3" 33)))
     ("d" 4)
     ("e" 5)
     "f"
     (:menu "g" ("g1" "g2" ("g3" 73) (:menu "g4" ("g41" "g42" "g43"))))
     ))
```

Figure 16-3. Pop-up-menu



pop-up-message [Function]

The function pop-up-message displays a message to the user. The user must then press either the OK or Cancel button to continue. The function returns the string displayed on the selected button.

Arguments:

- **Message**

  The message text to display in the message form.

Keyword Arguments:

- **:width**

  The width in pixels of the message form.

- **:foreground-color**

  The color for the message text. This keyword is ignored on WINDOWS platforms.

- **:background-color**

  The color for the background of the message window. This keyword is ignored on WINDOWS platforms.

- **:x-offset**

  The offset from the left of the screen (in pixels) at which to display the message. A nil value will cause the x-offset to be determined by the current mouse cursor on the screen. This keyword is ignored in WINDOWS platforms, and the message window always pops up in the center of the screen

- **:y-offset**

  The offset from the top of the screen (in pixels) at which to display the message. A nil value will cause the y-offset to be determined by the current mouse position on the screen. This keyword is ignored on WINDOWS platforms, and the message window always pops up in the center of the screen.

- **:done-label**

  A string which will be displayed on the left-hand (OK) button and returned by the function if the user selects this button.

- **:cancel-label**

  A string which will be displayed on the right-hand (Cancel) button and returned by the function if the user selects this button.

Example 16-24. Pop-up-message Example

```
> (pop-up-message (format nil "~a~%~a" "WARNING:" "Delete All Files?") :width
200 :done-label "Yes, I'm Sure" :cancel-label "Nope")
; Pops up the message "WARNING: Delete All Files?" on the screen for the user
to read.
; The user then selects the Yes button to dismiss the message.
"Yes, I'm Sure"
```

pop-up-text-prompt [Function]

This function opens a text prompt dialog. It allows 1 or multiple text entries. The text prompt dialog closes and the function returns when either "Ok" or "Cancel" is pressed. When "Ok" is selected, the function returns the user text string or a list of strings in the case of multiple entries. When "Cancel" is selected, the function returns nil.

Keyword Arguments:

- **:Nb-entries**

  Integer specifying the number of text entries desired.

- **:Title**

  Title string of the prompt dialog window.

- **:Prompt**

  Label string of the prompt field.

- **:init-text**

  String or list of strings specifying the initial content(s) of the prompt field(s). If nb-entries is greater than 1, a list of strings is required.

- **:input-length**

  Number of characters expected per entry. This keyword affects the width of the dialog window. This keyword is not effective on WINDOWS platforms when nb-entries = 1.

- **:x-offset**

  Offset (in pixels) of the dialog from the left of the screen. When nil, the dialog's x offset is set to the current x offset of the mouse cursor. This keyword is ignored on WINDOWS platforms when nb-entries = 1.

- **:y-offset**

  Offset (in pixels) of the dialog from the top of the screen. When nil, the dialog's y offset is set to the current y offset of the mouse cursor. This keyword is ignored on WINDOWS platforms when nb-entries = 1.

- **:foreground-color**

  Text color. This keyword is ignored on WINDOWS platforms when nb-entries = 1.

- **:background-color**

  Specifies the background color.

print-active-display-dialog [Function]

This function display a print options dialog, providing the user with several options available for printing the active graphics display. After setting the desired options, the user can then click the "Print" button to print. The dialog also allows printing to a file.

Keyword Arguments:

- **:x**

  Position (in pixels) of dialog from left of screen. When nil, the dialog pops up at current mouse position.

  Default Value: nil

- **:y**

  Position (in pixels) of dialog from top of screen. When nil, the dialog pops up at current mouse position.

  Default Value: nil

- **:print-command**

  Program (or shell command) used to print.

  Default Value: (logical-path :aml"AMLPrint.exe") on WINDOWS and "lp" on UNIX

- **:print-to-file?**

  When t, the active display is printed to a file. The user will be prompted for a file name after he/she clicks the "Print" button. Note that when an image format is selected (bmp, jpeg, tiff, gif, png), this keyword is ignored and printing is done to a file.

  Default Value: nil

- **:color?**

  When nil, print is in black and white.

  Default Value: t

- **:outlined?**

  When t, only the active viewport of the graphics display (when applicable) is printed.

  Default Value: nil

- **:format**

  Desired print format. Possible values are :ps, :hpgl :bmmp :tiff :jpeg :gif :png.

  Default Value: ps

- **:image-width**

  Image width (in pixels) when format is an image (:bmp :tiff :jpeg :gif :png). When nil, image width will be equal to the graphics' window width.

  Default Value: nil

- **:image-height**

Image height in pixels when format is an image (:bmp :tiff :jpeg :gif :png). When nil, image width will be equal to the graphics' window height.

Default Value: nil

- **:min-line-width**

  Minimum line width used when printing. This is used only with :ps and :hpgl.

  Default Value: nil

- **:line-width-scale**

  Scale of minimum line width value.

  Default Value: 1

root-window-mouse-position [Function]

This function returns the pixel location of the mouse with respect to the root window. The root window is the background area that defines the whole screen. The x y pixel locations are returned in a list.

See Also: window-mouse-position

Example 16-25. Root-window-mouse-position

```
> (root-window-mouse-position)
(141 226)
```

select-a-point-method [Function]

This function allows the user to pick a point from the current display. A menu of possible selection methods is presented. The function returns two values: The coordinates of the point selected and a quoted expression representing a formula that can be used to generate the coordinates of the point selected.

See Also: select-points-method.

Example 16-26. Select-a-point-method Example

```
(create-model 'box-object)
(draw (the))
(zoom :all)
(change-view :iso)
(select-a-point-method)
;;Select point-from option, then the box, then a vertex of the box
(0.5f0 0.5f0 -0.5f0)
(VERTEX-OF-OBJECT (THE BOX-OBJECT) 6)
(select-a-point-method)
;;Select center-of-face option, then the box, then a face of the box
(0.0f0 0.5f0 0.0f0)
(CENTER-OF-FACE (THE BOX-OBJECT) 4)
```

select-points-method [Function]

This function allows the user to pick a list of points from the current display. A menu of possible selection methods is presented. The function returns two values: A list of the coordinates of the points selected and a list of formulas that can be used to generate the coordinates of the points selected.

select-class-prompt [Function]

This function pops up a text prompt dialog that allows the user to type in a class name. The class name entered has to be that of a defined class. The dialog also contains a button that allows the user to open a class browser dialog to select the class by clicking on the desired class name. The function returns the class name selected or nil when no class is selected. The user can right-click to complete a class name after simply typing the prefix of the desired class name. If more than one match exist, a menu will pop-up showing all available classes with the given prefix. (The user must right-click inside the class name field).

Arguments:

- **Root-class-list**

  List of class name specifying the super-classes of the classes to be made available by the class browser dialog.

Keyword Arguments:

- **:default-class**

  Class name specifying the selected class by default

  Default Value: nil

- **:x**

  Offset (in pixels) of the dialog from the left of the screen. When nil, the dialog's x offset is set to the current x offset of the mouse cursor.

  Default Value: nil

- **:y**

  Offset (in pixels) of the dialog from the top of the screen. When nil, the dialog's y offset is set to the current y offset of the mouse cursor.

  Default Value: nil

- **:width**

  Width in pixels of the dialog.

- **:height**

  Height in pixels of the dialog.

- **:background-color**

  Background color string

- **:foreground-color**

  Text color string.

- **:multiple-classes?**

When t, the function returns a list of class names instead of a class name. Multiple class names can be specified by the user by simply separating them by a space character within the class name field of the form.

Default Value: nil

select-class-tree [Function]

This function opens a dialog that allows the user to select a class and returns the name of the selected class. The classes are represented in a tree form based on the AML class hierarchy starting at the root class specified The function does not return unless the user clicks on the "Ok" or "Cancel" buttons on the dialog. If the user clicks "Ok", the selected class name is returned. If the user clicks "Cancel", nil is returned. The form contains an "Update" button that allows the class tree to recreate the class hierarchy tree.

Arguments:

- **:Root-class**

  Name of root class

Keyword Arguments:

- **:x**

  Offset (in pixels) of the dialog from the left of the screen. When nil, the dialog's x offset is set to the current x offset of the mouse cursor.

- **:y**

  Offset (in pixels) of the dialog from the top of the screen. When nil, the dialog's y offset is set to the current y offset of the mouse cursor.

- **:width**

  Width in pixels of dialog.

- **:height**

  Height in pixels if dialog.

- **:foreground-color**

  Text Color.

- **:background-color**

  Background color of dialog.

select-color-dialog [Function]

This function opens a dialog that allows the interactive selection of a color It returns a list of 3 numbers representing the Red/Green/Blue (RGB) values of the selected color. RGB values are between 0 and 1. The function returns nil if no color is selected.

Keyword Arguments:

- **:RGB**

  List of rgb values of the initially selected color; rgb values are between 0 and1.

- **:x-offset**

Offset (in pixels) of the dialog from the left of the screen. When nil, the dialog's x offset is set to the current x offset of the mouse cursor. This keyword is ignored on WINDOWS platforms.

- **:y-offset**

  Offset (in pixels) of the dialog from the top of the screen. When nil, the dialog's y offset is set to the current y offset of the mouse cursor. This keyword is ignored on WINDOWS platforms.

- **:title**

  Title string of the dialog window.

- **:foreground-color**

  Text Color, ignored on WINDOWS platforms.

- **:background-color**

  Background color of dialog, ignored on WINDOWS platforms.

select-directory-dialog [Function]

This function displays the file system hierarchy so the user can select a directory for use by the application. The function returns the directory path string when a directory is selected, it returns nil otherwise. It takes the same format and arguments as the select-file-dialog function described below, except for the new? And filter keywords.

select-file-dialog [Function]

This function displays the file system hierarchy so the user can select a file for use by the application. The dialog provides the ability for the user to enter a new file name also.

Keyword Arguments:

- **:dir**

  A directory path string to be used as the starting location of the dialog.

- **:title**

  A string to be used as the title of the dialog.

- **:x**

  The horizontal pixel location at which to display the dialog. This keyword is ignored on WINDOWS platforms.

- **:y**

  The vertical pixel location at which to display the dialog. This keyword is ignored on WINDOWS platforms.

- **:foreground**

  The color for the foreground of the dialog. This keyword is ignored on WINDOWS platforms.

- **:background**

  The color for the background of the dialog. This keyword is ignored on WINDOWS platforms.

- **:resize**

To enable or disable the resize handles for the dialog when non nil. This keyword is ignored on WINDOWS, the dialog is always non-resizable.

- **:filter**

  A string in the form "*.*" used as a filter of the files allowed to be displayed in the dialog. The user can provide a list of extensions using ";" as a separator on a the filter string, i.e. "*.jpg;*.png;*.ps".

- **:work-area-bg**

  Background color of the area of the dialog where the files and directories are displayed. This parameter is ignored on WINDOWS platforms.

- **:work-area-fg**

  Foreground color (Text color) of the area of the dialog where the files and directories are displayed. This parameter is ignored on WINDOWS platforms.

- **:scroll-bar-color**

  Color of the scroll bars. This parameter is ignored on WINDOWS platforms.

- **:init-filename**

  It is a string specifying the default selected file name when the file dialog pops-up. This should only be the file name and not the entire file path.

select-font-dialog [Function]

This function pops up the WINDOWS font selection dialog and returns the user-selected font string. (The font string format as defined by AML). It returns nil if the user clicks cancel.

Keyword Arguments:

- **inti-font**

  Initial font selected

  Default Value: (trace-from (model-manager) (the interface font))

- **x**

  The x-offset of the dialog

  Default Value: 100

- **y**

  The y-offset of the dialog

  Default Value: 100

- **title**

  Title of the dialog

  Default Value: "Select Font"

Example 16-27. Select-font-dialog Example

```
(select-font-dialog :init-font "Courier,0,17,0,1" :x 20 :y 20 :title "Select
application font")
```

select-object-dialog [Function]

Opens the AML object selection dialog form and allows the user to select one or more objects. When the keyword tree-widget-object is an instance of ui-model-tree, the user will select from the tree widget specified instead of the system's object-selection-dialog. This function hangs blocks the program execution until the user ends or aborts object selection. When the system's object selection dialog is used, selection is accepted by pressing the "Ok" button or aborted by pressing the "Cancel" button. When a user specified tree widget is used for object selection, selection is ended when the user right-clicks on the tree widget. **Return Value:**The function returns the object instance selected when the keyword multiple-objects? is nil, or returns the list of object instances selection when multiple-objects? is t. The function returns nil if selection is either aborted or ended without selecting any object.

Arguments:

- **root-object**

  Desired object instance to be the root node of the object selection dialog. This argument is ignored when a valid tree-widget-object is specified.

Keyword Arguments:

- **:multiple-objects?**

  When t, the user is expected to select a list of objects. Selection is accepted by clicking on "Ok" of the system's object selection dialog is used or by right-clicking on the tree widget if a valid tree-widget-object is specified.

  Default Value: nil

- **:expand?**

  When t, all objects under the root-object specified are demanded/expanded. This keyword is ignored when a valid tree-widget-object is specified.

  Default Value: nil

- **:class**

  Root class of available objects. A defined class name must be specified.

  Default Value: 'object-root-class

- **:test**

  Quoted expression that must return t when evaluated on an object in order to make that object available. Default value is t, which means all objects pass the test.

  Default Value: t

- **:x**

  X-offset of the object selection dialog in pixels. When nil, it follows the mouse pointer position. This keyword is ignored if a valid tree-widget-object is specified.

  Default Value: nil

- **:y**

Y-offset of the object selection dialog in pixels. When nil, it follows the mouse pointer position. This keyword is ignored if a valid tree-widget-object is specified.

Default Value: nil

- **:width**

  Width of the object selection dialog in pixels. This keyword is ignored if a valid tree-widget-object is specified.

  Default Value: 310

- **:height**

  Height of the object selection dialog in pixels. This keyword is ignored if a valid tree-widget-object is specified.

  Default Value: 600

- **:background-color**

  Background-color of the object selection dialog. This keyword is ignored if a valid tree-widget-object is specified.

  Default Value: the interface background-color

- **:foreground-color**

  Text color of the object selection dialog. This keyword is ignored if a valid tree-widget-object is specified.

  Default Value: the interface foreground-color

- **:default-object-list**

  Default object list returned if the user does not select any object. Must be a list of object instances.

  Default Value: (list of objects)

- **:tree-widget-object**

  Instance of ui-model-tree. When specified, object selection is done on the tree widget specified here instead of the system's object selection dialog.

  Default Value: nil

- **:tree-widget-action**

  Quoted expression to be evaluated on every object selected when the user left-clicks on that object.

  Default Value: nil

- **:message**

  Allows displaying a message in the message pane while selecting an object.

  Default Value: nil

set-message-pane [Function]

This function is used to set the system's active message pane, i.e. where the strings passed to the function message are displayed.

Arguments:

- **Instance**

  Instance of ui-message-field-class.

set-retrieve-object-object-geometry-scale-factor [Function]

This function sets the default scale factor used when the user attempts to retrieve an object's geometry using the AML GUI.

Arguments:

- **scale-factor**

  The scale factor.

See Also: aml-save, aml-load.

set-save-object-object-geometry-scale-factor [Function]

This function sets the default scale factor used when the user attempts to save an object's geometry using the AML GUI.

Arguments:

- **scale-factor**

  The scale factor.

See Also: aml-save, aml-load.

waiting-for-mouse-click [Function]

This function suspends execution of next statement until a mouse-click has occurred. It returns a list specifying the mouse click, the x location in pixels and the y location in pixels. A mouse click of 0 represents the left mouse button, 1 represents the middle mouse button, and 2 represents the right mouse button.

Arguments:

- **Window**

  The window (window handle) to monitor for the mouse-click.

See Also: loop-while-waiting-for-mouse-click

Example 16-28. Waiting-for-mouse-click Example

```
(waiting-for-mouse-click (current-display-id))
;; returns : (0 316 24)
```

window-mouse-position [Function]

Returns the pixel location of the mouse with respect to the specified window. The x y pixel locations are returned in a multiple value form and not in a list form. (use multiple-value-list to convert multiple values into a list of values)

Optional Arguments:

- **Display-id**

  The window handle.

See Also: root-window-mouse-position

Example 16-29. Window-mouse-position Example

```
> (window-mouse-position)
;; returns :
750
486
> (multiple-value-list (window-mouse-position))
;; returns :
(750 490)
```

with-busy-cursor [Function]

With-busy-cursor is a wrapper function that allows the user to alter the display of the cursor to a stopwatch/hourglass while performing an action.

Arguments:

- **Body**

  The body of code to perform with the cursor display set as a stopwatch/hourglass.

Example 16-30. With-busy-cursor Example

```
> (with-busy-cursor (format nil "Program is running, please wait.~%") (sleep
5))
;; returns : NIL
```

with-message [Function]

This wrapper function allows the specified message to be displayed in the active message pane, and on leaving, to revert to the previous message.

Arguments:

- **Message-string**

  String to be displayed in the active message pane.

- **Body**

  The body of code to perform while the specified message is displayed in the active message pane.

See Also: message, set-message-pane

Example 16-31. With-message Example

```
> (with-message "Please wait" (func2) (func1) )
```

Application Source Code Samples

A. Box Model

The following example shows a box model managed by a form. Thanks to the apply-action and cancel-action features of the apply-cancel-mixin class, and ui-apply-button-class, ui-cancel-button-class features, as well as the AML dependency mechanism, building a user interface to manage a model does not necessarily require the definition of specialized methods or functions. In the following examples, the box-model-form requires the existence of a current model of class box-model. The apply button applies the widget values of the form to the box model properties, i.e. no change is made to the box model unless the apply button is pressed. The cancel button resets the widget values of the form to the current property values of the box model, i.e. to the values that were last applied. The form can be a subobject of any ui-layout-class, or for the sake of this example, it can be instantiated as a subobject of "(the model-manager interface forms)".

Example 16-32. Box Model Example 1

```
(define-class box-model
  :inherit-from(box-object)
  )

(define-class box-model-form
  :inherit-from(ui-form-class)
  :properties(
              ;; Property created to keep a pointer to the box model being
modeled.
              ;;Note: The function root-object does not establish any
dependency,
              ;;therefore the current-model property needs to be smashed
whenever
              ;;the root object changes
              current-model (let ((current-model (root-object))
                                  )
                              (when (typep current-model 'box-model) current-
model))
              x-offset 50 y-offset 50 height 280 width  250 label "Box Model"
              measurement 'percentage ;; This is the default formula anyway
              )
  :subobjects(
              (bdepth :class 'ui-labeled-field-class
                      x-offset 0 y-offset 0 width 100 height 10
                      label "Depth"
                      content (if ^^current-model (the depth (:from
^^current-model)) "N/A")
                      apply-action (when ^^current-model '(change-value
                                                           (the depth (:from
^^current-model))
                                                           (get-value (the
superior))))
                      cancel-action '(smash-value ^content))
              (bheight :class 'ui-labeled-field-class
```

```
                        x-offset 0 y-offset 10 width 100 height 10
                        label "Height"
                        content (if ^^current-model (the height (:from
^^current-model)) "N/A")
                        apply-action (when ^^current-model
                                       '(change-value
                                         (the height (:from ^^current-model))
                                         (get-value (the superior))))
                        cancel-action '(smash-value ^content)
                        )
               (bwidth :class 'ui-labeled-field-class
                       x-offset 0 y-offset 20 width 100 height 10
                       label "Width"
                       content (if ^^current-model (the width (:from
^^current-model)) "N/A")
                       apply-action (when ^^current-model
                                      '(change-value
                                        (the width (:from ^^current-model))
                                        (get-value (the superior))))
                       cancel-action '(smash-value ^content)
                       )
               (solid? :class 'ui-toggle-button-class
                       x-offset 0 y-offset 30 width 100 height 10
                       label "Solid"
                       status (when ^^current-model (the solid? (:from
^^current-model)))
                       apply-action (when ^^current-model
                                      '(change-value
                                        (the solid? (:from ^^current-model))
^status))
                       cancel-action '(smash-value ^status)
                       )
               (render :class 'ui-radio-buttons-class
                       x-offset 0 y-offset 40 width 100 height 10
                       labels-list '("Wire" "Shaded" "Isoline")
                       status (when ^^current-model
                                (case (the render (:from ^^current-model))
                                  ('boundary 0)
                                  ('shaded 1)
                                  ('isoline 2)))
                       apply-action (when (and ^^current-model ^status)
                                      '(change-value
                                        (the render (:from ^^current-model))
                                        (nth ^status (list 'boundary 'shaded
'isoline))))
                       cancel-action '(smash-value ^status))
               (apply :class 'ui-apply-button-class
                      x-offset 0 y-offset 90 width 25 height 10)
               (cancel :class 'ui-cancel-button-class
                       x-offset 25 y-offset 90 width 25 height 10
                       update-form? t)
               (draw :class 'ui-action-button-class
                     x-offset 50 y-offset 90 width 25 height 10
```

```
                label "Draw"
                button1-action (when ^^current-model
                                  '(draw ^^current-model))
                button3-action (when ^^current-model
                                  '(undraw ^^current-model)))
            (close :class 'ui-action-button-class
                   x-offset 75 y-offset 90 width 25 height 10
                   label "Close"
                   button1-action '(hide (the superior superior)))
            )
  )
```

Example 16-33. Box Model Example 2

```
;; Example 2 shows how the radio buttons in example 1 can be replaced by an
option menu and
;; achieve the same functionality.

(define-class box-model-form
  :inherit-from(ui-form-class)
  :properties(
;;;same as example 1 above
              )
  :subobjects(
;;;Same as example 1 above, except for the "render" subobject
              (render :class 'ui-option-menu-class
                      x-offset 0 y-offset 40 width 100
                      labels-list '("Wire" "Shaded" "Isoline")
                      options-list (list 'boundary 'shaded 'isoline)
                      selected-option (when ^^current-model (the render
(:from ^^current-model)))
                      apply-action (when (and ^^current-model ^selected-
option)
                                      '(change-value (the render (:from
^^current-model)) ^selected-option))
                      cancel-action '(smash-value ^selected-option)
                      )
              ;;...
              )
  )
```

B. Text Input Validation

The following example illustrates the use of a ui-typein-field-class in the context of validating user input before accepting it. The validation function is triggered when the return button is pressed while the field is in focus (UNIX only), or when the field looses focus (focusout-validation? = t). The validation function is also called whenever the method get-value is called on the ui-typein-field-class instance.

Example 16-34. Text Input Validation example

```
;; numberp is a lisp function that returns a non nil value
```

```
;; whenever its argument is a number.

(define-class input-validation-form
  :inherit-from(ui-form-class)
  :properties(
              x-offset 50 y-offset 50 width 200 height 250
              )
  :subobjects(
              (field1 :Class 'ui-typein-field-class
                      x-offset 0 y-offset 0 width 50 height 10
                      focusout-validation? t
                      validation-function 'numberp
                      error-message "Input is not a number"
                      content 5
                      )
              (field2 :Class 'ui-typein-field-class
                      x-offset 50 y-offset 0 width 50 height 10
                      focusout-validation? t
                      error-message "Input is not a number"
                      validation-function 'numberp
                      content 5
                      )
              (close :class 'ui-action-button-class
                     x-offset 0 y-offset 90 width 100 height 10
                     label "Close"
                     button1-action '(hide (the superior superior))
                     )
              )
  )


;; Instantiating and displaying the form from the AML command line for
testing
(setf f (add-object (the interface forms) 'f 'input-validation-form))
;; returns : (display f)

;; The user can then input a non-number into the field, hit tab to cause the
field to
;; loose focus, then notice the error message box.
;; (see description of ui-typein-field-class).
```

C. Class Selection Form

The following example defines a class selection form and the function interactive-select-class that displays that forms and returns the class name selected. The function takes the x-offset, y-offset and the root-class of the form as arguments. The form is defined as a ui-active-form-class, so when display is called, it does not return until the form is closed.This example creates the class-selection-form under "(the model-manager interface forms)", but it can actually be located under any ui-layout-class instance.

Example 16-35. Class Selection Form Example

```
(define-class class-selection-form
```

```
  :inherit-from(ui-active-form-class)
  :properties(
              x-offset 10 y-offset 10 width 300 height 500
              label "Select Class"
              root-class 'object
              selected-class (the superior tree selected-class)
              )
  :subobjects(
              (tree :class 'ui-class-tree
                    root-class ^^root-class
                    background-color 'white
                    x-offset 0 y-offset 0 width 100 height 100
                    button1-action '(hide (the superior superior))
                    )
              )
  )

(defun interactive-select-class (x y root-class)
  (let ((form
         (or
          (the interface forms class-selection-form (:error nil))
          (add-object (the interface forms) 'class-selection-form 'class-
selection-form)))
        )
    (change-value (the x-offset (:from form)) x)
    (change-value (the y-offset (:from form)) y)
    (change-value (the root-class (:from form)) root-class)
    (display form)
    (the selected-class (:from form))
    )
  )


;; from the AML command line:
(interactive-select-class 100 100 'ui-root)

;; Click on a class from the tree and the class name is returned after the
form hides.
```

D. Model Tree Form

The following example defines a form containing a model tree widget and an update button that allows updating the model tree widget after a change to the AML model tree takes place.

Example 16-36. Model Tree Form Example

```
(define-class object-tree-form
  :inherit-from(ui-form-class)
  :properties(
              x-offset 10 y-offset 10 width 300 height 500
              root-object (the model-manager)
```

```
              label "Object Tree"
              )
  :subobjects(
              (tree :class 'ui-model-tree
                    root-object ^^root-object
                    background-color 'white
                    x-offset 0 y-offset 0 width 100 height 95)
              (update-but :class 'ui-action-button-class
                          label "Update"
                          x-offset 0 y-offset 95 width 100 height 5
                          button1-action '(update (the superior superior)))
              )
  )
```

E. Form as a Series-Object

The following example illustrates a simple case of automatically generating a grid component widgets using an AML series-object.

Example 16-37. Form as a Series-Object example

```
(define-class form-series
  :inherit-from(series-object ui-grid-form-mixin ui-form-class)
  :properties(
              label "Series Form example"
              x-offset 30 y-offset 30 width 200 height 200
              measurement 'percentage
              unit-height 10   ;; 10 percent of the form height
              unit-width  50   ;; 50 percent of the form width
              ;; New props
              labels-list '("row1" "row2" "row3" "row4" "row5")
              values-list '(1 2 3 4 5)
              ;; Series properties
              recreate? nil ;; to avoid smashing the whole series in case
quantity changes
              quantity 10
              class-expression '(if (oddp !index) 'ui-grid-label-class 'ui-
grid-typein-field-class)
              init-form '(coordinates (list (floor (/ ^index 2)) (if (oddp
^index) 0 1))
                                      label   (nth (floor (/ ^index 2))
^^labels-list)
                                      content (nth (floor (/ ^index 2))
^^values-list))
              )
  :subobjects(
              (close :class 'ui-grid-action-button-class
                     coordinates '(9 0.5)
                     button1-action '(hide (the superior superior)))
              )
  )
```

F. Scrolling Subform

The following example illustrates one use of a list widget, a scrolled window and a subform. The field-series-subform is used to display a list of values and labels. It is created inside a scrolled window. The scrolled window is a subobject of the data-display-form. The data-display-form's input is a values-matrix and a labels-matrix. A list widget on the data-display-form controls what row of the values-matrix and labels-matrix is displayed by the field-series-subform.

Note : The ui-scrolled-window-class can also be used as a top-level-form?, not only as a subobject of another form as used below.

Example 16-38. Scrolling Subform Exaxample

```
;; Subform created inside a scrolled window and manages the display of
;; a list of values.

(define-class field-series-subform
  :inherit-from(series-object ui-grid-form-mixin ui-subform-class)
  :properties(
              width 200 height 400
              measurement 'pixels
              unit-height 30    ;; 30 pixels
              unit-width 80    ;; 80 pixels
              labels-list     (default)
              values-list     (default)
              ;;Series properties
              recreate? nil ;; to avoid smashing the whole series in case
quantity changes
              quantity (* (length ^values-list) 2)
              class-expression '(if (oddp !index) 'ui-grid-label-class 'ui-
grid-typein-field-class)
              init-form '(coordinates (list (floor (/ ^index 2)) (if (oddp
^index) 0 1))
                                      label   (nth (floor (/ ^index 2))
^^labels-list)
                                      content (nth (floor (/ ^index 2))
^^values-list))
              )
  )

(define-class field-series-scrolled-window
  :inherit-from( ui-scrolled-window-class)
  :properties(
              labels-list (default)
              values-list (default)
              )
  :subobjects(
              (subform :class 'field-series-subform)
              )
  )

;; Data-display-form: uses a List widget to select which list from the
values-matrix will
```

```
;; be displayed by the field-series-subform.
;;The values-matrix and the labels-matrix have been chosen arbitrarily

(define-class data-display-form
  :inherit-from( ui-form-class)
  :properties(
              label "Data Display"
              values-matrix '(
                              (a b d e f g h i j)
                              (1 2 3 4 5 6 7 8 9)
                              (k l m n o p q r s t u)
                              (10 11 12 13 14))
              labels-matrix (loop for i in ^values-matrix
                              collect
                              (loop for j in i collect (format nil "Item ~a"
j)))
              selected-index 0
              x-offset 50 y-offset 50 width 200 height 280
              )
  :subobjects(
              (list-label :class ' ui-label-class
                          x-offset 0 y-offset 0 width 100 height 10
                          label "Select Item")
              (list :class ' ui-list-class
                    x-offset 0 y-offset 10 width 100 height 30
                    labels-list '("List0" "List1" "List2" "List3")
                    items-list '(0 1 2 3)
                    selected-item 0
                    button1-action '(progn
                                     (change-value ^^selected-index
^selected-item)
                                     (update (the superior superior)))
                    )
              (fields-window :class 'field-series-scrolled-window
                             background-color "gray88"
                             x-offset 0 y-offset 50 height 40 width 100
                             labels-list (nth ^^selected-index ^^values-
matrix)
                             values-list (nth ^^selected-index ^^labels-
matrix)
                             )
              (close :class ' ui-action-button-class
                     x-offset 0 y-offset 90 height 10 width 100
                     button1-action '(hide (the superior superior))
                     )
              )
  )
```

G. Nested Apply and Cancel Actions

A ui-apply-button-class, when left clicked, triggers the apply-form-method of the apply-form it points to. By default the apply-form is its superior form. The apply-form-method only triggers the apply-actions of the subobjects of the form and does not traverse its whole tree. In order to reach nested subforms within a form, the user should link the "apply tree" as shown in the following example:

Example 16-39. Nested Apply and Cancel Actions Example

```
(define-class Nested-Apply-form
  :inherit-from(ui-form-class)
  :properties(
              ;; ...
              )
  :subobjects(
              (subform1 :class 'subform-class1
                        apply-action '(apply-form-method (the superior))
                        ;;...
                        )
              (subform2 :class 'subform-class2
                        apply-action '(apply-form-method (the superior))
                        ;;...
                        )
              (apply :class 'ui-apply-button-class)
              )
  )
```

Note: Subform1 and subform2 are user subform classes that contain widgets with apply-action properties. The same example can be applied for ui-cancel-button-class, cancel-form-method and cancel-actions.

H. Property Object Field

The following class definition describes how the AML developer can define super-classes of widgets linked to a model property object. Similar super-classes can be defined for model properties requiring other types of widgets (toggle, list, and option-menu...)

Example 16-40. Property Object Field Example

```
(define-class editable-property-field
  :inherit-from(ui-typein-field-class)
  :properties(
              model-property-object nil ;; should reference a property object
instance from the AML model
              content (if ^model-property-object (the (:from ^model-property-
object)) "N/A")
              apply-action'(when ^model-property-object
                                 (change-property-value ^model-property-object
                                                        (get-value (the
superior))))
              cancel-action '(smash-value ^content)
              )
```

)

Chapter 17. Graphical User Interface : Advanced Classes

Overview :

Chapter 17 describes advanced classes that automate the link between GUI objects and the AML model objects. It also describes application classes that allow rapid generation of AML model GUIs.

Naming convention

Unless specified otherwise:

1- A class with a "-class" name suffix is a class that can be instantiated or inherited from directly by the user.

2- A class with a "-mixin" name suffix is typically mixed (multiple inheritance) with another instantiable class in the context of defining a user class.

3- A class with a "-superclass" name suffix is a parent class that a group of instantiable classes inherit from. A super-class typically provides the properties and functionality that are shared by a group of sub-classes. For example: A box class is a super-class of a cube class. The user can inherit from a super-class.

Other classes described should not be instantiated or directly inherited from.

Preview

This chapter describes a set of design techniques and AML classes used to architecture and automate the design of an AML model user interface. These classes and tools are released within the AML ui-advanced-classes module. Throughout this chapter, it is assumed that the reader is familiar with the AML language and framework as well as the classes covered in chapter 16 of this manual. This chapter does not introduce any new widget primitive; it only enhances and automates the functionality of the base widget classes of chapter 16.

Interfacing an AML model is typically achieved by building two main objects: The data-model, and the graphical user interface (GUI) tree. The data-model consists of a set of properties and rules (formulas) and represents the "visible" knowledge of an AML model. What we mean by "visible" is the set of input/output properties and rules that the user of an AML application needs to worry about. The GUI tree consists of a hierarchy of AML widget instances communicating with the data-model.

Some property classes are defined as components of the data-model. They are described in Section 2, along with source code examples. These classes contain enough knowledge to automatically generate their corresponding widget instance in the GUI tree; this concept is further illustrated throughout this manual. Section 3 describes a set of component widget classes used in the GUI tree; each of these classes is defined to communicate with one specific property class introduced in Section 2. Section 3 classes are referred to as model interface widgets. Section 4 illustrates a set of advanced GUI classes each designed to interface a group of data-model properties (subset of the data-model); these classes are referred to as model interface widget-groups. Section 4 classes also introduce a set of properties capable of achieving a consistent look and functionality for widgets throughout the GUI of an AML application.

Finally, Section 5 presents a set of standard top-level forms and special purpose data-model classes designed to interface an AML application.

Data Model and Model Interface Properties

This section describes property classes to be used in designing a data-model. We will refer to these properties as model-interface properties since they are meant to constitute the interface between the user and the AML model through what we referred to as a data-model. A typical data-model design consists of an instance of data model object at the top level of the AML model. This data model object can be a single level object or contain a hierarchy of subobjects. The purpose of the data model is to add a level of abstraction between the user and the internal objects, geometry, processes, external programs... of an AML model. Also, Model-interface properties are defined to allow automatic link to GUI widgets and/or automate the generation of these widgets.

A. Property Classes

The following is a description of the model-interface properties available, starting by the super-class then going through the user instantiable classes alphabetically. The method ui-widget-class-name is defined with every property class. The model interface widget-groups defined in section 4 use ui-widget-class-name to determine the class of widget needed for a given property object.

model-interface-property-class [class]

This is the super-class of all model-interface property classes. This class is typically not instantiated by the user.

Inheritance: property-object

Properties:

- **Available?:**

  Flag (t or nil) specifying if the property is available/usable, usually based on the current values of the other data-model properties. A non-available property appears grayed out/disabled on the AML GUI.

- **Label:**

  String specifying the label of the property. The label is typically used by the GUI.

  Default Formula: (write-to-string (object-name (the superior))

ui-widget-class-name [Method]

Defined on Classes:

property-object

Returns ui-inspect-property-field-class. This method is defined on all model interface property classes. It is used by the AML GUI to determine the class of widgets used to interface a particular property class. Most of the classes in this Section will have their own ui-widget-class-name method that will return a different class.

See Also: ui-property-field-class

access-property-class [class]

This class is used for properties whose formula calls a specific method on a specific object instance. This is managed by the access-object and access-method. The user should not redefine the formula or change the value of an access-property-class; user input is restricted to specifying an access-object and an access-method. Typically, an access property is represented in the GUI by an action button that triggers the access-method.

Inheritance: model-interface-property-class

Properties:

- **Access-method**

  Name of the access method that the formula triggers. The access method must be defined on the class (or a super-class) of the access-object.

- **Access-object**

  Object instance that the access-method is called on.

The ui-widget-class-name method for this class returns : ui-access-button-class

computed-data-property-class [class]

This class is used for general-purpose properties whose values are not supposed to be edited by the user. These properties are typically output properties.

Inheritance: model-interface-property-class

The ui-widget-class-name method for this class returns : ui-computed-property-field-class.

See Also: The automatic-demand? property in some of the classes in next section, like ui-computed-property-field-class.

data-matrix-property-class [class]

This class is used for properties whose values expect a list of lists representing a matrix or a data table. It is in row-major order; i.e. each list within the parent list represents a row/record.

Properties:

- **columns-labels-list**

  List of strings specifying the labels of the columns/fields of the matrix/record.

- **mode**

  Widget representation of the property when using automatic widget generation. If mode is set to 'standard the data matrix widget appears in line like all other widgets. Otherwise, if mode is set to 'pop-up, the property is represented in the GUI by an action button that gives access to a stand-alone form containing the data matrix widget

  Default Formula: 'standard

- **rows-labels-list**

  List of strings specifying the labels of the rows/records of the matrix/table.

- **default-data-type**

This property specifies the data type of cells that the user does not provide a value for. For example, if row-labels-list contains a list of 4 strings (i.e. 4 rows), and the formula of the property evaluates to a list of 3 rows, the data in the 4th row is assumed to be of the type defined in default-data-type. Possible values are 'string and 'not-a-string.

Default Formula: 'string

The ui-widget-class-name method for this class returns : ui-data-matrix-property-class in standard mode and ui-data-matrix-button-class in pop-up mode.

color-selection-property-class [class]

This class is used for properties whose value expects a list of RGB values representing a color. It allows the interactive selection of a color using the AML color selection dialog. The value of this property should always be a list of 3 double-float numbers between 0 and 1 (Inclusive).

Inheritance: interactive-data-property-class

directory-selection-property-class [class]

This class is used for properties whose value expects a directory path string. It allows the interactive selection of a directory using the AML directory selection dialog.

Inheritance: interactive-data-property-class

Properties:

- **Root-directory**

  Path string of root directory of the directory dialog used for directory selection.

editable-data-property-class [class]

This class is used for general-purpose properties whose values are to be edited by the user. These properties are typically input properties.

Inheritance: model-interface-property-class

Properties:

- **editable?**

  When this property is nil, its parent property is considered a derived/output property and the background color of its GUI widget's will be that of output/computed properties. Further, the "apply-action" of its GUI widget will have no effect; therefore, its parent property value/formula will not be affected when the apply button is clicked or when the apply-action is triggered from the GUI.

- **validation-function**

  Name of a function used to validate user input. The function is user-defined and must have one argument, which is the input value entered by the user. This function must return a non-nil value for the input to be accepted. Default function (no function), accepts all user input. See data model example later in this manual.

  Default Formula: nil

- **validation-error-message**

String specifying the error message that pop-ups when the validation function runs and returns a nil value.

- **interactive-method**

  When the user right-clicks inside the field widget interfacing the editable-data-property-class instance, a menu pops up showing all the option labels specified by the interactive-method property. When an option is selected by the user, the interactive method is called on the editable-data-property-class instance. The index of the option selected is sent as the argument of the interactive method. If no option label list is provided, only one option (index = 1) is shown on the menu. The return values of the interactive method and their effect on the property object follow the same rules of the interactive method of interactive-data-property-class. Please refer to the example below.

The ui-widget-class-name method for this class returns : ui-property-field-class.

Example 17-1. Editable-data-property-class example 1

```
(define-class test-editable-data-property-class
    :inherit-from(point-object data-model-node-mixin)
    :properties(
              (coordinates :class 'editable-data-property-class
                           label "Coordinates"
                           formula '(0 0 0)
                           interactive-method '(select-point-method ("Select
a Point" "Select another Point"))
                           )
              property-objects-list (list
                                     (list (the superior coordinates self)
'(apply-formula? t))
                                     )
              )
  )

(define-method select-point-method editable-data-property-class (option)
  (message-box "Message" (format nil "select-point-method called with option
= ~a" option))
  (select-a-point-method)
  )
```

Example 17-2. Editable-data-property-class example 2

```
;;;; This code sample is provided as an example of how to perform a
particular
;;;; kind of task using foundation software provided by TechnoSoft.  It is
;;;; provided "as is" without warranty of any kind, expressed or implied.
;;;;
;;;; Licensed users of the Adaptive Modeling Language are free to use and
;;;; modify this code as long as the the original credits and disclaimers are
;;;; maintained.
;;;; ----------------------------------------------------------------------
--
;;;;  Purpose:
```

```
;;;;  This example illustrates the use of interactive-method in customizing
;;;;  editable-data-property-class and ui-property-field-class.
;;;;  It defines a new editable-data-property-class whose value/formula can
also
;;;;  be generated using a user-defined method.
;;;;  (The user-defined method will typically allow a visual mechanism to set
the
;;;;   value or the formula of the property)

(in-package :aml)

(define-class editable-data-property-ex-class
  :inherit-from(editable-data-property-class)
  :properties(
              ;; New input property
              apply-formula?  nil

              ;; Customizing inherited property
              editable?  (not ^apply-formula?)
              interactive-method '(property-field-ex-interactive-method
                                   ("Get Property Formula Method1"
                                    "Get Property Formula Method2"
                                    "Get Property Value Method1"
                                    "Get Property Value Method2"
                                    "Lock/Unlock Value"
                                    ))
              )
  )

;; This interactive-method is used by the widget class ui-property-field-ex-
class below
(define-method property-field-ex-interactive-method editable-data-property-
ex-class (index)
  (case index
    (1
     (change-value !apply-formula? t)
     ;; Here you will call method1 and return a value and a formula
     (list (object-name (the superior superior)) ;; value to show in field
widget
           '(the superior superior))             ;; formula to be applied
     )
    (2
     (change-value !apply-formula? t)
     ;; Here you will call method2 and return a value and a formula
     (list (/ (+ 5 4) 2.0)   ;; value to show in field widget
           '(/ (+ 5 4) 2.0)) ;; formula to be applied
     )
    (3
     (change-value !apply-formula? nil)
     (list 'value3)
     )
    (4
     (change-value !apply-formula? nil)
```

```
     (list 'value4)
     )
    (5
     (change-value !apply-formula? (not !apply-formula?))
     (let ((v (the))
           )
       (if (typep v 'aml-class) (object-name v) v))
     )
    )
  )

(define-method ui-widget-class-name editable-data-property-ex-class ()
  'ui-property-field-ex-class
  )

(define-class ui-property-field-ex-class
  :inherit-from(ui-property-field-class)
  :properties(

              ;; Customizing inherited properties
              content            (let ((v (the (:from ^model-property-
object)))
                                        )
                                   (if (typep v 'aml-class)
                                       (object-name v)
                                     v))
              apply-formula?     (the apply-formula? (:from ^model-property-
object))
              ;;

              )
  )

;;;; USAGE of INTERACTIVE METHODS:
;;;; ------------------------------------------------------------------------
---------------
;;;;  The interactive-method is the name of a method defined on editable-
data-property-class
;;;;  (or any subclass of it). The method must be defined with one argument
(an integer)
;;;;  The interactive-method property accepts two formats:
;;;;  1. Name of the interactive method
;;;;  Or
;;;;  2. List of the form (interactive-method-name ("option1 label" "option2
label" ... ))
;;;;     When the user right-clicks inside the field widget interfacing the
editable-data-property-class
;;;      instance, a menu pops up showing the option labels specified by the
interactive-method property.
;;;;     When an option is selected by the user, the interactive method is
automatically called on the
;;;;       editable-data-property-class instance. The argument of the
interactive method will be equal
```

```
;;;         to the index of the selected option (indices start at 1)
;;;   The interactive method must always return a list of two elements:
;;;   - First element is the generated value
;;;   - Second element is the generated formula (This is optional)
;;;   The first element returned be the interactive method is stored in the
field's content property.
;;;   When the apply-action of the widget is triggered:
;;;    If the apply-formula? property is t, the formula of the editable
property is changed to the formula
;;;    returned by the interactive method;
;;;    Otherwise, the value of the property is changed to the value returned
by the interactive method.
;;;   If the interactive method returns nil, it will have no effect on the
value or formula of the
;;;    editable property.
```

file-selection-property-class [class]

This class is used for properties that expect a string value representing a file path-name.

Inheritance: interactive-data-property-class

Properties:

- **Directory**

  Path string of the parent directory of the file.

  Default Formula: nil

- **Filter**

  String representing the file selection filter.

  Default Formula: *.*

- **Interactive-method**

  Its formula is set to 'select-file-method.

  Default Formula: 'select-file-method

select-file-method [Method]

Defined on Classes:

file-selection-property-class

Interactive method of file-selection-property-class. Opens the file selection dialog based on the property values of the file-selection-property-class instance and returns the path-name string of the selected file. Returns nil if no file is selected.

Arguments:

- **Instance**

  Instance of file-selection-property-class

- **Button**

  This is an integer. This argument is ignored for this class.

See Also: interactive-data-property-class.

file-view-property-class [class]

This class is used for properties whose value holds an ASCII file path-name string for viewing purposes. It is represented in the GUI by a button that gives access to a form that allows viewing the file content.

Inheritance: model-interface-property-class

The ui-widget-class-name method for this class returns : ui-file-view-button-class.

flag-property-class [class]

This class is used for "flag" properties that expect a value of t or nil. These properties are typically represented in the GUI with a toggle button.

Inheritance: model-interface-property-class

The ui-widget-class-name method for this class returns : ui-toggle-property-button-class. If the button is checked "on" and the original value of the property is not nil, the property value stays as is (i.e. non nil).

flag-list-property-class [class]

This class is used for properties whose value expects a list of t/nil flag. These properties are typically represented in the GUI with a toggle button group.

Inheritance: model-interface-property-class

Properties:

- **Labels-list**

  List of strings representing the labels associated with each flag in the value list. These labels are used by the GUI.

The ui-widget-class-name method for this class returns : ui-toggle-list-property-button-class.

interactive-data-property-class [class]

This class is an editable data property that also features a user-defined method that allows generating the desired value interactively/graphically. In other words, if a property value represents the coordinates of a point in space, the interactive method of this property will allow the user to select a point graphically instead of forcing him/her to type it.

Inheritance: model-interface-property-class

Properties:

- **Interactive-method**

  Name of the interactive method. The interactive method is a user method defined on Interactive-data-property-class (or any subclass of it). This method is triggered when the user clicks on the action button associated with the property field. This method should be defined with a button-index argument. When the user clicks the selection button of the property, the button-index is equal to 1 in the event of a left-click and 3 int the event of a right-click (Refer to ui-interactive-property-field-class). The interactive method should always return a list of two elements: The first element is the desired value of the property and the second element is the desired formula of the property. In case the formula is not needed, the returned list could only have one element: The desired value.

The ui-widget-class-name method for this class returns : ui-interactive-property-field-class.

Example 17-3. Interactive-data-property-class example 1

```
(define-class point-class
        :inherit-from(point-object)
        :properties(
                (coordinates :class 'interactive-data-property-class
                        formula '(0.0 0.0 0.0)
                        interactive-method 'interactive-select-point
                        )
                )
  )
;; Assumption:
;; The functions select-a-point-from-graphics-window and select-a-point-from-
menu are user functions.
;; They return a list of the '(x y z) coordinates of a point
(define-method interactive-select-point interactive-data-property-class
(button-index)
  (case button-index
    (1 (let ((point-coords (select-a-point-from-graphics-window))
             )
          (when point-coords (list point-coords nil))))
    (3 (let ((point-coords (select-a-point-from-menu))
             )
          (when point-coords (list point-coords nil))))
    )
  )
```

Example 17-4. Interactive-data-property-class example 2

If the apply-formula? property of the widget is set to t (as shown in the property-objects-list formula below), the widget interfacing this property will trigger a change-formula (instead of a change-value) operation when it is applied, otherwise it will trigger a change-value. The desired value and formula are returned by the interactive method (as shown in interactive-select-point below).

```
(define-class coordinate-from-point-object-class
        :inherit-from(data-model-node-mixin)
        :properties(
                (coordinates :class 'interactive-data-property-class
                        formula '(0.0 0.0 0.0)
                        interactive-method 'interactive-select-point
                )
        property-objects-list
        (list
        (list (the superior coordinates self) '(apply-formula? t )))
        )
 )
```

```
(define-method interactive-select-point interactive-data-property-class
(button)
        (let ((object (get-object :class 'point-object))
                )
        (when object
                (list
                (the coordinates (:from object)) ;; desired value
                (find-the-trace (the coordinates self (:from object)) :from
self) ;; desired formula
                )
          )
        )
 )
```

option-property-class [class]

This class is used for properties whose value is equal to one element of a list of available options. These properties can be represented with an option menu, a radio button group, or a combo box. A combo box is typically used whenever the value of the option-property-class is allowed to be different for any of the available options.

Inheritance: model-interface-property-class

Properties:

- **Labels-list**

  List of strings specifying a label for each option. Label-list is ignored when mode is 'combo because a user is allowed to edit an option making options and labels one entity.

- **Mode**

  GUI mode specifying the GUI representation of the property. Allowed values are 'radio,' menu, and 'combo (See Section 3 for corresponding GUI Widgets). Default formula is 'radio.

- **Options-list**

  List of available options. The value nil cannot be a member of the options-list property.

The ui-widget-class-name method for this class is defined as follows :

```
(case !mode
        ('combo 'ui-combo-property-field-class)
        ('radio 'ui-option-property-radio-buttons-class)
        ('menu 'ui-option-property-menu-class))
```

See Also: ui-combo-property-field-class, ui-option-property-radio-buttons-class, ui-option-property-menu-class

object-selection-property-class [class]

This class is used for properties whose value expects either an object instance or a property from a selected object instance.

Inheritance: model-interface-property-class

Properties:

- **Allowed-classes-list**

  List of class names specifying the classes of objects allowed as a potential value for the property. Allowed-classes-list does not need to be a list and can just specify a class name if only one class is allowed.

- **Error**

  Specifies what to be returned if a "the" reference on the property leads to an error.

- **Selection-property-name**

  Specifies the name of the desired property of the selected object. If the actual object instance is desired, this property must be set to 'self.

  Default Formula: 'self

- **Object-selection-method**

  Name of a user-defined method. This method is used by the following widgets: ui-property-selection-button-class, ui-object-selection-button-class. This method is called on every object selected by the user and returns an object instance. The object instance returned by this method is considered the desired object. The default formula is 'object-selection-method which is a method that returns the same object selected. In other words, the default behavior is that the object selected by the user is considered the desired object. The user can define a method on an AML class and specify that method name here in order to allow the user to select an object but return another object. The method specified here must always return an object instance. Example: The object-selection-method is typically used to select an object that does not have a graphical representation or does not appear in the model tree. Assume the example below: We have a coordinate system class representing the location of three candidate objects. The selected object of the three is based on the value of the property status. All three objects do not have a graphical representation and the coordinate system is typically drawn to represent them. When the user selects a coordinate system, one of the candidate objects is returned.

```
;; This is the class where the object selection is needed
(define-class object-selection-class
  :inherit-from(data-model-node-mixin)
  :properties(
        (selected-object :class 'object-selection-property-class
                formula nil
                label "Select Object"
                object-selection-method 'my-object-selection-method
                )
        )
)

;; This is the default behavior of my-object-selection-method
(define-method my-object-selection-method object-root-class ()
        self ;; returns the object itself
        )
```

```
;; This is the coordinate system class that contains the 3 candidate ob-
jects
(define-class my-coordinate-system-class
:inherit-from(coordinate-system-class)
:properties(
        status 1
        (object1 :class 'object
                )
        (object2 :class 'object
                )
        (object3 :class 'object
                )
        )
)

;; Customization of my-object-selection-method to my-coordinate-system-
class
(define-method  my-object-selection-method my-coordinate-system-class ()
(case !status (1 !object1) (2 !object2) (3 !object3))
)
```

- **Property-list-method**

  If "selection-property-name" is nil, a menu pops up asking the user to select a property when an object is selected. The pop-up menu displays the labels/name of the property object returned by the method specified in "property-list-method". "property-list-method" is a new property of object-selection-property-class and ui-property-selection-button-class. It expects the names of a method that returns a list of property object instances.

- **Eval?**

  If eval? is nil, the generated formula will contain a "self" at the end of the the-reference statement.

  Default Formula: t

Note : Allowed-classes-list and Selection-property-name are effective only when using a ui-property-selection-button-class as the widget class of this property, which is the default behavior of the ui-widget-class-name method below. The ui-widget-class-name method for this class returns : ui-property-selection-button-class.

object-filtered-selection-property-class [class]

This class has the same functionality as object-selection-property-class with one addition. This class has a cadidate-objects-list property.

Inheritance: object-selection-property-class

Properties:

- **candidate-objects-list**

  When this property is nil, this class behaves exactly like the parent class. When this property contains a list of objects, only the objects in this list can be highlighted. When selecting an object graphically and the candidate-objects-list property contains a list of objects, only the objects in this list can be highlighted. Whe selecting an object from the tree, you will be able to select any object in the tree but

an informal message will be displayed stating that the currently selected object is invalid and will state the list of valid class types and candidate objects that can be selected from.

Default Formula: nil

•

Example 17-5. object-filteres-selection-property-class example 2

```
;;;;  Purpose: object-filtered-selection-property-class example code.
;;;;
;;;;  Usage: Add an instance of the class 'test-property-container-class.
;;;;         Draw all of the objects and fit the drawn graphical objects to
the screen.

(in-package :aml)

(define-class test-property-container-class
  :inherit-from (object)
  :properties (
               ;;; Default Behavior
               ;;; Should behave just like object-selection-property-class.
               (property-1 :class 'object-filtered-selection-property-class
                           formula nil
                           ;; The formula for candidate-objects-list is nil
by default.
                           )

               ;;; Just the candidate-objects-list property formula is
modified.
               ;;; Only objects from the candidate-objects-list can be
selected.
               (property-2 :class 'object-filtered-selection-property-class
                           formula nil
                           candidate-objects-list (list
                                                   (the superior superior
box-object-1)
                                                   (the superior superior
box-object-4)
                                                   (the superior superior
cylinder-object-2)
                                                   )
                           )

               ;;; Just the allowed-classes-list property formula is
modified (inherited from object-selection-property-class).
               ;;; Only objects that are instances of 'box-object or
instances of classes that inherit from 'box-object can be selected.
               (property-3 :class 'object-filtered-selection-property-class
                           formula nil
                           allowed-classes-list '(box-object)
                           ;; The formula for candidate-objects-list is nil
by default.
```

```
                        )

              ;;; Both the candidate-objects-list property formula and the
allowed-classes-list property formula are modified.
              ;;; Only objects from the candidate-objects-list that are
instances of 'box-object or instances of classes that inherit
              ;;; from 'box-object can be selected.
              (property-4 :class 'object-filtered-selection-property-class
                          formula nil
                          allowed-classes-list '(box-object)
                          candidate-objects-list (list
                                                  (the superior superior
box-object-1)
                                                  (the superior superior
box-object-4)
                                                  (the superior superior
cylinder-object-2)
                                                  )
                          )
              )
  :subobjects (
              (box-object-1 :class 'box-object
                            orientation (list
                                         (translate (list 0 0 0))
                                         )
                            )
              (box-object-2 :class 'box-object
                            orientation (list
                                         (translate (list 1.5 0 0))
                                         )
                            )
              (box-object-3 :class 'box-object
                            orientation (list
                                         (translate (list 3.0 0 0))
                                         )
                            )
              (box-object-4 :class 'box-object
                            orientation (list
                                         (translate (list 4.5 0 0))
                                         )
                            )
              (cylinder-object-1 :class 'cylinder-object
                                 orientation (list
                                              (translate (list 0 -1.5 0))
                                              )
                                 )
              (cylinder-object-2 :class 'cylinder-object
                                 orientation (list
                                              (translate (list 1.5 -1.5
0))
                                              )
                                 )
              )
```

```
    )

  (define-method property-classification-list test-property-container-class
  ()
    '(
      ("Main Properties"
       (
        property-1
        property-2
        property-3
        property-4
        )
       )
      )
    )
```

allowed-selection? [Method]

Defined on Classes:

object-selection-property-class

Arguments:

- **Property-instance**

  Instance of object-selection-property-class.

- **Object**

  Object instance selected

The user can redefine this method on object-selection-property-class or any user-defined class that inherits from it. When the method returns nil, it notifies that the object instance selected is invalid. It is useful in order to disable the selection of certain objects. This method is effective only when using ui-property-selection-button-class or ui-object-selection-button-class in the GUI (Refer to Section 3). By default, this method simply returns the object instance passed to it.

Example 17-6. Allowed-selection? example

The following example shows the usage of the method allowed-selection? along with a user-defined object selection property class that allows the selection of box-object instance whose width property falls within a specified range.

```
(define-class box-selection-property-class
        :inherit-from(object-selection-property-class)
        :properties(
                        width-range '(0.5 1.5)
                )
        )

(define-method allowed-selection? box-selection-property-class (box-object)
        (and
(< (the width (:from box-object)) (nth 1 !width-range))
        (> (the width (:from box-object)) (nth 0 !width-range))
        ))
```

objects-selection-property-class [class]

This class is used for properties whose value expects a list of object instances.

Inheritance: model-interface-property-class

Properties:

- **Allowed-classes-list**

  List of class names specifying the classes of objects allowed as a potential element of the object list. Allowed-classes-list does not need to be a list and can just specify a class name if only one class is allowed. It can also accept property classes causing the following behavior. When an object is interactively selected (from the graphics or from the tree), a property selection menu pops-up to allow the user to select a property. Remark: An object in the graphics display will not be selectable unless its class name is a member of allowed-classes-list, which means properties of that object are not selectable either.

- **Object-selection-method**

  Name of a user-defined method. This method is used by the widget ui-multiple-object-selection-button-class. It is called on every object selected by the user and returns an object instance. The object instance returned by this method is considered the desired object. The default formula is 'object-selection-method which is a method that returns the same object selected. In other words, the default behavior is that an object selected by the user is considered the desired object. The user can define a method on an AML class and specify the method name here in order to allow the user select an object but return another object. The method specified here must always return an object instance. See object-selection-method example above.

  Default Formula: 'object-selection-method

The ui-widget-class-name method for this class returns : ui-multiple-object-selection-button-class.

allowed-selection? [Method]

Defined on Classes:

objects-selection-property-class

The user can redefine this method on objects-selection-property-class or any user-defined class that inherits from it. When the method returns nil, it notifies that the object list selected is invalid. It is useful in order to disable certain selections. This method is effective only when using ui-multiple-object-selection-button-class in the GUI (Refer to Section 3). It returns the object list by default.

Arguments:

- **Property-instance**

  Instance of objects-selection-property-class.

- **Object-list**

  Object list selected

point-selection-property-class [class]

This class is used for properties whose value expects the coordinates list of a point. The interactive method of this property allows the user to select a point from the grid specified by the property grid-object. If no grid-object is specified, a point is interactively selected by calling "select-a-point-method". If the user wishes the interactive method to generate a new formula for this property (not just a value), the apply-formula? property of the widget interfacing this property must be set to t. This can be achieved by using the enhanced format of property-objects-list. See example below. It is necessary to set "apply-formula?" to t if a dependency on the object from which the point was selected. from is needed. Refer to select-a-point-method for more information on the formulas generated.

Inheritance: interactive-data-property-class

Properties:

- **Grid-object**

  Instance of grid-class (or any sub-class of it)

See Also: select-a-point-method, grid-on-plane-class, grid-on-surface-class

Example 17-7. Point-selection-property-class Example

```
(define-class point-selection-object
        :inherit-from(data-model-node-mixin)
        :properties(
                (point :class 'point-selection-property-class
                        formula '(0.0 0.0 0.0)
                        )
                property-objects-list
                (list (list (the superior point self) '(apply-formula? t)))
                )
        )
```

range-property-class [class]

This is an editable property that requires its value to be a real number within a certain range.

Inheritance: editable-data-property-class

Properties:

- **range**

  A list of 2 numbers '(min max) defining the allowed range that the value of this property needs to be within. Default formula indicates a range of -infinity to +infinity.

  Default Formula: nil

- **range-tolerance**

  This tolerance is used to check the equality of a certain value to the boundaries of the range.

  Default Formula: 0.00001

- **range-inclusive?**

This property is a list of 2 t/nil values indiciating whether the allowed values of the property can be equal to the boundary values specified in the range. The first flag is for the left bound, the second is for the right bound.

Default Formula: '(t t)

text-view-property-class [class]

This class is used for properties whose value holds a string for viewing purposes. It is represented in the GUI by a button that gives access to form that allows viewing the string.

Inheritance: model-interface-property-class

The ui-widget-class-name method for this class returns : ui-text-view-button-class.

value-list-property-class [class]

This class is used for properties whose value expects a list of values. The list of values is a selection from a list of available values. This property is represented in the GUI by an editable field. An action button on the left side of the editable field can open a form to allow selection of values from a list of available values as well as manipulation in the order of the selected values.

Inheritance: interactive-data-property-class

Properties:

- **available-values-list**

  List of available values.

- **available-labels-list**

  List of strings specifying the labels of the available values. The default formula turns the available values into strings.

- **allow-duplicates?**

  When t, allows a value to be selected more then once from the list of available values.

  Default Formula: (default nil)

- **available-values-label**

  String specifying the header label (as it appears on the GUI form) of the available values list.

  Default Formula: (default "Available Values")

- **selected-values-label**

  String specifying the header label (as it appears on the GUI form) of the selected values list.

  Default Formula: (default "Selected Values")

ui-form-class-name [Method]

Defined on Classes:

value-list-property-class

This method returns the class name of the form that is opened by the widget interfacing the value-list-property-class. This method returns ui-value-list-property-form-class by default. This method can be redefined by the user to return the name of a user-defined form if the user wishes to customize the form's

behavior. The user-defined form must inherit from ui-value-list-property-form-class in order for the interface to behave as expected.

See Also: ui-multiple-value-selection-form-class, ui-value-list-property-form-class

var-unit-data-property-class [class]

This class is for properties that require the handling of unit conversion, where the input formula/value that the user provides is converted to another unit that the model uses. The user formula/value is specified in value-in. The user should not edit the formula of a var-unit-data-property-class instance. The formula converts value-in from unit-in to unit. **For unit expressions and unit conversion, please refer to the unit conversion section of chapter 14 of the AML reference manual.** A "the" reference to a var-unit-data-property-class will always return the converted value.

Inheritance: model-interface-property-class

Properties:

- **Available-units-list**

  List of unit expressions specifying the allowed units for unit-in.

  Default Formula: (Default (dimensionnally-similar-units-list ^unit)

- **Unit**

  Unit expression to be used as the unit to convert to. A "the" reference to the property will always return the value in this unit.

  Default Formula: (Default 'length)

- **Unit-in**

  Unit expression to be used as the input unit to the property.

  Default Formula: (Default 'length)

- **Units-labels-list**

  List of strings specifying the labels of the available units. This list is used by the ui-var-unit-property-field-class widget described in section 3.

- **Value-in**

  User input formula/value to the property.

  Default Formula: 0

The ui-widget-class-name method for this class returns : ui-var-unit-property-field-class.

B. Data Model Example

The following is a simple example of a data model using some of the model-interface property classes described above. Refer to section 5 for more AML application examples.

```
(define-class data-model
:inherit-from(object)
:properties(
            (dim1 :class 'editable-data-property-class'
                  formula 5
```

```
                label "D1 (cm)"
                validation-function 'numberp
                validation-error-message "Dim1 must be a number"
                )
          (dim2 :class 'editable-data-property-class
                formula 5
                label "D2 (cm)")
          (dim3 :class 'option-property-class
                options-list '(10 20 30 40)
                labels-list '("10" "20" "30" "40")
                mode 'menu
                formula 10
                label "D3 (cm)"
                )
          (dim4 :class 'var-unit-data-property-class
                value-in 1
                unit-in 'm
                unit 'cm
                label "D4 (m)")
          (operation :class 'option-property-class
                     options-list '(* + / -)
                     labels-list '("multiply" "add" "divide" "subtract")
                     mode 'radio
                     formula '*
                     label "Operation")
          (result :Class 'computed-data-property-class
                  formula (when ^operation
                            (apply ^operation (list ^dim1 ^dim2 ^dim3
^dim4)))
                  label "Result")
          )
  )
```

Model Interface Widgets

Introduction

This section describes a set of widget classes corresponding to the model-interface property classes described in section 2. These classes are called model interface widgets and are defined to communicate with the model-interface property classes. They are instantiated as components of the GUI tree. While using model interface widgets, the user only needs to link each widget to a property object in the AML model via the model-property-object. When the model property object is of the correct type, the widget is capable of reading from that property and writing to it without requiring the user to write specific "read", "apply" or "cancel" methods.

For example:

Assume the current AML model contains the property density. Density is a property of class editable-data-property-class. As described later, ui-property-field-class is a widget class defined to manage editable-data-property-class instances. Here is how a ui-property-field-class would be used:

```
...
(density-field :Class 'ui-property-field-class
        model-property-object (the current-AML-model density self)
        coordinates '(0 0)
        )
...
```

The user only needs to specify what property object the density-field widget manages and the position of the widget on its parent form. The widget automatically derives its content and other properties from the model-property-object specified; moreover, the apply-action and cancel-action of the widget are also defined to write to the model-property-object and re-read from the model-property-object respectively. This design dramatically reduces the amount of programming required and makes the source code more robust and maintainable. Model interface widgets are always defined/instantiated as subobjects of ui-group (i.e. form, subform, and frame...); they can also be automatically generated as described in Section 4.

Widget Classes and Methods

The following is a description of the model interface widgets available, starting by the super-class then going through the user instantiable classes alphabetically:

model-interface-widget [class]

This class is the super-class of all model interface widgets. It should not be instantiated by the user.

Inheritance: object

User input properties:

Properties:

- **Automatic-apply?**

  When non-nil, automatically triggers the apply-action whenever a new value is "accepted". In other words, the new status/content/value for the widget is applied automatically to the model-property-object without the need for the user to hit an apply button. "Accepting" a status/content/value depends on the type of widget. Please refer to sub-classes of model-interface-widget for specific description.

  Default Formula: nil

- **Model-property-object**

  Instance of a model-interface-property-class property. It can also be an instance of any property-object, but this may require the user to redefine some of the input/output formulas of the model interface widget classes described in this section. Every instantiable sub-class of model-interface-widget is defined to better communicate with a specific sub-class of model-interface-property-class

  Default Formula: nil

Input Properties Derived from Model:

Properties:

- **Apply-action**

  Specifies the apply-action of the widget. It changes the value of the property instance specified by model-property-object to whatever the current content/status/value of the widget is.

  Default Formula: '(apply-action-method !superior)

- **Gray?**

  Specifies the availability of the widget. This value depends on the available? value from the model-property-object. When available? is non-nil, gray? is nil and vice-versa. The user does not need to specify a formula for this property unless he/she wishes to overwrite its behavior.

- **Label**

  Holds the label from the model-property-object specified. When a label is not found, it defaults to the name from the model-property-object. When no model-property-object is specified, it is an empty string. The user does not need to specify a formula for this property unless he/she wishes to overwrite its behavior.

ui-access-button-class [class]

This is an action button widget defined to communicate with access-property-class. Its model-property-object is typically an access-property-class instance

Inheritance: model-interface-widget. ui-grid-action-button-class

Properties:

- **Access-method**

  Holds the access-method from the model-property-object. Equals nil when no model-property-object is specified.

- **Access-object**

  Holds the access-object from the model-property-object. Equals nil when no model-property-object is specified.

- **Access-value**

  The value from the model-property-object.

- **Button1-action**

  Calls the access-method on the access-object instance and stores the returned value in access-value.

Note : The automatic-apply? property is ignored for this class. The get-value method for this class returns the access-value of the widget.

ui-combo-property-field-class [class]

This is a combo box widget defined to communicate with option-property-class. Its model-property-object is typically an option-property-class instance. If automatic-apply? is non-nil, the apply-action is triggered when either of the following is true: 1. The widget looses focus (hitting the tab key while the widget is in focus works as well). 2. The user hits the return key while the widget is in focus. As of AML3.3, hitting the return key only works on non-WINDOWS platforms. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil.

Inheritance: model-interface-widget, ui-grid-combo-box-class

Properties:

- **Cancel-action**

  Smashes the value of the content property so the widget resets its content to the model-property-object's value when the cancel-action is triggered. (Update must be called to reflect the reset).

- **Content**

  The value from the model-property-object. Equals nil when no model-property-object is provided. The content does not necessarily equals to a member of the options-list.

- **Options-list**

  The options-list from the model-property-object. The value nil cannot be a member of the options-list property.

See Also: ui-grid-combo-box-class

ui-computed-property-field-class [class]

This is a ui-property-field-class whose editable?, apply-action and cancel-action properties are set to nil. By default its field background color is light gray. This widget is defined to communicate with computed-data-property-class.

Inheritance: model-interface-widget, ui-grid-labeled-field-class

Properties:

- **Automatic-demand?**

  When nil, the widget does not demand the value from model-property-object in case that property value is unbound, and shows "unbound" instead. A button that demands the value when clicked is provided on the right side of the widget.

  Default Formula: t

See Also: ui-property-field-class

ui-data-matrix-button-class [class]

This is an action button representing a data matrix property. Its model-property-object typically points to an instance of data-matrix-property-class. When clicked, it opens a stand-alone scrollable form containing a data matrix (data table) widget that interfaces the model-property-object instance.

Inheritance: model-interface-widget, ui-grid-action-button-class

User Input Properties:

Properties:

- **Range**

  Width and height of the scrolling area of the data matrix form that this button opens.

  Default Formula: '(500 500)

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **Apply-action**

  Default Formula: nil

- **Button1-action**

  Default Formula: Opens/Creates the data-matrix-form-class instance.

- **Cancel-action**

  Default Formula: nil

- **Data-matrix-form-class**

  Class name of the data matrix form that this button opens. Defaults to 'ui-data-matrix-form-class. Can be redefined by the user if he/she wishes to create his/her own data matrix form.

Note: The automatic-apply? property is ignored for this class.

ui-data-matrix-property-class [class]

This is a field matrix widget (a matrix of field widgets) defined to communicate with data-matrix-property-class. Its model-property-object is typically an instance of type data-matrix-property-class.

Inheritance: model-interface-widget, ui-grid-component-mixin, ui-field-matrix-class, apply-cancel-mixin

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **Cancel-action**

  Smashes the value of the content property so the widget resets its content to the model-property-object's value when the cancel-action is triggered. (Update must be called to reflect the reset).

- **Columns-labels-list**

  The columns-labels-list from model-property-object. Equals nil when no model-property-object is provided.

- **Content**

  The value from the model-property-object. Equals nil when no model-property-object is provided. It is a list of list of values representing a matrix in row major order.

- **Field-height**

  Default Formula: (if (> ^num-rows 0) (- (floor (/ ^height (+ 1 ^num-rows))) 3) 0)

- **Field-width**

  Default Formula: (if (> ^num-columns 0) (- (floor (/ ^width (+ 1 ^num-columns))) 3) 0)

- **Num-columns**

  Formula is set to the length of the columns-labels-list.

- **Num-rows**

  Formula is set to the length of the rows-labels-list.

- **Rows-labels-list**

The rows-labels-list from the model-property-object. Equals nil when no model-property-object is provided.

Note : The automatic-apply? property is ignored for this class.

ui-file-selection-property-field-class [class]

It is used to interface property objects with a value that is a file path string. This widget is a composite widget consisting of an action button that is horizontally adjacent to a text field. Clicking on the action button will pop-up a file selection dialog that allows the user to browse the file system and select a file, or type-in a new file name.

Inheritance: model-interface-widget, ui-grid-component-mixin ui-action-field-pair-class

Properties:

- **directory**

  Specifies a directory path (string) that is the initial directory that the file selection dialog open in. If this property is nil, the file selection dialog will open in the last directory browsed.

  Default Formula: nil

- **filter**

  Specifies the file filter to use in the file selection dialog.

  Default Formula: "*.*"

Example 17-8. Ui-file-selction-property-field-class example

```
;;Example of using ui-file-selection-property-field-class in conjunction with
the
;;method property-classification-list:

(define-class my-file-test-class
  :inherit-from(object)
  :properties(
              file1   nil
              file2   nil
              outfile nil
              )
  )

(define-method property-classification-list my-test-class ()
  '(
    ;; Input Category
    ("Input"
     (
      (file1 (foreground-color "blue" directory (logical-path :temp) filter
"*.txt") ui-file-selection-property-field-class)
      (file2 (foreground-color "red" directory (logical-path :temp) filter
"*.txt") ui-file-selection-property-field-class)
      )
     )
    ;; Output Category
```

```
    ("Output"
     (
      (outfile nil ui-file-selection-property-field-class)
     )
    )
   )
)
```

ui-file-view-button-class [class]

This is an action button typically representing a file-view-property-class. Its model-property-object typically points to an instance of file-view-property-class. It opens a file editor for viewing the content of the ASCII file specified by file-name.

Inheritance: model-interface-widget, ui-grid-action-button-class

Input Properties Derived from Model: (Do not need to be set by user)

Properties:

- **apply-action**

  Default Formula: nil

- **button1-action**

  Opens the form to view the file specified by file-name. This property should not be set by the user.

- **cancel-action**

  Default Formula: nil

- **file-name**

  Holds the value from model-property-object.

Note: The automatic-apply? property is ignored for this class.

ui-inspect-property-field-class [class]

This is an editable field widget with an action button located at its left. The action button opens a form that allows editing the formula of the property being edited. By default, the field background color is white. If automatic-apply? is non-nil, the apply-action is triggered every time either of the following is true: 1. The field loses focus (can also happen by hitting the tab key while the field is in focus). 2. The user hits return while the widget is in focus. As of AML3.3, hitting the return key only works on non-WINDOWS platforms. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil.

Inheritance: model-interface-widget, ui-grid-component-mixin, ui-action-field-pair-class

User Input Properties:

Properties:

- **Automatic-demand?**

  When nil, the property value is not automatically demanded when the widget is displayed; the field will show "<unbound>". Also, a demand button will appear in the gui.

  Default Formula: nil

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **Content**

  Default formula gets the value of the property being edited. It is equal to "<Unbound>" if the property being edited has not yet been demanded and automatic-demand? is equal to nil.

  Default Formula: "<Unbound>"

ui-interactive-property-field-class [class]

This is a composite widget class constituted of an action button horizontally adjacent to a text field. The action button is on the left side. As in any typical field widget class, the output of the widget is the content of the text field. The text field can be editable, and the action button provides the user with the ability of running a function/method whose return value is thrown into the text field. This widget is defined to communicate with interactive-data-property-class. Its model-property-object is typically an instance of interactive-data-property-class. If automatic-apply? is non-nil, the apply-action is triggered when either of the following is true: 1. The widget looses focus (hitting the tab key while the widget is in focus works as well). 2. The user hits the return key while the widget is in focus. As of AML3.1.3, hitting the return key only works on non-WINDOWS platforms. 3. The user changes the content of the widget by clicking on the action button which runs the interactive-method. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil.

Inheritance: model-interface-widget, ui-grid-component-mixin, ui-action-field-pair-class

User Input Properties:

Properties:

- **Apply-formula?**

  When apply-formula? is t :Whenever the user edits the value in the text widget and hits apply, a change-value occurs. On the other hand, if the interactive method is triggered and the user hits apply without editing the text in the field, a change-formula occurs. When apply-formula? is nil : Only a change-value occurs when the user clicks apply.

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **Button1-action**

  Triggers the interactive-method with the button index parameter = 1: Stores the return value of the interactive-method in content and updates the widget. (Refer to interactive-data-property-class).

- **Button3-action**

  Triggers the interactive-method with the button index parameter = 3. Stores the return value of the interactive-method in content and updates the widget. (Refer to interactive-data-property-class).

- **Cancel-Action**

  Smashes the value of the content property, so the widget resets its content to the model-property-object's value when the cancel-action is triggered. (Update must be called to reflect the reset).

- **Content**

The value from the model-property-object. Equals an empty string when no model-property-object is specified.

- **Field-background-color**

  Formula is set to "white" when editable? is non-nil, "gray" otherwise.

- **Interactive-method**

  Name of the interactive method. This property holds the value of the interactive-method from the model-property-object. Equals nil if no model-property-object is specified.

ui-multiple-object-selection-button-class [class]

This is a composite widget class constituted of an action button horizontally adjacent to a non-editable text field. The action button is on the left side. This widget is defined to communicate with objects-selection-property-class. Its model-property-object is typically an instance of objects-selection-property-class. **Usage and Behavior:** - The user left-clicks on the button and selects a list of objects (If a list of objects is already selected, the results of the new selection is appended to the list). The method allowed-selection? is then called on the model-property-object and is passed the selected objects list. If allowed-selection? returns a non-nil value, the object-selection-method is called on each selected object and returns the desired object in order to form the desired object list. By default the desired object is the selected object (Default object-selection-method returns the object itself). The desired objects list is then stored in the selected-object-list property. If the user left-clicks on the button, object selection follows the global object selection mechanism (set from the AML menu bar). If the user right-clicks on the button, selection is via a tree menu. Right clicking also allows deselecting the current list. - When the apply-action is triggered, the formula of the property instance specified by model-property-object is changed to reference the list of objects specified by selected-object-list. - If automatic-apply? is non-nil, the apply-action is triggered every time a list of objects are selected. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil.

Inheritance: model-interface-widget, ui-grid-component-mixin, ui-action-field-pair-class

User Input Properties:

Properties:

- **Parent-object**

  Object instance specifying the root of the tree of selectable objects.

  Default Formula: nil

- **tree-widget-object**

  This property should be equal to an instance of ui-model-tree. When the user is prompted by the application to interactively select an object or a property from the model tree, the window corresponding to the ui-model-tree given here is used for interactive selection (Left-click to select an object from the tree, right-click to abort). When this property is nil, a default model tree form pops-up for selection. Please refer to the method tree-widget-object.

  Default Formula: (when (the superior model-property-object) (tree-widget-object (the superior model-property-object)))

- **candidate-objects-list**

This property will filter the objects that the user can select from. When this list is 'nil, no additional filter will be applied beyond what the class did previously.

Default Value: nil

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **Apply-action**

  Changes the formula of the property instance specified by model-property-object to the list of objects found in selected-object-list.

- **Cancel-action**

  Smashes the value of the selected-object-list property.

- **Class-name**

  Class name or list of class names of allowable classes. When selecting a list of objects, only instances of the classes specified in this property are selectable. Formula is set to the allowed-classes-list from the model-property-object.

- **Content**

  Holds a list of the object names in the selected-object-list.

- **Selected-object-list**

  This property refers initially to the value of the property instance specified by model-property-object. The value of this property is updated every time the user selects a new list of objects. It always holds the most recently selected object list.

- **Object-selection-method**

  Name of a user-defined method. This method is called on every object selected by the user and returns an object instance. The object instance returned by this method is considered the desired object. The user can define a method on an AML class and specify that method name here in order to allow the user to select an object but return another object. The method specified here must always return an object instance. See example in previous section - data model example.

  Default Formula: (the object-selection-method (:from ^model-property-object))

Note : The user should not redefine the formulas of button1-action and button3-action.

See Also: tree-widget-object

ui-object-selection-button-class [class]

This is a composite widget class constituted of an action button horizontally adjacent to a non-editable text field. The action button is on the left side. Its model-property-object, when specified, is typically an instance of type object-selection-property-class. This class is capable of creating new objects, selecting existing objects, and deleting a selected object. It is used for interfacing an object-selection-property-class instance and also for general-purpose object management. By clicking on the action button the user can create, select, delete or deselect an object. The available options are controlled by the properties allow-object-creation? and allow-object-deletion? (See below). If the user left-clicks on the button, object selection follows the global object selection mechanism (set from the AML menu bar). If the user right-clicks on the button, selection is via a tree menu. When the apply-action is triggered, the formula

of the property instance specified by model-property-object is changed to reference the instance specified by selected-object. If automatic-apply? is non-nil, the apply-action is triggered every time an object is selected. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil.

Inheritance: model-interface-widget, ui-grid-component-mixin, ui-action-field-pair-class

User Input Properties:

Properties:

- **Allow-object-creation?**

  When non-nil, the widget can handle creation of objects under parent-object. To create an object, the user clicks on the action button provided and selects "new" from the pop-up menu. If more than one class is specified in class-name, the user will be prompted to select a class.

  Default Formula: nil

- **Allow-object-deletion?**

  When non-nil, the widget will allow deleting the selected object. To delete the selected object, the user clicks on the action button provided and selects "delete" from the pop-up menu.

  Default Formula: nil

- **Object-name-prefix**

  Name prefix of objects to be created. When the property class-name is a list (see below), object-name-prefix can be a list of name prefixes with each prefix corresponding to a class name in the class-name list.

  Default Formula: (default (if (listp ^class-name) 'object ^class-name))

- **Parent-object**

  Object instance specifying the root of the tree of selectable objects. Also, when new objects are created, they are added as subobjects of the parent-object. It is recommended that parent-object be a name generator to avoid prompting the user for an object name every time a new object needs to be created.

  Default Formula: nil

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **Apply-action**

  Changes the value from the model-property-object to the instance specified by selected-object.

- **Cancel-action**

  Smashes the value of the selected-object property.

- **Class-name**

  Class name or list of class names of available classes. When creating an object, the object created is of the class specified by this property (If a list of classes is specified, a menu pops up prompting the user to select one of the available classes). When selecting an object, only instances of the classes specified in this property are selectable. The formula is set to the allowed-classes-list from the model-property-object.

- **Content**

  Holds the name of the selected object when it exists.

- **Selected-object**

  This property is initially set to the value of the property instance specified by model-property-object. It is updated every time an object is selected/accepted or a new object is created. It always holds the most recently selected/created object instance. See "Accepting a selected object" below.

- **Object-selection-method**

  Name of a user-defined method. This method is called on every object selected by the user and returns an object instance. The object instance returned by this method is considered the desired object. The user can define a method on an AML class and specify that method name here in order to allow the user to select an object but return another object. The method specified here must always return an object instance. See example in previous section - data model example.

  Default Formula: (the object-selection-method (:from ^model-property-object))

Note : The user should not redefine the formulas of button1-action and button3-action. **Usage and Behavior:** - Selecting an Object: First, the user selects an object. Second, the method allowed-selection? is called on the model-property-object and is passed the selected object. If allowed-selection? returns a non-nil value (it does by default), the object-selection-method is called on the selected object and the object instance returned by that method is considered to be the desired object. By default, the object-selection-method simply returns the object instance it was called on; therefore, the object selected by the user is returned. Finally, the desired object is accepted as follows: o The value of the property selected-object is changed to the new selected object. o The widget is updated. (!update?). o The method post-object-selection-method is called on the superior of the widget instance. - Deleting the Selected Object: When the user decides to delete the selected object, a menu pops up to confirm the decision. If confirmed, the object instance specified by the property selected-object is deleted as follows: o The method pre-object-deletion-method is called on the superior of the widget instance. o The selected-object is deleted. o The property selected-object is changed to nil. o The method post-object-deletion-method is called on the superior of the widget instance.

post-object-selection-method [Method]

Defined on Classes:

ui-root

This method is automatically called after an object is selected and accepted from a ui-object-selection-button-class widget. It is not called on the widget instance, but rather on the superior instance of the widget (i.e the form/sub-form/frame instance that the widget belongs to). The method is also passed a flag parameter; this parameter is non-nil when the object selected is a newly created object (i.e. the user created a new object instead of selecting an existing one). By default the method only contains one statement: !update?. The user can redefine this method in order to perform application specific tasks that are needed after an object is selected/created by the ui-object-selection-button-class instance. It should be redefined on the class of the superior instance of the widget.

Arguments:

- **Instance**

  Instance of ui-root. (I.e. inherits from ui-root)

- **New-object?**

  Flag parameter. When non-nil, specifies that the object selected is a newly created object. This parameter can be used for application specific behavior.

pre-object-deletion-method [Method]

Defined on Classes:

ui-root

This method is automatically called before an object is deleted by a ui-object-selection-button-class widget. It is not called on the widget instance, but rather on the superior instance of the widget (i.e the form/sub-form/frame instance that the widget belongs to). By default the method only contains one statement: !update?. The user can redefine this method in order to perform application specific tasks that are needed before an object deleted.

post-object-deletion-method [Method]

This method is automatically called after an object is deleted by a ui-object-selection-button-class widget. It is not called on the widget instance, but rather on the superior instance of the widget (i.e the form/sub-form/frame instance that the widget belongs to). The user can redefine this method in order to perform application specific tasks that are needed after an object deleted.

See Also: ui-grid-component-mixin, ui-action-field-pair-class, update, allowed-selection?

ui-option-property-menu-class [class]

This is a labeled option menu defined to communicate with option-property-class. Its model-property-object is typically an option-property-class instance. If automatic-apply? is non-nil, the apply-action is triggered every time a new option is selected. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil.

Inheritance: model-interface-widget, ui-grid-labeled-option-menu-class

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **Cancel-action**

  Smashes the value of the selected-option property so the widget resets its selected option to the model-property-object's value when the cancel-action is triggered. (Update must be called to reflect the reset).

- **Labels-list**

  The labels-list from the model-property-object. Equals nil when no labels-list is found.

- **Options-list**

  The options-list from the model-property-object. The value nil cannot be a member of the options-list.

- **Selected-option**

  The value from the model-property-object. Equals nil when no model-property-object is provided. The selected-option should always be equal to a member of the options-list.

ui-option-property-radio-buttons-class [class]

This is a labeled radio buttons group defined to communicate with option-property-class. Its model-property-object is typically an instance of option-property-class. If automatic-apply? is non-nil, the apply-action is triggered every time an new option from the radio group is clicked. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil.

Inheritance: model-interface-widget, ui-grid-component-mixin, ui-labeled-radio-buttons-class

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **Cancel-action**

  Smashes the value of the status property so the widget resets its selected option to the model-property-object's value when the cancel-action is triggered. (Update must be called to reflect the reset).

- **Labels-list**

  The labels-list from the model-property-object. Equals nil when no labels-list is found.

- **Options-list**

  The options-list from the model-property-object. The value nil cannot be a member of the options-list property.

- **Orient**

  Default Formula: :horizontal.

- **Selected-option**

  The value from the model-property-object. Equals nil when no model-property-object is provided. The selected-option should always be equal to a member of the options-list.

- **Status**

  This property specifies holds the index of the selected options.

  Default Formula: (position ^selected-option ^options-list)

- **Wrap-count**

  Default Formula: 1

ui-property-field-class [class]

This is a labeled field defined to communicate with editable-data-property-class. Its model-property-object is typically an editable-data-property-class instance. By default, the field background color is white. If automatic-apply? is non-nil, the apply-action is triggered every time either of the following is true: 1. The field loses focus (can also happen by hitting the tab key while the field is in focus). 2. The user hits return while the widget is in focus. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil.

Inheritance: model-interface-widget, ui-grid-labeled-field-class

Properties linked to AML model (Do not need to be set by the user)

Properties:

- **cancel-action**

Smashes the value of the content property, so the widget resets its content to the model-property-object's value when the cancel-action is triggered. (Update must be called to reflect the reset).

- **Content**

  The value from the model-property-object. Equals an empty string when no model-property-object is specified.

ui-property-selection-button-class [class]

This is a composite widget class constituted of an action button horizontally adjacent to a non-editable text field. The action button is on the left side. This widget inherits from ui-object-selection-button-class; however, it does not feature any of the object creation and deletion functionality. Its purpose is to interface an object-selection-property-class instance. Its model-property-object is typically an instance of type object-selection-property-class. The value of the property instance specified by model-property-object is always an object instance or a property value of a selected object instance. **Usage and Behavior:** - The user clicks on the button and selects an object. The method allowed-selection? is then called on the model-property-object and is passed the selected object. If allowed-selection? returns a non-nil value, the object-selection-method is called on the selected object and returns the desired object. By default the desired object is the selected object (Default object-selection-method returns the object itself). The desired object is then stored in the selected-object property. If the user left-clicks on the button, object selection follows the global object selection mechanism (set from the AML menu bar). If the user right-clicks on the button, selection is via a tree menu. Right clicking also allows deselecting the current object. - Selection-property-name specifies the name of a property of the selected object. The value of the property specified by selection-property-name is stored in the selected-value property. If the desired selected-value is the object itself, selection-property-name must be equal to 'self. If selection-property-name is equal to nil, a menu pops-up for the user to select a property from the selected object. - When the apply-action is triggered, the formula generated uses a standard "the" reference to the destination object/property instead of a "the-list" reference. If the "editable?" property of the widget is t, and the user enters a new value in the field, this new values is applied to the property when the widget's apply-action is triggered. The "the-reference" formula is not overwritten, only the value is changed using change-property-value. - If automatic-apply? is non-nil, the apply-action is triggered every time an object is selected. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil.

Inheritance: ui-object-selection-button-class

User Input Properties:

Properties:

- **tree-widget-object**

  This property should be equal to an instance of ui-model-tree. When the user is prompted by the application to interactively select an object or a property from the model tree, the window corresponding to the ui-model-tree given here is used for interactive selection (Left-click to select an object from the tree, right-click to abort). When this property is nil, a default model tree form pops-up for selection. Please refer to the method tree-widget-object.

  Default Formula: (when (the superior model-property-object) (tree-widget-object (the superior model-property-object)))

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **apply-action**

  Changes the formula of the corresponding model property to reference the selected object/property.

- **cancel-action**

  Smashes the value of the selected-object and selection-property-name properties to restore the widget status to equal the model property values.

- **content**

  Equals to the selected-value. If selected-value is an object instance, content is set to the object name of that instance.

- **selected-value**

  This is an output property that holds the value of the property specified by selection-property-name. Selected-value is equal to the selected-object if selection-property-name equal 'self or if selected-object is not an object instance.

- **selection-property-name**

  Name of the desired property from the selected-object. The default formula is set to the selection-property-name of the property instance specified by model-property-object. When no selection-property-name is found, the formula returns 'self.

See Also: tree-widget-object

ui-property-filtered-selection-button-class [class]

This class has the same functionality as ui-property-selection-button-class with one addition. This class has a cadidate-objects-list property.

Inheritance: ui-property-selection-button-class

User Input Properties:

Properties:

- **candidate-objects-list**

  When this property is nil, this class behaves exactly like the parent class. When this property contains a list of objects, only the objects in this list can be highlighted. When selecting an object graphically and the candidate-objects-list property contains a list of objects, only the objects in this list can be highlighted. Whe selecting an object from the tree, you will be able to select any object in the tree but an informal message will be displayed stating that the currently selected object is invalid and will state the list of valid class types and candidate objects that can be selected from.

  Default Formula: nil

tree-widget-object [Method]

Defined on Classes:

aml-class

The default behavior can be described as follows: A widget interfacing an AML property (like object-selection-property-class) needs a reference to a tree widget to allow interactive selection of objects and

properties from the tree. When a tree reference is needed by that widget, the system automatically calls the method tree-widget-object on the widget instance. This method calls tree-widget-object on the corresponding property object. If the call on the property object returns nil, which is the default behavior, it searches for a tree widget associated with the main form that the widget belongs to.

ui-text-view-button-class [class]

This is an action button typically representing a text-view-property-class. Its model-property-object points to an instance of text-view-property-class. It opens a text editor for viewing the content of the text string.

Inheritance: model-interface-widget, ui-grid-action-button-class

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **Apply-action**

  Default Formula: nil

- **Button1-action**

  Opens a form to view the text string. This formula should not be redefined by the user.

- **Cancel-action**

  Default Formula: nil

- **Text**

  Text string. Holds the value from model-property-object.

Note: The property automatic-apply? is ignored for this class.

ui-toggle-list-property-button-class [class]

This is a ui-toggle-group-class defined to communicate with flag-list-property-class. Its model-property-object is typically a flag-list-property-class instance. If automatic-apply? is non-nil, the apply-action is triggered every time a button is toggled on/off. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil.

Inheritance: model-interface-widget, ui-grid-toggle-group-class

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **Cancel-action**

  Smashes the value of the status property, so the widget resets its status to the model-property-object's value when the cancel-action is triggered. (Update must be called to reflect the reset).

- **Labels-list**

  The labels-list from the model-property-object. Equals nil when no labels-list is found.

- **Status:**

  The value from the model-property-object (expected to be a list of t/nil flags). Equals nil when no model-property-object is specified.

ui-toggle-property-button-class [class]

This is a ui-toggle-button-class defined to communicate with flag-property-class. Its model-property-object is typically a flag-property-class instance. If automatic-apply? is non-nil, the apply-action is triggered every time the button is toggled on/off. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil. If the button is checked "on" and the original value of the property is not nil, the property value stays as is (i.e. non nil).

Inheritance: model-interface-widget, ui-grid-toggle-button-class

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **cancel-action**

  Smashes the value of status, so the widget resets its status to the model-property-object's value when the cancel-action is triggered. (Update must be called to reflect the reset).

- **status**

  The value from the model-property-object. It is equal to nil when no model-property-object is specified.

ui-value-list-property-form-class [class]

This form class inherits from ui-multiple-value-selection-form-class and is used to interface a model property of class value-list-property-class.

Inheritance: ui-multiple-value-selection-form-class

Input Properties:

Properties:

- **Model-property-object**

  Instance of value-list-property-class.

Derived Properties (These properties reference their corresponding properties in the model-property-object specified)

Properties:

- **available-labels-list, available-values-list, allow-duplicates?, close-button?, left-column-label, right-column-label**

Ouput Property

Properties:

- **selected-values-list**

  This property is initially derived from the model-property-object specified. It is then updated as the user selects/unselects values.

ui-var-unit-property-field-class [class]

This is a unit conversion field defined to communicate with var-unit-data-property-class. Its model-property-object is typically a var-unit-data-property-class instance. This widget can also handle the unit conversion locally in case the model-property-object is not of type var-unit-data-property-class. Unit

expressions and unit conversions are based on the unit table/functions described in chapter 14 of the AML reference manual. If automatic-apply? is non-nil, the apply-action is triggered when either of the following is true: The field looses focus (hitting the tab key while the widget is in focus works as well). The user hits the return key while the field is in focus. As of AML3.1.3, hitting the return key only works on non-WINDOWS platforms. Unless the application requires it, it is recommended that automatic-apply? keeps its default value of nil.

Inheritance: model-interface-widget, ui-grid-component-mixin, ui-unit-conversion-field-class

Input Properties Derived from Model: (Do not need to be set by the user)

Properties:

- **Apply-action**

  If convert-in-model? is non-nil, apply-action changes the values from the model-property-object's unit-in and value-in to the current unit-in and value-in respectively. Otherwise, apply-action changes the value from the model-property-object to the locally converted value.

- **Available-units-list**

  Holds the available-units-list from the model-property-object when convert-in-model? is non-nil; otherwise, returns the available dimensionally similar units to the specified unit.

- **Cancel-action**

  Smashes the value of the content property.

- **Content**

  Holds the value-in from the model-property-object when convert-in-model? is non-nil; otherwise, holds the value of the property instance specified by model-property-object. When no model-property-object is specified, content is 0.

- **Convert-in-model?**

  Specifies whether the unit conversion is performed in the data model or locally within the widget instance. When non-nil, the unit conversion is performed in the data model. The formula is set to return a non-nil value whenever the model-property-object is of type var-unit-data-property-class; therefore, when the model-property-object is an instance of var-unit-data-property-class, the conversion is performed in the data model (This is recommended.)

- **Unit**

  Holds the unit value from the model-property-object. Equals nil when no unit is found.

- **Unit-in**

  Holds the unit-in value from the model-property-object. Equals nil when no unit-in is found.

- **Units-labels-list**

  Holds the units-labels-list from the model-property-object when convert-in-model? is non-nil; otherwise, converts the local available-units-list to a list of strings.

Redefined inherited properties:

Properties:

- **Background-color**

  Default Formula: "white"

- **Field-width**

  Default Formula: (- 100 30 ^label-width).

- **Label-background-color**

  Default Formula: (the superior superior background-color (:error "gray83")).

Model Interface Widget - Groups

This section illustrates a set of advanced GUI model classes designed to interface a group of data-model properties; these classes are referred to as model interface widget-groups. They automatically link their component widgets (subobjects) to the data-model based on a list of model-interface property instances provided. Unless otherwise specified, components of a model interface widget group are instances of the model-interface-widget classes described in section 3 and are automatically generated and linked to their corresponding model-property-object.

A model interface widget group constitutes the GUI of a subset of the data-model. This section also introduces the super-classes of model interface widget-groups. These super-classes are forms and subforms that define a set of properties capable of achieving a consistent look and functionality in AML application GUI.

A. Super Classes

The purpose of the super-classes and properties described below is standardizing sizes, positions and functionality of some GUI components. Model Interface Widget-groups inherit from these super-classes; the user is also encouraged to inherit from them when defining his/her own form/sub-form class in order to take advantage of their capabilities.

pixel-unit-height, pixel-unit-width (Properties)

The properties pixel-unit-height and pixel-unit-width are described here: Their purpose is to standardize the size of widgets throughout the GUI of an AML application.

As a reminder, if a form class inherits from ui-grid-form-mixin, it expects its widget components to inherit from ui-grid-component-mixin. Any component that inherits from ui-grid-form-mixin derives its x-offset, y-offset, width and height from:

1. its own size and coordinates properties and

2. its superior's unit-height and unit-width

The size and coordinates components are factors of the unit-height and unit-width, refer to the "Grid form and components" section of this manual. The unit-height and unit-width are the properties that drive the size of component widgets. A common setback to the unit-height and unit-width properties is that they can represent either a pixel value or a percentage value depending on the measurement property. Therefore it is hard to manage a common unit value throughout the GUI of the application, especially if the user wants to use percentage measurement for some forms and pixel measurement for others.

Pixel-unit-height and pixel-unit-width allow the user to specify a unit value in pixels for the height and width respectively. Then, depending on the measurement property value, the value for unit-height and unit-width are automatically derived as follows:

1. The unit-height is equal to the pixel-unit-height if measurement is 'pixels or equal to 100 * (pixel-unit-height / height of the form in pixels) if measurement is 'percentage.

2. The unit-width is equal to the pixel-unit-width if measurement is 'pixels or equal to 100 * (pixel-unit-width / width of the form in pixels) if measurement is 'percentage.

Note :

1. The actual "driving" unit of ui-grid-form-mixin form/subform is still the unit-height and unit-width; pixel-unit-height/pixel-unit-width is only an input used to derive unit-height/unit-width.

2. When using pixel-unit-height/pixel-unit-width along with percentage measurement, the unit-height/unit-width derived does not exactly reflect the desired pixel values unless the pixel-unit-height/pixel-unit-width represents an integer percentage of its superior form's height/width in pixels. This is due to the fact that unit-height/unit-width has to be truncated to an integer. For example: If a form's height is equivalent to 500 pixels and percentage measurement is used, a pixel-unit-height of 25 is a good value since it is exactly 5 percent of 500. However, a pixel-unit-height of 24 is not recommended because it represents 4.x percent of 500 and would be truncated to 4.

The classes ui-standard-form-superclass and ui-standard-subform-superclass are described below; they define the pixel-unit-height and pixel-unit-width properties. The default formulas of pixel-unit-height and pixel-unit-width call the function default; therefore, it is possible for the user to specify them only at the top level of the GUI tree and their value will propagate to all pixel-unit-height and pixel-unit-width properties within that GUI tree.

ui-standard-form-superclass [class]

Inheritance: ui-form-class, ui-grid-form-mixin

User Input Properties:

Properties:

- **pixel-unit-height**

  Default Formula: (default 26) for UNIX/motif platforms and (default 30) for WINDOWS platforms.

- **pixel-unit-width**

  Default Formula: (default 70)

Redefined Inherited Properties

Properties:

- **height**

  Default Formula: 270 for UNIX/motif platforms and 252 for WINDOWS platforms.

- **Unit-height**

  Derived from pixel-unit-height.

- **unit-width**

  Derived from pixel-unit-width.

- **width**

  Default Formula: 300

See Also: pixel-unit-height and pixel-unit-width

ui-standard-subform-superclass [class]

Inheritance: Ui-grid-component-mixin, ui-subform-class, ui-grid-form-mixin

User Input Properties:

Properties:

- **pixel-unit-height**

  Default Formula: (default 26) for UNIX/motif platforms and (default 30) for WINDOWS platforms.

- **pixel-unit-width**

  Default Formula: (default 70)

Properties linked to AML model (Do not need to be set by the user)

Properties:

- **height**

  Default Formula: 270 for UNIX/motif platforms and 252 for WINDOWS platforms.

- **measurement**

  Default Formula: (default 'pixels)

- **unit-height**

  Derived from pixel-unit-height.

- **unit-width**

  Derived from pixel-unit-width.

- **width**

  Default Formula: 300

See Also: pixel-unit-height and pixel-unit-width

ui-apply-cancel-close-form-superclass [class]

This is a ui-standard-form-superclass that contains three buttons: A ui-apply-button-class, a ui-cancel-button-class, and a close button that hides the form. The buttons are positioned horizontally on the bottom left corner of the form in the following order: apply, cancel, close.

Inheritance: ui-standard-form-superclass

Properties:

- **Apply**

  A ui-apply-button-class instance whose height is equal to the form's unit-height and width to the form's unit-width. As expected, this button calls apply-form-method on the form when clicked.

- **Cancel**

  A ui-cancel-button-class instance whose height is equal to the form's unit-height and width to the form's unit-width. As expected, this button calls cancel-form-method on the form when clicked. Cancel-form-method is called with the keyword update-form? set to t in order to update the form.

- **Close**

A ui-action-button-class instance whose height is equal to the form's unit-height and width to the form's unit-width. The button1-action of this button is set to close its parent form.

The apply-form-method method is redefined on ui-apply-cancel-close-form-superclass as follows:

```
(define-method apply-form-method ui-apply-cancel-close-form-superclass ()
        (call-next-method)
        (post-apply-method self))
```

The cancel-form-method method is redefined on ui-apply-cancel-close-form-superclass as follows:

```
(define-method cancel-form-method ui-apply-cancel-close-form-superclass (&key
(close? nil)

(update-form? nil))
        (call-next-method)
        (post-cancel-method self)
        )
```

post-apply-method [Method]

Defined on Classes:

ui-apply-cancel-close-form-superclass

This method does not perform any action by default. It can be redefined by the user to perform desired actions after the form values have been "applied". See apply-form-method in the description of the class above.

post-cancel-method [Method]

Defined on Classes:

ui-apply-cancel-close-form-superclass

This method does not perform any action by default. It can be redefined by the user to perform desired actions after the form values have been "cancelled". See cancel-form-method in the description of the class above.

B. Group Widget Classes

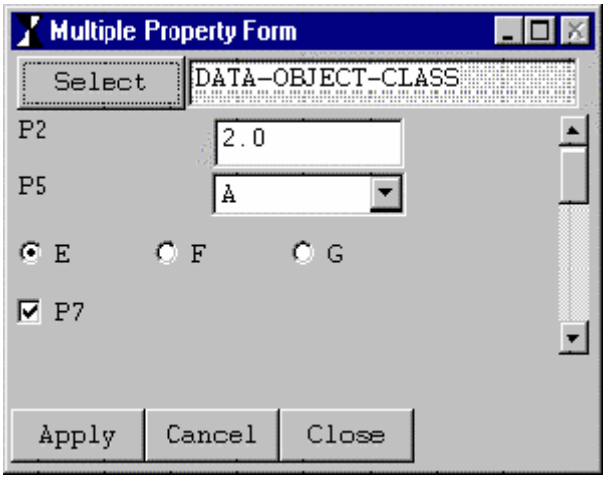
The following is a description of the model interface widget-group classes and their corresponding methods. As mentioned above, a model interface widget group manages a list of data-model properties and represents a GUI for a subset of the data-model. The classes are listed in alphabetical order. All classes below can be instantiated by the user.

ui-multiple-property-form-class [class]

This class is a ui-apply-cancel-close-form-superclass containing an object selection button and a property editing area. Its purpose is to provide an input/output GUI for a set of properties from the data-model. These properties are determined by the property-objects-list. The property-objects-list must be

set by the developer and is typically derived from the current-object instance. 1. The Current-object is controlled by the object selection button; the current-object is typically an object instance of the data-model. The object selection button is also capable of creating a new object instance to be the current-object. It can also delete the selected current-object. 2. The property editing area is a scrolled area containing an instance of ui-multiple-property-subform-class. It is an interface to the property instances specified in the property-objects-list. It automatically creates model-interface-widget instances corresponding to the property-objects-list instances provided. Refer to ui-multiple-property-subform-class for details.

Figure 17-1. Instance of ui-multiple-property-form-class.



The scrollable area between the select button on top and the apply button at the bottom is the property editing Area. It is actually an instance of ui-multiple-property-subform-class. The property-objects-list property corresponding to this figure is a list of 4 property instances.of class (from top to bottom): editable-data-property-class, option-property-class in menu mode, option-property-class in radio mode, and flag-property-class. 3. The apply, cancel and close button are inherited from ui-apply-cancel-close-form-superclass. Note : This form is designed to be instantiated as a top-level form, i.e. not as a sub-object of another form.

Inheritance: ui-apply-cancel-close-form-superclass

User Input Properties:

Properties:

- **Allow-object-creation?**

  Flag, when non-nil, specifies that the selection button will allow the creation of new object instances to be the current-object.

- **Allow-object-deletion?**

  Flag, when non-nil, specifies that the object selection button will have the option of deleting the current-object.

- **Current-object**

  Object instance being edited. The object selection button updates this instance every time it selects or creates an object. This property always holds the instance of the most recently selected/created object.

Default Formula: nil

- **Current-object-class**

  Class name or list of class names of allowable classes for the current-object.

  Default Formula: object-root-class

- **Label-area-width**

  Integer specifying a percentage of the width of the widgets that the label section occupies. This is reflected in the automatically created widgets of the ui-multiple-property-subform-class instance contained in the scroll-window subobject.

- **Parent-object**

  Object instance specifying the root of the tree of selectable objects.

  Default Formula: (root-object)

- **Property-objects-list**

  List of model-interface property instances to be managed by the property editing area. The default formula returns all the model-interface-property-class instances of the current-object. If the user wishes to set this list, he/she should typically derive it from the current-object instance. Refer to ui-multiple-property-subform-class to learn the enhanced capabilities property-objects-list and see an example.

- **Row-clearance**

  Refer to ui-multiple-property-subform-class.

  Default Formula: 5

- **Select-object-label**

  String specifying the label of the object selection button.

  Default Formula: "Select Object"

- **Work-area-pixel-unit-width**

  Specifies the pixel-unit-width value of the ui-multiple-property-subform-class instance contained in the scroll-window subobject. Refer to pixel-unit-width and ui-multiple-property-subform-class.

- **Work-area-range**

  Width and height (list width height) in pixels of the ui-multiple-property-subform-class instance contained in the scroll-window sub-object.

  Default Formula: '(200 400)

Redefined Inherited Properties

Properties:

- **Label:**

  Default Formula: "Edit Properties"

Subobjects :

Properties:

- **Scroll-window**

Instance of ui-multiple-property-scrolled-window. This is the property editing area scrolled-window mentioned above. It contains an instance of ui-multiple-property-subform-class.

- **Select-object**

  Instance of ui-object-selection-button-class whose model-property-object is nil. This is the object selection button described above.

See Also: ui-object-selection-button-class, ui-multiple-property-subform-class

ui-multiple-property-list-class [class]

This class allows the editing of a list of property object instances. It includes a list widget that displays the property labels/names and values, as well as a text editing area that allows changing the value of the selected property from the list. For efficiency purposes, it is recommended to use this class instead of a ui-multiple-property-subform-class when dealing with a list of simple editable input properties from the data-model. When an instance in the list does not have a label, the object name is displayed instead.

Inheritance: ui-standard-subform-superclass

User Input Properties:

Properties:

- **Demand-property-values?**

  When nil, does not demand the value of the unbound properties in the property-objects-list.

  Default Formula: t

- **Property-objects-list**

  List of property-object instances. (Does not allow the enhanced format of the property-objects-list of ui-multiple-property-subform-class). This list can also contain object instances that do not inherit from property-class.

  Default Formula: nil

- **Type1-Format**

  Format used by the instances in property-objects-list that inherit from property-class and do not contain properties themselves. Specifies the display format of the property name and its value in the list widget. Refer to the function format in the AML reference manual.

  Default Formula: "~20a ~20a"

- **Type2-Format**

  Format used by the instances in property-objects-list that inherit from property-class and contain other properties themselves. Specifies the display format of the property name and its value in the list widget. Refer to the function format in the AML reference manual.

  Default Formula: "~19a > ~20a"

- **Type3-Format**

  Format used by the instances in property-objects-list that do not inherit from property-class. Specifies the display format of the object name and its instance value in the list widget. Refer to the function format in the AML reference manual.

  Default Formula: "~18a >> ~20a"

Output Properties

Properties:

- **Selected-index**

  Holds the index (integer that starts at 0) of the selected item of the list widget. The user should not change the formula of this property.

- **Selected-property-object**

  Holds the instance of the selected item of the list widget. The user should not change the formula of this property.

ui-multiple-property-scrolled-window [class]

This is a scrolled window containing a ui-multiple-property-subform-class subobject. It is used when a ui-multiple-property-subform-class is needed and not enough space is available on the parent form to accommodate it.

Inheritance: ui-scrolled-window-class

User Input Properties:

Properties:

- **Label-area-width**

  Determines the label-area-width property value of subform.

  Default Formula: 50

- **Parent-object**

  Determines the parent-object property value of subform.

  Default Formula: (default (root-object))

- **Pixel-unit-height**

  Determines the pixel-unit-height property value of subform.

  Default Formula: (default 26) for UNIX/Motif platforms, and (default 30) for WINDOWS platforms.

- **Pixel-unit-width**

  Determines the pixel-unit-width property value of subform.

  Default Formula: (default (first ^work-area-range))

- **Property-objects-list**

  Determines the property-objects-list property of subform. Refer to the property-objects-list and example of ui-multiple-property-subform-class.

  Default Formula: (Default nil)

- **Row-clearance**

  Determines the row-clearance property value of subform.

  Default Formula: 5

- **Work-area-range.**

Width and height (list width height) in pixels of subform.

Default Formula: '(200 300)

Redefined Inherited Properties

Properties:

- **Apply-action**

  Default Formula: Formula is set to '(apply-form-method (the superior)) to link the apply mechanism to the subform.

- **Cancel-action**

  Default Formula: Formula is set to '(cancel-form-method (the superior) :update-form? t) to link the cancel mechanism to the subform and update at the end.

Subobjects:

Properties:

- **Subform**

  Instance of ui-multiple-property-subform-class.

See Also: ui-multiple-property-subform-class

ui-multiple-property-subform-class [class]

This class is a ui-standard-subform-superclass that controls the automatic creation of model-interface-widget instances based on a list of property instances provided. The property instances typically inherit from model-interface-property-class (Section 2) and are determined by the property-objects-list property. A widget instance is created for every property instance. A property instance "knows" what type of model-interface-widget class it needs in order to graphically interface itself to the user: This is achieved thanks to the method ui-widget-class-name. Ui-widget-class-name is defined on property-object and on every model-interface property class to return the name of the model-interface-widget corresponding to it. A multiple-property subform calls the method ui-widget-class-name on the property instances of property-objects-list to learn the classes of widgets it needs to generate. The different ui-widget-class-name methods are defined in Section 2. Widgets are generated as sub-objects of the multiple-property subform instance using series-object capabilities. By default they are created vertically from top to button in a single column, their size property is set to '(1 1) so their width and height are equal to unit-height and unit-width respectively. As described previously, unit-height and unit-width are derived from pixel-unit-height and pixel-unit-width.

Inheritance: ui-standard-subform-superclass, series-object

User Input Properties:

Properties:

- **Label-area-width**

  Determines the label-area-width property value of the automatically created widgets. The label-area-width determines a percentage of the width of a widget that the label section occupies. If a widget does not have a label-area-width, this property is ignored.

  Default Formula: 50

- **Parent-object**

Determines the parent-object property value of the automatically created widgets that can take a parent-object property, for example: parent-object is a property of ui-object-selection-button-class and ui-property-selection-button-class. Refer to Section 3 above.

Default Formula: (root-object)

- **Property-objects-list**

  Main input property. It determines the property instances to be interfaced as well as desired redefinition of the default formulas of the automatically created widgets. Refer to "Property-objects-list format" next.

  Default Formula: nil

- **Row-clearance**

  Percentage value representing the desired vertical clearance between widgets. It is a percentage of unit-height.

  Default Formula: 5

Redefined Inherited Properties

Properties:

- **Apply-action**

  The default formula is set to link the apply mechanism to the sub-objects (i.e. to the widgets).

  Default Formula: '(apply-form-method (the superior))

- **Cancel-action**

  The default formula is used to link the cancel mechanism to the sub-objects (i.e. to the widgets) and update them at the end.

  Default Formula: '(cancel-form-method (the superior) :update-form? t)

- **Height**

  Default Formula: Formula is set to fit all the property widgets on the form.

- **Measurement**

  Default Formula: 'pixels (Recommended)

- **Pixel-unit-width**

  Default Formula: (default 150)

- **Width**

  Default Formula: ^pixel-unit-width

Property-objects-list format: Property-objects-list is a list whose elements can be: 1. A property object instance or 2. :blank to specify a blank widget (null-object) or 3. A string to specify a label widget The model-interface-widget instances are generated vertically from top to bottom in the order dictated by property-objects-list. Their size property is set to '(1 1) by default and their model-property-object is derived from the property-objects-list sequentially. Any property formula of a model-interface-widget instance generated can be redefined by property-objects-list to provide the user with the same flexibility that comes with manually building a form that groups multiple widgets. Therefore, an element of property-objects-list can also be a pair whose first element is any of the 3 allowed type listed above and

the second element a list of property formula specification to overwrite the default formulas of the corresponding widget. Refer to the ui-widget-class-name method defined with property class in section 1 to know which class of widget is created for a particular property class.

Example 17-9. Ui-multiple-property-subform-class example

Let's assume a ui-multiple-property-subform-class is used to edit the properties of an instance of the following class:

```
(define-class box-cylinder-union
        :inherit-from(union-object)
        :properties(
                (width :class 'editable-data-property-class
                        formula 5.0)
                (height :class 'editable-data-property-class
                        formula 5.0)
                (depth : class 'editable-data-property-class
                        formula 5.0)
                (diameter :class 'editable-data-property-class
                        formula 2.0)
                (cylinder-height :class 'editable-data-property-class
                        label "Height"
                        formula 10.0)
                (simplify? :class 'flag-property-class
                        formula t) ;; simplify the union geom
                object-list (list ^box ^cylinder)
                )
        :subobjects(
                (box :class 'box-object ...etc)
                (cylinder :class 'cylinder-object ...etc)
                )
  )
```

The subform that manages an instance of the above class is defined as follows:

```
(define-class box-cylinder-union-subform
        :inherit-from(ui-multiple-property-subform-class)
        :properties(
                object-to-edit (root-object)
                ;; assuming the current model is an instance of box-cylinder-
union
                property-objects-list
                (trace-from ^object-to-edit
                        (list
                        '("Box" (foreground-color "blue" font "8x13Bold"))
                        (the width self)
                        (the height self)
                        (the depth self)
                        :blank
                        '("Cylinder" (foreground-color "blue" font
"8x13Bold"))
```

```
                        (the cylinder-height self)
                        (the diameter self)
                        '(:blank (size '(2 1))) ;; to allow 2 unit blank rows
                        '("Union" (foreground-color "blue" font "8x13Bold"))
                        (list (the simplify? self) '(label-align :center))
                        ))
        )
    )
```

Application Classes and Methods

A. Data Model Node

A data model node is defined as an object instance that is a part of a hierarchy of objects forming the data-model.

data-model-node-mixin [class]

This class is designed to represent a node of a data model.tree. It contains application-specific knowledge about the node it represents. It also manages the properties of that node that are part of the data-model and that should be a part of the application GUI. Data-model-node-mixin is typically inherited into user defined classes. Standard data model application forms (described later) make use of data model nodes for the automatic generation of model GUIs.

Inheritance: object

Properties:

- **Available?**

  When nil, does not allow GUI access to the property-object-list of the node from the data model tree.

  Default Formula: t

- **Label**

  Label of the node as displayed in the GUI data model tree.

- **Property-objects-list**

  List of model-interface-property-class instances that define the data-model properties of this node. Properties included in this list will be available in the automatically generated forms that use data-model-node-mixin. The list follows the same format of property-objects-list described with ui-multiple-property-subform-class(link at "See Also:"). All labels are "vertically center-justified" and can accept multiple lines.

- **Subnode-object-list**

  Subobject instances of this node that are automatically demanded when the data model GUI tree is displayed.

  Default Formula: (children (the superior) :class 'data-model-node-mixin).

- **Subobject-class-name**

This property specifies the object class (es) that will be available when adding a subobject to this node from a data model GUI. . It accepts a class name symbol, a list of class names to allow more than 1 class or a list of '(class-label class-name) pairs. Example: '(box-object cylinder-object) or '(("Box" 'box-object) ("Cylinder" 'cylinder-object).

Default Formula: 'object

- **Subobject-name-prefix**

  Prefix (symbol) of sub-objects that are added from the data model GUI. Can also be a list of prefixes when the subobject-class-name is a list.

  Default Formula: 'subobject

- **User-added-subobjects?**

  When non-nil, allows runtime addition of sub-objects to the node from the data model GUI.

  Default Formula: nil

- **User-deletable?**

  When nil, does not allow deletion of the node from the data-model GUI.

  Default Formula: t

- **User-form-class**

  Class name of a user-defined form. When a data-model form is used, and if that form supports a user-defined area, an instance of the form class specified here will be displayed in the user-defined area every time the data-model node is opened for editing. Forms that support user-defined areas are: data-model-tree-inspection-class and data-model-main-form-class. See example 2 below.

See Also: ui-data-model-tree-inspection-class, ui-multiple-property-subform-class, ui-data-model-main-form-class

Example 17-10. Data-model-node-mixin example 1

Below is the class definition of a box, a cylinder and their union. They can be instantiated as a data-model that one of the application forms defined later can interface. The table-values property of the union-data-model-class is for example purposes only and does not add any functionality to the union.

```
(define-class box-data-model-class
  :inherit-from(box-object data-model-node-mixin)
  :properties(
              property-objects-list (trace-from
                                     (the superior)
                                     (list (list (the depth self)
                                                 '(automatic-apply? t))
                                           (the height self)(the width
self)))
              )
  )

(define-class cylinder-data-model-class
  :inherit-from(cylinder-object data-model-node-mixin)
  :properties(
              property-objects-list (trace-from
                                     (the superior)
```

```
                                    (list (the diameter self)
                                          (the height self))
                                    )
                diameter 1.0
                height   8.0
                )
  )

(define-class union-data-model-class
  :inherit-from(union-object data-model-node-mixin)
  :properties(
                label "Union of Box and Cylinder"
                user-added-subobjects? t
                property-objects-list (trace-from
                                    (the superior)
                                    (list (the simplify? self)
                                          (the render self)
                                          (the table-values self)))
                (simplify? :class 'flag-property-class
                           formula t
                           )
                (render :class 'option-property-class
                        options-list '(boundary shaded isoline)
                        mode 'radio
                        formula 'boundary
                        )
                (table-values :class 'data-matrix-property-class
                              label "Table"
                              formula '( (1 2 3) (a b c))
                              rows-labels-list '("row1" "row2")
                              columns-labels-list '("column1" "column2")
                              mode 'pop-up
                              )
                subobject-class-name 'union-data-model-class
                object-list (list ^box ^cylinder)
                )
  :subobjects(
                (cylinder :class 'cylinder-data-model-class
                          label "Cylinder node"
                          user-deletable? T
                          )
                (box :class 'box-data-model-class
                     label "Box Node"
                     user-deletable? T
                     )
                )
  )
```

Example 17-11. Data-model-node-mixin example 2

A user-defined form can be associated with a data model node. When the user clicks on a node (object) of a data-model form tree, the form associated with that object is displayed in the user-defined area. The

user-defined area is typically a part of the work area of a data-model form and is enabled by setting the property user-defined-area? (of the data-model form) to t. In order to associate a data model node with a user-defined form, the user only needs to set the property user-form-class of the data model node to the class name of the desired form. It must a defined class that inherits from ui-group.

Next is a systematic example on user-form-class: A form that is intended to be used in the user-defined-area of a data model form usually needs a reference to the selected object of the work area. The form also typically needs a reference to the data model form for potential updating/refreshing functionality.

First let's define a function that will be used any time the data model form instance is needed. This function is needed in order to abstract the location of the data model form from the user.

```
(defun get-data-model-form ( )
        ;; This will reference the location where the data model form is
instantiated
        ;; (May vary with applications)
        (the interface form data-model-main-form (:error nil))
        )
```

Then, let's define a generic user-defined-area form that all user-defined-area forms will inherit from:

```
(define-class ui-user-defined-area-form-mixin
        :inherit-from(ui-standard-subform-superclass)
        :properties(
                pixel-unit-width (default 60)
                pixel-unit-height (default 26)
                work-area-selected-object (default nil)
                main-form-object (get-data-model-form)
                )
        )
```

The property work-area-selected-object will hold the object instance of the current object being edited because of the following:

1. The formula of work-area-selected-object calls the function default

2. The user-defined-area form is always instantiated under the data model form object

Therefore, the work-area-selected-object property will find the work-area-selected-object property of the data model form and will hold its value. This guarantees that every time the property work-area-selected-object of the user-defined-area form is accessed, it will return the object instance being edited.

Next is the definition of a user form that contains a button that calls the method sample-method on the object being edited in the work area of the data-model form. The method then refreshes that work area.

```
(define-class ui-draw-object-form-class
        :inherit-from(ui-user-defined-area-form-mixin)
        :properties(
                pixel-unit-width 60
                measurement 'pixels
                height (* 10 ^pixel-unit-height)
                width (* 4 ^pixel-unit-width)
```

```
                )
        :subobjects(
        (button1 :class 'ui-grid-action-button-class
                coordinates '(1 0.5)
                size '(1 1)
                label "Run Sample Method"
                button1-action '(let ((obj ^^work-area-selected-object)
                                      )
                          (when (and obj (not (object-has-been-deleted? obj)))
                                  (sample-method obj)
                                  (when ^^main-form (update-work-area ^^main-
form))))
                        )
                )
   )
```

Finally, a data model object class is associated with the ui-draw-object-form-class as follows:

```
(define-class sample-data-model-object-class
        :inherit-from(data-model-node-mixin)
        :properties(
                property-objects-list ...etc
                user-form-class 'ui-draw-object-form-class
                )
        )
```

work-area-superior-object [Method]

Defined on Classes:

data-model-node-mixin

This method is automatically called when the "superior" button of a data-model's work area toolbar is left-clicked (up and down arrow bitmap). This method is used to specify/customize what the parent instance of the object being edited is, from the work-area's point of view. This method must always return an object instance.

work-area-children [Method]

Defined on Classes:

data-model-node-mixin

This method is called when the "subobject" button of a data-model's work area toolbar is right-clicked (up and down arrow bitmap). This method is used to define/customize the child instances of the object being edited, from the work-area's point of view. It must always return a list of object instances or just return nil.

B. Application Forms

The non-documented properties and sub-objects of the forms described in this section must not be accessed by AML users. This will ensure proper functionality and will not to jeopardize compatibility with future version.

ui-data-model-main-form-class [class]

A data model main form is a standard application form designed to interface a data-model that typically consists of a hierarchy of data-model-node-mixin instances. This class inherits from ui-data-model-tree-inspection-class (described later) and adds to it a canvas, a graphics control form, a toolbar, and a message pane. An instance of this form class is accessible from the AML menubar: Layout->System Layouts->AML Main Modeling Form.

Inheritance: ui-data-model-tree-inspection-class

Properties:

- **User-defined-canvas-actions-list**

  List of user defined actions that will be available on the toolbar below the canvas. It is a list of (action label image) lists where action-method is a method defined on the form class (refer to the example of user-defined-actions-list), label is the label string and image is the full path name of a bitmap file (bmp on WINDOWS, xbm/xpm on UNIX) that will appear on the corresponding toolbar button.

See Also: ui-data-model-tree-inspection-class, data-model-node-mixin

Example 17-12. Ui-data-model-main-form-class Example

Using the example shown with data-model-node-mixin above, here is an example of instantiating a ui-data-model-main-form-class to interface the union of the box and the cylinder.

```
(define-class application-main-form-class
        :inherit-from(ui-data-model-main-form-class)
        ;; Keep the default formulas of the input properties
        )

;; Instantiation
(add-object (the interface forms) 'application-main-form 'application-main-
form-class)
(create-model 'union-data-model-class)
(change-value
                (The interface forms application-main-form data-model-object)
                (root-object))
(display (The interface forms application-main-form))
```

select-canvas [Method]

Defined on Classes:

ui-data-model-main-form-class

This method returns the instance of the ui-canvas-class contained in the ui-data-model-main-form-class instance in question.

ui-data-model-tree-inspection-class [class]

A ui-data-model-tree-inspection-class is a standard application form designed to interface a data-model that typically consists of a hierarchy of data-model-node-mixin instances. The form points to a root object which is an instance specifying the root of the desired object tree. It can function in two modes: all-objects or data-model. In data-model mode, the form interfaces data-model-node-mixin instances only and captures the knowledge built into these nodes. In all-object mode, all objects under the root specified are interfaced. The form is divided into a tree area and a work area. The user has control over the size of these areas. Their size can be manipulated for the purpose of customizing the form by adding other widgets to it.

Inheritance: ui-standard-form-superclass

Properties:

- **Data-model-object**

  Root instance to interface. This instance is the root of the tree in the tree area.

  Default Formula: root-object

- **Maximum-size**

  The property specifies the maximum number of data model node types/classes expected in the tree in order to pre-allocate the necessary memory. If more is needed during a session, the necessary memory is dynamically allocated at runtime.

  Default Formula: 200

- **Menubar-class**

  When a valid class name that inherits from ui-menubar-class is specified here, an instance of that class is instantiated as a menubar for the form.

  Default Formula: nil

- **Mode**

  Specifies the mode of the tree area.

  Default Formula: ' data- model. Can also be set to 'all.

- **Name-generator**

  Instance of name-generator used to automatically generate the names of objects added from the work area or the tree area.

  Default Formula: nil to prompt the user for names

- **Pre-defined-actions-list**

  List of AML pre-defined actions that will be available on the tree actions menu. It is a list of couples where each couple contains an action and a label string. If an action X is specified, it is expected that a method named X-method be defined on the class ui-data-model-tree-inspection-class (or on a subclass of it). The label associated with an action will appear on the tree actions menu. The actions appearing in the formula above already have corresponding methods pre-defined in the system. While customizing an application, the developer can decide to change the formula of pre-defined-actions-list by removing one or more action entries. See also the property user-defined-actions-list below.

  Default Formula:

```
        '((edit-node "Edit")
        (inspect-node "Inspect")
        (expand-node "Expand")
        (add-subobject "Add")
        (draw-node "Draw")
        (shade-node "Shade")
        (delete-node "Delete")
        (undraw-node "Undraw")
        (unshade-node "Unshade")))
```

- **Subform-init-form**

  List of formula specifications used to change the default property formulas of the work area sub-forms. The formulas specified overwrite the default formula of the sub-form class. Example: (background-color "blue" font "10x20)

- **Tree-area-width**

  Width of the tree area. This value is affected by the measurement property of the form. It is a fixed pixel value when measurement is 'pixels and a percentage of the width of the form when 'measurement is 'percentage

  Default Formula: 44

- **User-defined-actions-list**

  List of user defined actions that will be available on the tree actions menu. These actions will be added to the pre-defined actions. This list expects the same format followed by pre-defined-actions-list (above). It expects a list of (action label-string) pairs where action is the prefix of a method named action-method defined on the form class and label-string is the representation of the action on the tree actions menu. Example:

```
(define-class application-form
:inherit-from(ui-data-model-tree-inspection-class)
:properties(
        user-defined-actions-list
        '((action1 "Action 1") (action2 "Action 2"))
        ))

(define-method action1-method application-form ()
        (let ((object !selected-object)
                )
        (pop-up-message
        (format nil "The object ~a has been selected"
                (object-name object))))
)

(define-method action2-method application-form ()
        (let ((object !selected-object)
                )
        (pop-up-message
        (format nil "The object ~a has been selected"
                (object-name object))))
```

)

Default Formula: nil

- **User-defined-area?**

  When non-nil, splits the work area in two parts with its lower part reserved to display a user defined form corresponding to the work-area-selected-object. Refer to the property user-form-class of data-model-node-mixin.

  Default Formula: nil

- **User-defined-area-height**

  Integer specifying the initial height of the user-defined area as a percentage of the height of the work area.

  Default Formula: 20

- **Work-area-height**

  Height of the work area (Includes the work area actions menu). This value is affected by the measurement property of the form. It is a fixed pixel value when measurement is 'pixels and a percentage of the height of the form when 'measurement is 'percentage.

  Default Formula: 99

- **Work-area-pixel-unit-width**

  Integer specifying the pixel-unit-width value used by the automatically generated widgets of the work area.

  Default Formula: (* 0.45 ^width).

- **Work-area-width**

  Width of the work area. This value is affected by the measurement property of the form. It is a fixed pixel value when measurement is 'pixels and a percentage of the width of the form when 'measurement is 'percentage

  Default Formula: 54

Output Properties

Properties:

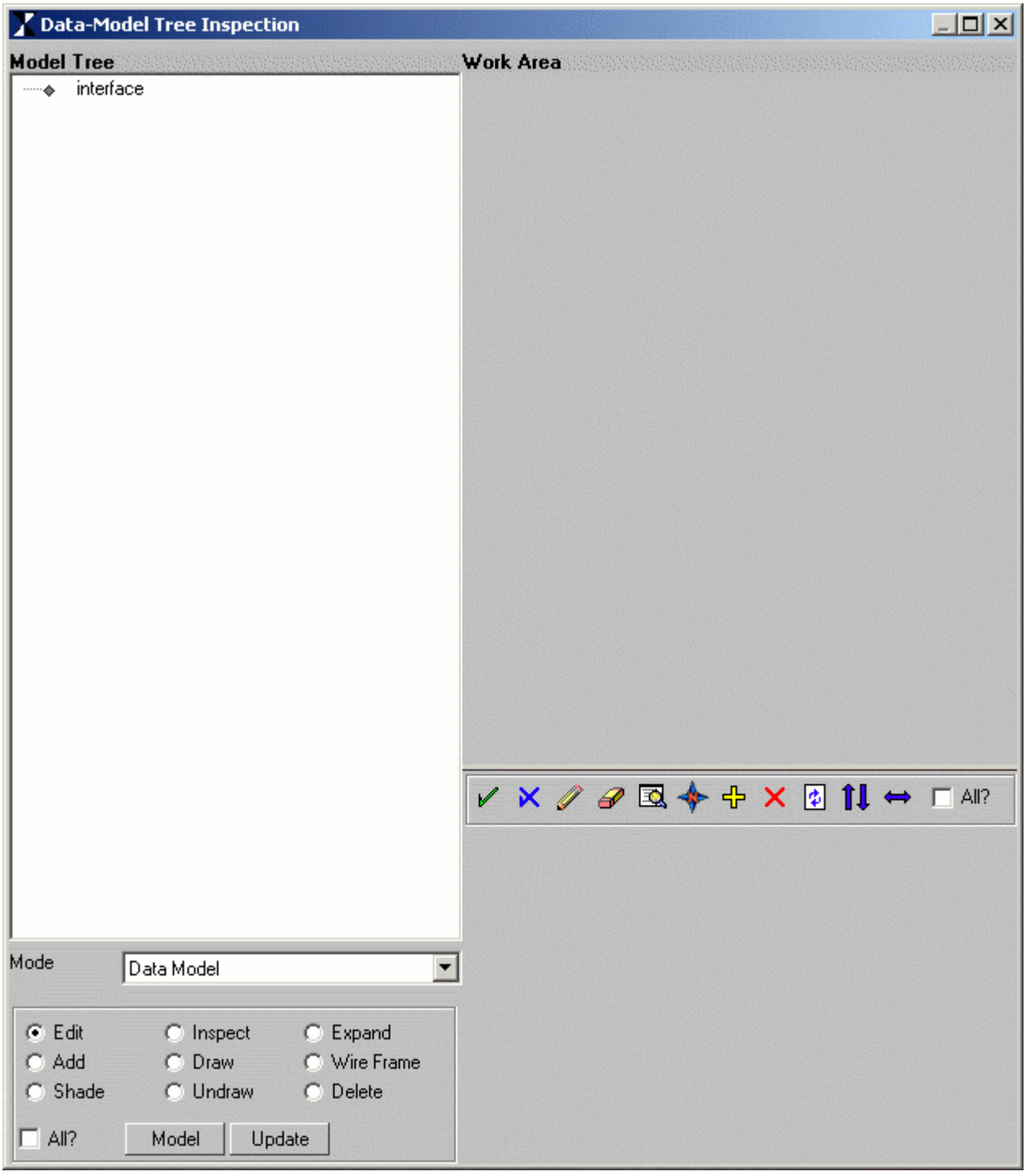
- **Selected-object**

  This property always holds the instance of the selected object (node) of the tree area. The user should not redefine the formula or change the value of this property. The property is used to query the form for the selected object.

- **Work-area-selected-object**

  This property always holds the instance of the selected object (node) of the work area. The work-area-selected-object and the selected-object usually hold the same instance unless new object selection functionality is added to the work area as in ui-data-model-main-form-class (subclass of ui-data-model-tree-inspection-class). This property is used to query the form for the selected object of the work area.

Figure 17-2. Ui-data-model-tree-inspection-class form



Tree Area - When the user clicks on a node (object) from the tree area, the selected action of the tree action menu is performed on the selected node. **Tree Actions Menu** Mode - This menu allows switching the form from data-model to all-objects mode and vice-versa. Edit - Opens the selected node for editing by displaying the corresponding group of widgets in the work area The properties to edit

should be specified in the property-objects-list of the selected object. Refer to data-model-node-mixin above. If the form is in data-model mode, the selected node's available? property must be t. If the form is in all-objects mode, the selected node can be edited anytime. The classes of widgets generated in the work area are always based on the classes of the properties specified in the property-objects-list of the selected node. Refer to the method ui-widget-class-name. **Note:** In all-object mode, if the selected object does not have a property-objects-list property, the method property-names-to-inspect is used to derive the properties to edit. Property-names-to-inspect is pre-defined on several AML classes and can be defined by the user; it must return a list of property names. This method is also used by the "Show All" toggle of the ui-inspect-object-form-class. Inspect - Inspects the selected node by displaying the object inspection form. This action allows modification of the formula and values of all the properties of the selected node. It also allows the addition of new properties and subobjects. Initially, the properties returned by calling the method property-names-to-inspect on the selected node are opened for inspection. The "show all" toggle on the object inspection form can be toggled on to show all properties. By default property-names-to-inspect is defined on aml-class to return nil meaning that all properties will be inspected. Property-names-to-inspect can be redefined by the developer and must return a list of property names. Inspecting an object should be performed only by advanced AML users. Expand: - Demands (creates) the subobjects of the selected node if they exist but have not been demanded yet. If the "All" toggle button is checked, the entire tree under the selected node is demanded v/s only the subobjects. Add - Adds an object to the selected node. Even if the selected-node is an instance of data-model-node-mixin, this action does not listen to the subobject-class-name, subobject-name-prefix and user-added-subobjects properties. Delete - Deletes the selected node when allowed. The user must be very careful in deleting objects since they can be referenced by other objects. Draw - When applicable, draws the selected node on the current active display/canvas. If the "All" toggle button is checked all objects in the tree starting at the selected node are also drawn. Undraw - When applicable, clears the selected node from the current active display/canvas. If the "All" toggle button is checked all objects in the tree starting at the selected node are also cleared. Shade - When applicable, shades (sets rendering to 'shaded) the selected node. If the "All" toggle button is checked all objects in the tree starting at the selected node are also shaded. Unshade - When applicable, unshades (sets rendering to 'boundary) the selected node. If the "All" toggle button is checked all objects in the tree starting at the selected node are also unshaded.

select-tree-widget [Method]

Defined on Classes:

ui-data-model-tree-inspection-class

Getting the object instance of the tree widget of data model forms. Returns object instance of the tree widget (ui-model-tree) of the form.

Arguments:

- **form**

  An instance of ui-data-model-tree-inspection-class

post-apply-method [Method]

Defined on Classes:

ui-data-model-tree-inspection-class

This method does not perform any action by default. It can be redefined by the user to define additional behavior for ui-data-model-tree-inspection-class (or for a user form that inherits from it). This method is automatically called when a user hits the apply button of the work area. The user can access the property work-area-selected-object of the form to get the selected object that is affected by the "apply". This method should be called from user source code.

Example 17-13. Post-apply-method example

Assuming my-form-class is a class that inherits from ui-data-model-tree-inspection-class.

```
(define-method post-apply-method my-form-class ()
        (let ((current-object !work-area-selected-object)
                )
        (perform-method-on-object selected-object)
        )
```

post-cancel-method [Method]

Defined on Classes:

ui-data-model-tree-inspection-class

This method does not perform any action by default. It can be redefined by the user to define additional behavior for ui-data-model-tree-inspection-class (or for a user form that inherits from it). This method is automatically called when a user hits the cancel button of the work area. The user can access the property work-area-selected-object of the form to get the selected object that is affected by the "cancel". This method should be called from user source code.

pre-subobject-addition-method [Method]

Defined on Classes:

ui-data-model-tree-inspection-class

This method does not perform any action by default. It can be redefined by the user to define additional behavior for ui-data-model-tree-inspection-class (or for a user form that inherits from it). This method is automatically called before a subobject is added to the selected node using the add-to-button of the work area action menu. The user can reference the property work-area-selected-object to get the object being added to. This method should be called from user source code.

post-subobject-addition-method [Method]

Defined on Classes:

ui-data-model-tree-inspection-class

This method does not perform any action by default. It can be redefined by the user to define additional behavior for ui-data-model-tree-inspection-class (or for a user form that inherits from it). This method is automatically called after an object is added using the add-to-button of the work area action menu. The subobject instance that has been added is passed as an argument to the method. This method should be called from user source code.

Arguments:

- **Instance-1**

  Instance of ui-data-model-tree-inspection-class

- **Instance-2**

  Instance of the object that was just added.

pre-object-deletion-method [Method]

Defined on Classes:

ui-data-model-tree-inspection-class

This method does not perform any action by default. It can be redefined by the user to define additional behavior for ui-data-model-tree-inspection-class (or for a user form that inherits from it). This method is automatically called right before a selected node is deleted using the tree action menu or the work area action menu. The user can reference the property work-area-selected-object or selected-object to get the object being deleted. This method should be called from user source code.

post-object-deletion-method [Method]

Defined on Classes:

ui-data-model-tree-inspection-class

This method does not perform any action by default. It can be redefined by the user to add behavior to a form that inherits from ui-data-model-tree-inspection-class. This method is automatically called right after a selected node is deleted using the tree action menu or the work area action menu. This method should be called from user source code.

See Also: data-model-node-mixin
ID_class_45_ui_45_data_45_model_45_main_45_form_45_class

select-subform [Method]

Defined on Classes:

ui-data-model-tree-inspection-class

The work area of a ui-data-model-tree-inspection-class above manages a set of subforms. Each subform corresponds to an object (or a set of objects of the same class) in the tree being inspected. The subforms are created and/or displayed on demand when an object is "edited". This method queries the form and returns the instance of the work area subform that corresponds to the node/object passed. If neither the object passed nor any other object of the same class has already been opened for "editing", the method returns nil specifying that no subform has been created yet for this specific object. This method is useful mainly for debugging purposes.

Arguments:

- **Form-instance**

  Instance of ui-data-model-tree-inspection-class.

- **Object**

  Object instance whose corresponding subform will be returned.

update-user-defined-area [Method]

Defined on Classes:

ui-data-model-tree-inspection-class

This method updates/refreshes the display of the user-defined area of the ui-data-model-tree-inspection-class instance.

update-tree-area [Method]

Defined on Classes:

ui-data-model-tree-inspection-class

This method updates/refreshes the display of the tree area of the ui-data-model-tree-inspection-class instance.

update-work-area [Method]

Defined on Classes:

ui-data-model-tree-inspection-class

This method updates/refreshes the display of the selected work-area form of the ui-data-model-tree-inspection-class instance.

C. User Interface Classification of Object Properties

property-classification-list [Method]

Defined on Classes:

graphic-object,

aml-class

The purpose of this method is to classify properties of a class into categories and types. This classification is used by the object inspection form and the work-area forms (data-model work areas) to allow a clean and rapid way to classify properties of an object in the AML GUI. The existence of the property-classification-list will eventually make the property-names-to-inspect method obsolete; however, all applications using property-names-to-inspect will still be compatible with the new GUI functionality. The method is expected to return property data in a certain format which is best described by the example below. The user is encouraged to define this method on his/her application classes.

This is a description of the format that should be returned by the method when it is defined by the programmer:

```
(list
(list "Category1 Label" property-name-list)
(list "Category2 Label" property-name-list)
...
)
```

A "property-name-list" is a list of property names (symbols) and can be a list of "property-info" lists. A property-info-list is a list of 3 elements: - '(property-name init-form widget-class) - The init-form is an init-form list (similar to the one used by series-object) that defines property formulas of the widget used to interface the property. - The widget-class is the class of the widget that will be used to interface the property (It must inherit from "model-interface-widget"). If widget-class is omited or is equal to nil, the default widget-class is used based on the class of the property specified (refer to the method ui-widget-class-name). NOTE: If you are defining the method property-classification-list on a class that inherits from data-model-node-mixin, you must explicitly set the formula of the property-objects-list property to nil. The reason for this is that the property-objects-list property has a formula in it in the data-model-node-mixin class definition. For backwards compatibility, the work-area checks for the existence of the property-objects-list property and if it evaluates to a value other than nil, the work-area will use this data. If the property-objects-list property evaluates to nil, then the property-classification-list method will be used by the work-area.

Example 17-14. Property-classification-list Example

Here is an example showing the default definition of "property-classification-list" on aml-class and graphic-object:

```
(define-method property-classification-list aml-class ()
  (let ((prop-names-to-inspect (property-names-to-inspect self))
        )
    (when prop-names-to-inspect
      (list
       (list "Main Properties" prop-names-to-inspect)
       )
      )
    )
  )

(define-method property-classification-list graphic-object ()
  (list
   (list "Main Properties" (property-names-to-inspect self))
   (list "Graphics Properties"
         '((render (options-list '(shaded boundary boundary-shaded) labels-
list '("Shaded" "Wire-frame" "Boundary-Shaded")) ui-option-property-menu-
class)
           (display? nil ui-toggle-property-button-class)
           (color (interactive-method 'select-color-method) ui-color-
selection-field-class)
           line-type
           line-width
           ))
   )
 )
```

Setup of an AML Application Interface

Setting up the interface of an AML application typically involves the following steps:

- Defining the form classes of an application

- Defining the application layout

- Setting up the initialization functions

o The interface main object

o The aml-init-function

The interface of an AML session is always started by calling the function aml.

Function

aml [Function]

This function starts the AML graphics and geometry server and transfers control from the AML listener (command prompt) to the user interface. Under UNIX, the same effect can be achieved by pressing the F6 key. The F6 key action is not present under Windows NT. On UNIX platforms, pressing the Edit then command button from the menu bar or calling the function exit-ui will return control to the AML listener.On Windows, the above actions for exiting the user interface do not have any effect; the user has control of the AML listener (command prompt) and interface simultaneously at all times.

Keyword Arguments:

- **:hide-source-editor?**

  When non-nil, hides the AML command prompt and the source code editor forms when the function is called. This function is effective only on WINDOWS

A. Defining the application layout

After defining the form classes of an application, an application layout class is typically defined as shown in the ui-layout-class example previously in this manual. A similar example is provided below for convenience. Please see the example below.

B. The interface main object

The interface main object is instantiated when aml starts and is represented by the TechnoSoft logo form that appears on the top left of the screen when the function aml is called. It contains properties that allow customizing the startup of an AML application.

The interface main can be referenced with the statement (the interface main).

1. **Display-menubar?**

   When non-nil, the AML main menubar is displayed when the function aml is called.

   Default Formula: t

2. **Menubar-height**

   Specifies the height in pixels of the AML main menubar.

   Default Formula: 32

3. **Menubar-widget**

   Specifies the width in pixels of the AML main menubar.

Default Formula: 500

4. **Pre-inits**

   When the aml function is called, the value of this property is.evaluated before the TechnoSoft logo form is displayed.

   Default Formula: nil

5. Post-inits

   When the aml function is called, the value of this property is evaluated after the TechnoSoft logo form and the main menubar (when applicable) are displayed.

   Default Formula: nil

Please see the example below.

C. The aml-init-function

aml-init-function [Function]

The aml-init-function is a system-defined function that is called at the end of the execution of the aml function. It is a blank function by default. The user can redefine this function to initialize his/her AML application. Refer to the example below.

D. Example

The following two statements are placed in the application source code. However, they also need to be placed in the aml-init.tsi file in case the user is running from a user-saved AML image. (See save-aml-image) The aml-init.tsi file is located in the working directory on WINDOWS platforms and in the user's home directory on UNIX platforms (The filename is preceded by a dot on UNIX, ".aml-init.tsi")

```
(change-formula (the interface main pre-inits) '(sample-pre-init-function))
(change-formula (the interface main post-inits) '(sample-post-init-function))
```

The following code is placed in the application source code files. (Consider main-form-class, form1-class, and form2-class to be user defined top-level forms.)

```
(define-class sample-application-layout
  :inherit-from(ui-layout-class)
  :properties(
              main-form-postion '(10 10)
              form1-position '(200 10)
              form2-position '(400 10)
              )
  :subobjects(
              (main-form :class 'main-form-class
                         x-offset (first ^^form1-position)
                         y-offset (second ^^form1-position)
                         )
              (form1 :class 'form1-class
                     x-offset (first ^^form1-position)
```

```
                    y-offset (second ^^form1-position)
                    )
             (form2 :class 'form2-class
                    x-offset (first ^^form2-position)
                    y-offset (second ^^form2-position)
                    )
             )
  )

(defun display-application-main-form ( )
  (let* ((layout (the interface sample-layout (:error nil)))
         )
    (when layout (display (the main-form (:from layout))))
    ))

(defun sample-pre-init-function ( )
  (unless (the interface sample-layout (:error nil))
    (add-object (the interface) 'sample-layout 'sample-application-layout))
  (change-formula (the interface main menubar-width) 600)
  )

(defun sample-post-init-function ( )
  ;; Adding a menu to the AML main menubar
  (add-header-menu 'application-menu
                   :label "Sample AML Application"
                   :menu-items
                   '(("Main Form" '(display-application-main-form))
;;; You can have more menu items here
                     )
                   )
  )

(defun aml-init-function ( )
        ;; Other final initializations can be placed here.
  )
```

Application Forms

Classes and Methods

This section describes a set of special purpose forms that can be instantiated as a part of the GUI of an AML application. The non-documented properties and sub-objects of these forms must not be accessed by AML users; this will ensure proper functionality and will not to jeopardize compatibility with future version.

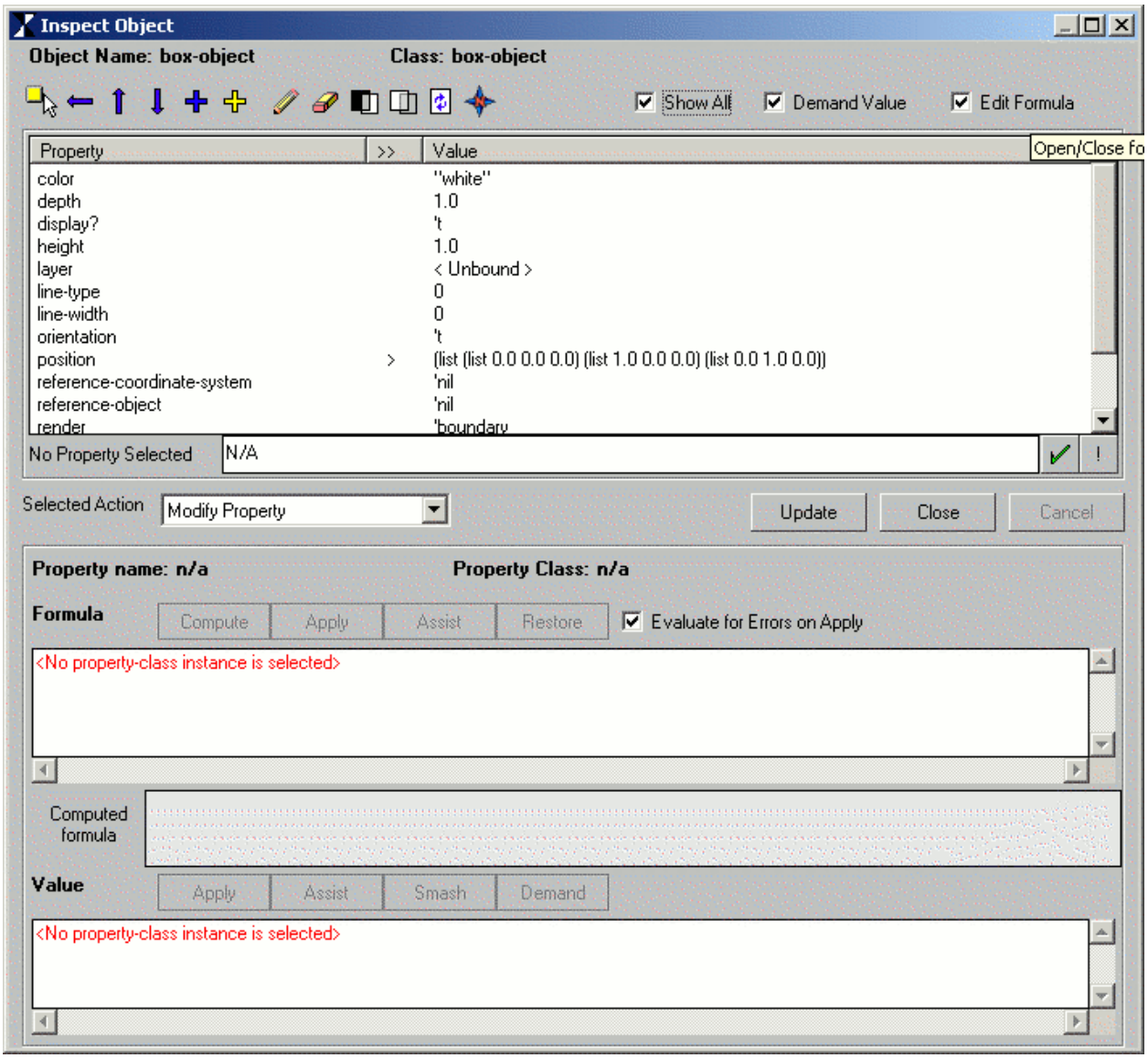
ui-inspect-object-form-class [class]

This form allows the inspection of an object and its properties and allows the addition of new properties and subobjects to that object. It also contains additional interactive functionality related to the object

being inspected. When the user adds a property to an object using the inspection form, the new property is not referenced and its value stays unbound until demanded. When the user adds a subobject to an object using the inspection form, the new subobjet is not referenced and stays unbound until demanded.

Inheritance: ui-standard-form-superclass

Figure 17-3. Ui-inspect-object-form-class



inspect-object-form [Method]

Defined on Classes:

ui-inspect-object-form-class

This method opens an object for inspection using a ui-inspect-object-form-class instance.

Arguments:

- **Object-instance**

  instance of the desired object to inspect

Keyword Arguments:

- **:creation?**

  Optional keyword. When non-nil, the form is opened is "add-object" mode, can be used when an object is being added to allow canceling the addition (i.e. deleting the object) and to disable access to some features of the inspection form.

- **:properties**

  Optional keyword specifying the a list of property names of object-instance. Only these properties will be shown in the property list area when the "show all" toggle is off. When nil, all properties are shown.

- **:x**

  Optional keyword specifying the offset (in pixels) of the inspection form from the left of the screen. When nil, takes the default value defined with the form class.

- **:y**

  Optional keyword specifying the offset (in pixels) of the inspection form from the top of the screen. When nil, takes the default value defined with the form class.

- **:width**

  Optional keyword specifying the width (in pixels) of the inspection form. When nil, takes the default value defined with the form class.

- **:height**

  Optional keyword specifying the height (in pixels) of the inspection form. When nil, takes the default value defined with the form class.

property-names-to-inspect [Method]

Defined on Classes:

ui-inspect-object-form-class

This method is automatically called by the system when an object is opened for inspection (i.e. also when the method inspect-object-form is called on that object). The AML developer can redefine this method on any class to specify the "Main" properties that are visible when an instance of that class is opened for "inspection". Only these properties are visible when "Main Properties" is selected in the property category menu (the top-left of inspection form). This method must always return a list of property names. This method is pre-defined on a number of AML classes, it returns nil for all other classes. When it returns nil, no "Main Properties" option will be available on the form, and all properties are visible. Example:

```
(define-method property-names-to-inspect box-object ( )
'(width height depth)
)
```

See Also: property-classification-list

ui-graphic-control-toolbar-class [class]

The figure below shows an instance of this class. It must be instantiated as a subobject of any form. This class is used to interactively manage and control a graphics display window (A valid instance of ui-canvas-class must be specified by the property canvas-object). It controls view parameters, rendering, lights and other graphical object interaction features. The bitmaps used by the buttons of this class require the existence of a valid ui-bitmaps logical path. This path should already exists in the default configuration of AML.

Inheritance: ui-subform-class

Properties:

- **canvas-object**

  Points to an instance of ui-canvas-class. All buttons of the toolbar (described below) deal with the canvas instance specified here.

  Default Formula: nil

- **contour-level**

  Contour level for analysis plots (when dealing with a canvas-object whose type = 'indexed-color).

  Default Formula: 16

- **delta-pan**

  Pan increment factor used by the incremental pan buttons. This value can be changed by right clicking on the mouse-pan button.

  Default Formula: 1

- **delta-rotate**

  Incremental rotation angle used by the incremental rotate buttons. This value can be changed by right clicking on the mouse-rotate button.

  Default Formula: 10

- **main-form**

  Instance of ui-data-model-tree-inspection-class or ui-application-main-form-mixin. The method edit-node-method and inspect-node-method must be defined on this main form. This form is needed for the "Edit" button described below to function correctly.

  Default Formula: nil

- **reference-coordinate-system**

  Instance of coordinate-system-class. This is the reference coordinate system used by the view buttons of the toolbar. When nil, it will reference the global coordinate system.

  Default Formula: nil

- **zoom-factor**

  Zoom factor used by the zoom buttons.

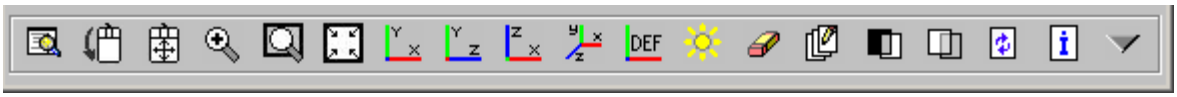
  Default Formula: 1.25

- **availability-list**

  This property controls the availability of the toolbar button. When nil, all buttons ae enabled. Otherwise, it must be a list of t/nil flags, one flag per button (t means the button is enabled, nil means the button is grayed out).

  Default Formula: nil

Figure 17-4. ui-graphic-control-toolbar-class



Button Description - (From left to right) **Edit** - Left-click selects an object edit. Right-click selects an object to inspect. **Mouse-rotate (left click)** - Sets the graphics permanent mode of the active display to "Rotate View"; i.e. left clicking and dragging on the active display anytime will rotate the view. Mouse rotation options on active graphics display: - Left-click and drag: Free rotation. - Left-click and drag while holding the shift key down: Rotate about the view vertical axis. - Left-click and drag while holding the control key down: Rotate about the view horizontal axis. - Left-click and drag while holding both the control and shift keys down: Rotate about the view normal axis. **Mouse-pan (left click)** - Sets the graphics permanent mode of the active display to "Pan View"; i.e. left clicking and dragging on the active display anytime will pan the view. Mouse panning options on active graphics display: - Left-click and drag: Free panning. - Left-click and drag while holding the shift key down: Pan along the view horizontal axis. - Left-click and drag while holding the control key down Pan along the view vertical axis. - Left-click and drag while holding both the control and shift keys down Zoom. **Mouse-pan (right click)** - Pops up a prompt to allow changing the value of the incremental pan factor used by the following 3 buttons. **Zoom In/Out** - Left-click to zoom the view in, right-click to zoom-out. **Zoom window** - Left-click to define/drag a zoom window on the active canvas. **Fit** - Left-click to fit all drawn objects within the current view rectangle. **Four View buttons** - Changes the active display's view to align with the view axes specified on the buttons. The reference coordinate system is the global coordinate system by default. The reference coordinate system can be set to any instance of coordinate-system-class by right-clicking on the "Default View" button and selecting the "Change View Reference Coord system" option. **Default View** - Left-click to change the default view, right-click gives user-defined views options. Notes on the format of the view files: 1. The file can define several views, each view definition occupies exactly 5 lines. 2. Blank lines are allowed between views. 3. Any view parameter line can be NIL or blank. The parameter in the active display is used in that case. **Example Views**

```
View Number 1
0.0 0.0 1.0 This is the EYE vector, We can put any comment here
0.0 1.0 0.0 This is the UP vector
0.0 0.0 0.0 This is the camera target position
200.0 200.0 width, height of camera field

View Number 2
0.0 0.0 1.0 This is the EYE vector, We can put any comment here
0.0 1.0 0.0 This is the UP vector
nil
200.0 200.0 width, height of camera field
```

```
Another View
1.0 0.0 0.0 This is the EYE vector, We can put any comment here
0.0 0.0 1.0 This is the UP vector
nil
nil
```

Lights - Opens the light management form to allow the addition and modification of lights. **Undraw** - Left-click to undraw one or more object. The object selection method is based on the system's global object selection mechanism. Right-click clears all objects from the canvas. **Draw Group** - Left-click to draw a defined group of objects. Right-click to define a drawing group. **Shade** - Left-click to shade one of more objects. The object selection method is based on the system's global object selection mechanism. Right-click will give more shading options. **Unshade** - Left-click to unshade one of more objects. The object selection method is based on the system's global object selection mechanism. **Regen** - Left-click to call regen which will regenerate any "smashed" geometry of drawn objects. Right-click to call refresh which will flush the canvas by keeping all valid drawn objects and clearing all "smashed" objects. **Information** - Left-click to select an object and display information about it. **More Options** - Left-click to access several other options.

ui-multiple-value-selection-form-class [class]

This is an active form that allows the interactive selection of a list of values from a list of available values

Inheritance: ui-active-form-class, ui-standard-form-superclass

Input Properties:

Properties:

- **available-labels-list**

  List of strings specifying the labels of the available values as they appear on the form. By default, it loops through available values and converts each value to a string.

- **available-values-list**

  List of available values.

- **allow-duplicates?**

  When non-nil, allows the selection of a value more then once.

  Default Formula: nil

- **close-button?**

  When nil, the close button does not appear on the form.

  Default Formula: t

- **left-columns-label**

  String specifying the header label of the left column.

  Default Formula: (default "Selected Values")

- **right-columns-label**

String specifying the header label of the right column.

Default Formula: (default "Available Values").

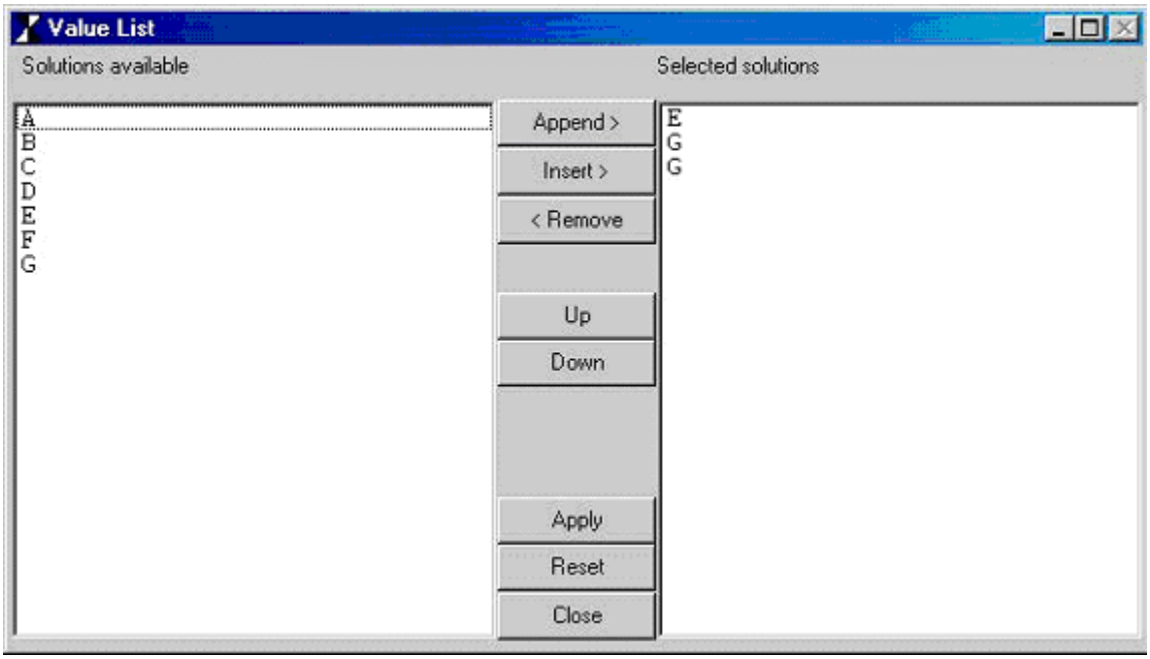
Input/Output Properties:

Properties:

- **selected-values-list**

  List of selected values. This property is modified every time the user changes the selected values on the form.

Figure 17-5. ui-mutliple-value-selection-form-class



Layouts

Classes

When developing an AML application, a layout should be used to group the user interface top-level-forms of that application. The layout is a manager and a placeholder of a set of top-level forms; a top-level form is typically a subobject of a layout. An AML application can define more than one layout. Layouts are instances of ui-layout-class described below.

ui-layout-class [class]

Ui-layout-class is a manager of a group of top-level-forms of an AML application. It is typically instantiated as a subobject of "(the model-manager interface)". The layout controls the general appearance properties (font, foreground-color, background-color...) of the overall application interface unless those properties are overwritten at a lower level of the interface tree. The user can add properties to a layout to manage the position, sizes and other parameters of the layout forms. This user interface

architecture can allow communication between forms sharing a common layout in an efficient and robust manner.

Inheritance: object

refer to ui-widget for a description of the following properties

Properties:

- **Font**

  By default, it gets its value from (the model-manager interface).

  Default Formula: (default)

- **Foreground-color**

  By default, it gets its value from (the model-manager interface).

  Default Formula: (default)

- **Background-color**

  By default, it gets its value from (the model-manager interface).

  Default Formula: (default)

- **Light-slope-color**

  By default, it gets its value from (the model-manager interface).

  Default Formula: (default)

- **Dark-slope-color**

  By default, it gets its value from (the model-manager interface).

  Default Formula: (default)

- **Slope-thickness**

  By default, it gets its value from (the model-manager interface).

  Default Formula: (default)

- **Selected-color**

  By default, it gets its value from (the model-manager interface).

  Default Formula: (default)

See Also: ui-widget, ui-form-class, Setup of an AML application Interface.

Example 17-15. ui-layout-class example

```
;; Consider form1-class, form2-class, and form3-class to be  user defined
forms that inherit from ui-form-class.

(define-class Sample-application-layout
  :inherit-from(ui-layout-class)
  :properties(
              form1-postion (10 10)
              form1-position (200 10)
              form3-position (400 10)
              )
```

```
  :subobjects(
              (form1 :class 'form1-class
                     x-offset (first ^^form1-position)
                     y-offset (second ^^form1-position)
                     )
              (form2 :class 'form2-class
                     x-offset (first ^^form2-position)
                     y-offset (second ^^form2-position)
                     )
              (form3 :class 'form3-class
                     x-offset (first ^^form3-position)
                     y-offset (second ^^form3-position)
                     )
              )
  )

;; To instantiate this application layout, the AML-init-function can be used:

(defun AML-init-function ( )

  (unless (the interface sample-layout (:error nil))
    (add-object (the interface) 'sample-layout 'sample-application-layout))

  )

;; Keep in mind that instantiating a layout does not display the forms of
that layout.
```

Chapter 18. Foreign Function Interface

Introduction

The foreign-function interface allows the user to link compiled foreign code dynamically into a running AML session. Foreign code is defined to be code not written in AML. For example, code written in C or Fortran is foreign code. The foreign-function interface allows users to load compiled code written in a foreign programming language into a running AML, execute it from within AML, return to AML and pass data back and forth between AML and the foreign code.

This mechanism is very powerful, as programs need not be re-coded into AML to use them. Another advantage arises during program development. For example, a large graphics library can be linked into AML and all the functions will be accessible interactively. This enables rapid prototyping of systems that use the library functions.

All foreign code should be in a shared object (.so or .sl on Unix) or dynamic library (.dll on Windows) file which is mapped into a running AML process. The function load is used to load the foreign code (library) into the AML image. Thus foreign code must always be loaded into an image at runtime and can not be saved in an image using save-aml-image.

Notes:

One library file cannot depend on an already loaded library file.

Each file must be complete in itself such that all necessary routines must either be defined in the file or must be in a library specified when the file is created.

If you modify a so/sl/dll file that has been loaded, you must load the modified file. If debugging and modifying the so/sl/dll file while it has already been loaded into an AML session, the user will have to release the library using the unload-library function, modify it, and then load it again.

Syntax

Functions

char*-to-string [Function]

This function converts a "C" string to an AML string.

Arguments:

- **pointer**

  memory pointer returned by list-to-fptr or allocated in external code

define-foreign-function [Language Construct]

This construct is the base mechanism to define a foreign function in AML. For every external function that is called in AML, a corresponding function needs to be defined in AML using this construct. Each input argument to the external function needs to have a one-to-one corresponding argument that is in "data-type" agreement.

Example 18-1. Format of define-foreign-function

```
(define-foreign-function
                (aml-function-name
                (:language foreign-language)
                (:name foreign-function-name)
                (:return-type foreign-function-return-type)
                )
        (arg1 foreign-type)
        (arg2 foreign-type)
        ......
        )
```

Arguments:

- **aml-function-name**

  a symbol specifying the name of the function that will be called in AML

- **foreign-language**

  :c or :fortran

  Default Value: :c

- **foreign-function-name**

  A string specifying the name of the foreign function this function calls

- **foreign-function-return-type**

  The foreign-type which the foreign function returns. Note that if you define :string as the :return-type in your foreign function definition, AML will make a copy of your string (char*) returned from the foreign function, which means you can not get hold of the original string pointer address; this may cause a memory leak. One alternative is to define the :return-type as :pointer and to use foreign-string-value to convert it to an AML string. You will also need to call free-foreign-pointer to free the original string returned from the foreign function.

- **argn**

  Argument name (need not be the same name as in the foreign function, but the number of arguments and the type need to match)

- **foreign-type**

  Can be one of the types described in Figure 1.1 below. In general, any type that you come across that does not appear in this table will be treated as an integer by default in AML.

Figure 18-1. define-foreign function - Table 1

AML Foreign Type	C Type	Fortran Type
:int	int	integer
:float	float	real
:double	double	double precision
:char	char	character*N
:pointer	any pointer or void *	integer [arg(N)]
:short	short	integer
:void	void	integer
:string	char*	character*N

define-foreign-callable [Language Construct]

This construct is the base mechanism to define a function in AML that is callable from a foreign code. To call it from a foreign language, aml-call-with-name ('function-name' ... arg) has to be used. Then the AML runs the function.

Example 18-2. Format of define-foreign-callable

```
(define-foreign-callable
                (function-name foreign-function-name)

        ((arg1 foreign-type)
        (arg2 foreign-type)
        ......)
        )
```

foreign-aref [Function]

This function obtains a value from a foreign array. Note that the index must be within the size of the array and must be in agreement with respect to data type. Any reference outside the size of the array will return random data that is currently in the referenced memory space.

Arguments:

- **foreign-pointer**

  memory pointer returned by list-to-fptr or allocated in external code

- **index**

  index counter within the array starting from 0

- **Foreign-type**

  argument for the data type of the value placed in the foreign array

Example 18-3. Foreign-aref Example

```
> (list-to-fptr '(11 234 345) :int)
;; returns : 56650768
> (foreign-aref 56650768 0 :int)
```

```
;; returns : 11
> (foreign-aref 56650768 1 :int)
;; returns : 234
> (foreign-aref 56650768 2 :int)
;; returns : 345
> (foreign-aref 56650768 3) ;;out of array size
; returns bad data : 0
> (free-foreign-pointer 56650768)
;; frees the pointer and returns : t
```

To set a new value to foreign-aref we use setf :

```
>(setf (foreign-aref pointer index type) new-value)
```

foreign-value [Function]

This function gets the current value specified in the foreign-pointer (allocated in AML or in external code).

Arguments:

- **foreign-pointer**

  memory pointer returned by list-to-fptr or allocated in external code

- **Foreign-type**

  argument for the data type of the value placed in the foreign array

To change the value use setf :

```
> (setf (foreign-value pointer type) new value)
```

foreign-type-size [Function]

This function returns the memory size of a type.

Arguments:

- **Foreign-type**

  argument for the data type of the value placed in the foreign array

Example 18-4. Foreign-type-size Example

```
> (foreign-type-size :int)
;; returns : 4
> (foreign-type-size :double)
;; returns : 8
```

foreign-string-value [Function]

This function takes a string-value returned from a foreign function and converts to an AML string.

Arguments:

- **string-value**

  Value that contains a string given from a foreign function

Example 18-5. Foreign-string-value Example

```
(malloc-foreign-string "Hello")
;; returns : 56676600
(foreign-string-value 56676600)
;; returns : "Hello"
(free-foreign-pointer 56676600)
;; returns : t
(foreign-string-value 56676600)
;; returns : "nil"
```

This example was executed on a PC. Note that the "nil" returned from the last function call to foreign-string-value given above is platform dependent such that it may return different values on a Unix machine. Users should never make a call to access a foreign value that has been freed.

free-foreign-pointer [Function]

This function frees the foreign memory that is originally allocated in AML by the functions list-to-fptr or malloc-foreign-pointer, or memory allocated in external code. This is equivalent to "free" in C. Note that a user can also free a pointer that was allocated in external code. Note that this function will free the memory allocated, but the memory space is not guaranteed to be cleared immediately when the function is called. The memory is cleared periodically at certain times in the AML session. Therefore any future references to the same memory space may contain ill-formed data.

Arguments:

- **foreign-pointer**

  This is a memory pointer (integer) returned by list-to-fptr or malloc-foreign-pointer or allocated in external code

get-list-from-fptr [Function]

This function returns a list of values specified in the foreign-pointer (allocated in AML or in external code).

Arguments:

- **foreign-pointer**

  memory pointer returned by list-to-fptr or allocated in external code

- **number-of-values**

  the number of values will be returned

- **Foreign-type**

  :int, :float, :double, :string, :pointer, :short, :char

list-to-fptr [Function]

This function allocates memory in AML and initializes the value that will be accessed in the foreign function. This is equivalent to allocating memory through "malloc" in C to create an array. Any pointer created using this function can be passed to the C function and de-referenced like any C pointer. This function creates an array from the list of values passed to the function.

Arguments:

- **list-of-values**

  the list of values used to initialize the memory

- **Foreign-type**

  :int, :float, :double, :string, :pointer, :short, :char

load [Function]

This function loads a foreign library into the current AML session. All loaded foreign code should be in a shared object (.so or .sl on Unix) or dynamic library (.dll on Windows) file.

Arguments:

- **foreign-file**

  a string to specify the foreign file that will be loaded into AML. This should be a .dll for the PC, .sl for HP, and SGI, and .so for IBM and SUN platforms.

malloc-foreign-pointer [Function]

This function allocates memory for use in foreign code.

Arguments:

- **size**

  The amount of bytes

malloc-foreign-string [Function]

This function converts an AML string to a string that can be passed to a foreign function.

Arguments:

- **string**

  AML string

For an example refer to foreign-string-value. This is the same function as string-to-char*.

string-to-char* [Function]

Please refer to malloc-foreign-string.

unload-foreign-library [Function]

This function unloads a foreign library from the current AML session.

Arguments:

- **foreign-file**

  a string to specify the foreign file that will be unloaded from AML. This should be a .dll for the PC, .sl for HP, and SGI, and .so for IBM and SUN platforms.

Examples

The following example shows a typical application of the Foreign Function Interface in AML.

Example 18-6. Example in "C" - "Scaling a Box"

The purpose of scale-box aml function is to allocate the foreign pointers in AML, assigning l, w, d values and to call the AML function called scale_box_c, which calls the C function called scale_box and finally gets the values back using the foreign-value function.

Note that a print statement in the AML code prints the result, which is 1, returned from the scale_box_c function.

The following function called scale_box is written in "C" simply multiplying the values located in three pointers by a scale factor.

```
/*----------------------------------------*/
#include "stdio.h"
#include "stdlib.h"
int scale_box(double s, double *l, double *w, double *d)
{
*l *= s;
*w *= s;
*d *= s;
return(1);
}
/*----------------------------------------*/
```

The following AML code defines the foreign function within AML called scale_box_c which is written in "C", calls the "C" function named scale_box which returns the value 1 (a signed 32 bit number). This function takes four arguments; one is a double float type and the rest are pointers to double float values. The foreign function scale_box_c is then called within an AML function called scale-box that takes four number arguments. Note that the AML function scale-box forces all of the arguments passed to the function to be double float data types.

```
(in-package :aml)

(define-foreign-function
        (
        scale_box_c ;;name of AML function
        (:language :c) ;;language is in "C"
        (:name "scale_box") ;;name of exported "C" function
        (:return-type :signed-32bit) ;;data type returned from "C" function
        )
        (scale :double) ;;first argument passed to "C" function
        (l (:pointer :double)) ;;second argument passed to "C" function
```

```
        (w (:pointer :double)) ;;third argument passed to "C" function
        (d (:pointer :double)) ;;fourth argument passed to "C" function
)

(defun scale-box (scale l w d)
(let* (
        (l-ptr (list-to-fptr (list (double l)) :double)) ;;pointer to l
        (w-ptr (list-to-fptr (list (double w)) :double)) ;;pointer to w
        (d-ptr (list-to-fptr (list (double d)) :double)) ;;pointer to d
        (c-call (print (scale_box_c (double scale) l-ptr w-ptr d-ptr)))
;;call to ff
(result (list
        (foreign-value l-ptr) ;;extract value from l pointer
        (foreign-value w-ptr) ;;extract value from w pointer
        (foreign-value d-ptr) ;;extract value from d pointer
        ))
)
(free-foreign-pointer l-ptr) ;;free memory in l pointer
(free-foreign-pointer w-ptr) ;;free memory in w pointer
(free-foreign-pointer d-ptr) ;;free memory in d pointer
result ;;return result from above
)
)
```

The typical steps involved in this process are as follows:

- Compile the "C" code and create a shared library.

- Load the shared library scale_box.o into the AML environment with...

```
(load "scale_box.so") ;;on Unix
(load "scale_box.dll") ;;on PC
```

- Compile or load the AML code, containing the scale_box_c and scale-box functions.

- Call scale_box from AML with...

```
(scale-box 10.0 2.0 3.0 3.8)

(20.0 30.0 38.0)
```

Notice the scale-box AML function printed the value 1 to show that AML called the foreign function scale_box_c.

Example 18-7. Example in FORTRAN - Calling FORTRAN Functions and Subroutines

The following examples show how to exchange data between AML and FORTRAN.

Notes:

1. The system allows the use of either Functions or Subroutines in FORTRAN. (See add-function in AML, AddFunction in FORTRAN, and also add-subroutine, add-subroutine-internal in AML and AddSubroutine in FORTRAN.)

2. If interfacing with non-array values, define a foreign-function with :language as :fortran and pass values to the function as it would be called in a FORTRAN function. In other words, it will pass by reference and AML will get changed values back directly. (See add-subroutine, add-subroutine-internal in AML and AddSubroutine in FORTRAN.)

3. If dealing with array values, define the foreign-function with :language as :c and treat it just like a "C" array and the system will handle this. (See add-arrays-subroutine-internal and add-arrays-subroutine in AML, and AddArraysSubroutine in FORTRAN.)

```
(in-package :aml)

(define-foreign-function
  (add-function
   (:language :fortran)
   (:name "ADDFN") ;; name of exported FORTRAN function
   (:return-type :double)
   )
  (a :double)
  (b :double)
  )

(define-foreign-function
  (add-subroutine-internal
   (:language :fortran)
   (:name "ADDSBR") ;; name of exported FORTRAN subroutine
   (:return-type :void)
   )
  (a :double)
  (b :double)
  (c :double)
  )

(defun add-subroutine (a b)
   (let* (
          (c 0.0)
          )
     (add-subroutine-internal (double-float a) (double-float b) c)
     c
     )
   )

(define-foreign-function
  (add-arrays-subroutine-internal
        (:language :fortran)
        (:name "ADDARR") ;; name of exported FORTRAN subroutine
        (:return-type :void)
        )
  (a :pointer)
  (b :pointer)
  (c :pointer)
  )
```

```
;; NOTE: The FOTRAN code assumes that a-list and b-list are a list of 5
numbers each
(defun add-arrays-subroutine (a-list b-list)
  (let* (
         (c 0.0)
         (a-ptr (list-to-fptr (double-float a-list)    :double))
         (b-ptr (list-to-fptr (double-float b-list)    :double))
         (c-ptr (list-to-fptr (double-float (list c))  :double))
         return-value
         )
    (add-arrays-subroutine-internal a-ptr b-ptr c-ptr)
    (setf return-value (nth 0 (get-list-from-fptr c-ptr 1 :double)))
    (free-foreign-pointer a-ptr)
    (free-foreign-pointer b-ptr)
    (free-foreign-pointer c-ptr)
    return-value
    )
  )
```

The following code contains corresponding FORTRAN functions and subroutines which must be compiled into a shared library or dll. The programmer has to make sure all functions are exported in the DLL.

```
FUNCTION ADDFN(a,b)
REAL*8 a,b,ADDFN
ADDFN = a+b
END

SUBROUTINE ADDSBR(a,b,c)
REAL*8 a,b,c
WRITE (6,*) a,b
c=a+b
WRITE (6,*) c
RETURN
END

SUBROUTINE ADDARR(a,b,c)
REAL*8 a,b,c
DIMENSION a(5), b(5)
INTEGER i
c=0
 DO 10 i=1,5
  WRITE (6,*) a(i), b(i)
  c = c + a(i) + b(i)
10 CONTINUE
RETURN
END
```

Example 18-8. Examples using PostThreadMessage and foreign callable between AML and "C".

```
;; AML
(define-foreign-callable (add-two) ((first-number :int)
                                    (second-number :int)
                                    (return-value :pointer))
  (without-gc
    (set-foreign-value return-value
                       (+ first-number second-number)
                       :int)))

;; C
/* Most portable method for calling AML code from foreign code.
   Works on all supported platforms.
   AML will process aml_call_with_name immediately, even if it is busy
   performing other calculations.  Developers must be careful.
 */

#ifdef __WIN32__
#include <windows.h>
extern "C" {
__declspec(dllexport) void (*aml_call_with_name) (char *, ...);
}
#else
void (*aml_call_with_name) (char *, ...);
#endif

int
add_two_in_aml (int first_number, int second_number)
{
    int sum;

    aml_call_with_name ("add-two", first_number, second_number, &sum);

    return sum;
}

;; AML
(defvar *add-two-message-id*       10001)
(defvar *add-two-reply-message-id* 10002)

(defun add-two (wparam lparam)
  "Windows thread message demonstration.
wparam - thread id of the sender
lparam - array of two integers"
  (post-thread-message wparam
                       *add-two-reply-message-id*
                       (+ (foreign-aref lparam 0 :int)
                          (foreign-aref lparam 1 :int))
                       (null-pointer)))

(register-message-handler *add-two-message-id* 'add-two)

;; C
#include <windows.h>
```

```
/* Safest method for calling AML code from foreign code.
   Works on Microsoft Windows only.
   AML will process the thread message only when it isn't busy.
   The developer is freed from synchronization concerns. */

int
add_two_in_aml (DWORD aml_thread_id, int first_number, int second_number)
{
    int numbers[2];
    MSG msg;

    numbers[0] = first_number;
    numbers[1] = second_number;

    /* Post the message to AML.  Does not block. */
    PostThreadMessage (aml_thread_id,
                       10001,
                       (WPARAM) GetCurrentThreadId(),
                       (LPARAM) numbers);


    /* Wait until AML posts a reply message. */
    GetMessage (&msg, NULL, 10002, 10002);

    return (int) msg.wParam;
}
```

Building a Dynamic-Link Library (.dll) on the PC

Building a C .dll:

The following instructions are based on using Microsoft Visual C++.

1. Define your project as a DLL (Win32 dynamic link library). Say your project is called aml_foreign

2. Add your .c file containing the functions to be called from AML into the project.

3. Suppose you want to export the following functions to call from AML:

test_func

test_func_2

4. Add a file to your project called aml_foreign.def containing the following text:

LIBRARY aml_foreign

DESCRIPTION 'aml foreign function interface on NT'

EXPORTS test_func

test_func_2

5. Add the foreign functions corresponding to test_func and test_func_2 using define-foreign-function within AML.

Building a Fortran .dll: Please refer to your Fortran documentation for help with compiling a PC Fortran .dll.

Building a Shared Library (.sl or .so) on UNIX

Building shared libraries on Solaris 2.4 or later

On Solaris, you must produce .so files which are loadable into AML. The -K pic flag is optional (only on Solaris), and creates slightly different modes of sharing when used. As an example:

% cc -c -K pic my_file.c

% ld -G -o my_file.so my_file.o

Fortran is similar:

% f77 -c my_file.f

% ld -G -o my_file.so my_file.o

Building shared libraries on HP-UX 10.20

The shared objects loadable by AML are .sl files. You make these by using cc and ld in this way:

% cc +z -Ae +DA1.1 -c my_file.c

%ld -b -o my_file.sl my_file.o

For Fortran, the commands are:

% f77 +z +DA1.1 -c my_file.f

%ld -b -o my_file.sl my_file.o

Building shared libraries on SGI/IRIX 6.2 or later

You must use the -n32 switch to the C compiler and linker to be able to produce .so files which are loadable into AML. For example:

% cc -n32 -c -K PIC my_file.c

% ld -n32 -shared -all -o my_file.so my_file.o

or for Fortran:

% f77 -n32 -c my_file.f

% ld -n32 -shared -all -o my_file.so my_file.o

Building shared libraries on AIX 4.2 or later

Please contact TechnoSoft Inc. to obtain the make_shared script to produce .so files which are loadable into AML. For example:

% cc -c -D_BSD -D_NO_PROTO -D_NONSTD_TYPES -D_MBI=void my_file.c

% bin/make_shared -o my_file.so my_file.o

and for Fortran:

% xlf -c my_file.f

% bin/make_shared -o my_file.so my_file.o -lm

Chapter 19. Parsing Classes

Patterns

Parsing classes use patterns for matching text. A pattern, in its simplest form, is a string. For example: "velocity = 5" is a pattern that will match any occurrence of "velocity = 5" within a text file or string. Patterns are more powerful as they can be expressed in terms of regular expressions in order to search and match variable patterns. Next is a brief description of the constructs that form regular expression patterns that can be used with AML parsing classes:

General Definition

A regular expression pattern can be defined as a string of constructs of the form: " C0 [quantifier]C1[quantifier]C2[quantifier]..." C is one of the following:

- An alphanumeric character
- An escaped character (see below)
- A character class (see below)
- A parenthesized pattern (see below)

A quantifier is optional and specifies the number of times the preceding C must occur.

Quantifiers

The regular expression "ab{1,3}c" is an example of using a quantifier. It will match any string that starts with "a" followed by 1,2 or 3 "b"s then followed by "c". The quantifier {1,3} specifies that the character "b" can occur no less then 1 time but no more then 3 times. An occurrence of "abc" is considered a match as well as an occurrence of "abbc" and "abbbc". Quantifiers:

- {n,m} - Must occur at least n times but no more then m times
- {n,} - Must occur at least n times
- {n} - Must occur exactly n times
- * - Must occur 0 or more times (same as {0,})
- + - Must occur 1 or more times (same as {1,})
- ? - Must occur 0 or 1 time (same as {0,1})

Quantifiers are "greedy" by default, which means they match the biggest pattern allowed. For example, if a text file contains the string "aaaaaa", the regular expression "a{2,5}" will find one "aaaaa" match rather than three "aa" matches. To indicate that a minimal size match is required, the minimal symbol "?" can follow any quantifier, so if "aa" is desired over "aaaaa", the regular expression must be specified as "a{2,5}?".

Escaped Characters

Escaped characters are

- \n - New line
- \r - Carriage return
- \t - Tab
- \f - Form feed
- \e - Escape
- \007 - Any Octal ASCII value
- \x7f - Any Hexadecimal ASCII value
- \cx - Control-X

Example 19-1. 1

```
Example1: The regular expression "velocity = 500\n" will match any occurrence
of the string "velocity = 500" directly followed by a new line character.
```

Character Classes

Character classes are used to specify a set of characters to match within a regular expression. [a-zA-Z] will march any character, [0-9] will match any single digit. By placing a ^ as the first element in the brackets, a negated character class is created, for example: [^A-Z] matches any character other than an upper-case letter. Common character classes have their own predefined symbol. Common Character Classes:

- \d - A digit
- \D - A non-digit
- \w - An alphanumeric character
- \W - A non-alphanumeric character
- \s - A white space character
- \S - A non-white space character
- . - Any character except a new line character

Example 19-2. 1

```
Example 1: The regular expression "velocity =(.)+\n" will match any line that
starts with the expression "velocity=" regardless of what follows it until a
new line character is found.
```

Example 19-3. 2

```
Example 2: To match the name "John Smith" within a file without knowing if a
middle name will occur, the regular expression  "John(.)+Smith" can be used.
```

Parenthesized Patterns

When more than one character needs to be quantified, a quantifier can be preceded by a pattern between parentheses. For example: "(abc\n){5}" will match any occurrence of 5 lines each containing the string "abc" directly followed be a new line character. Without the parentheses, the quantifier {5} would be

quantifying the "\n" character only. Parentheses can be used anywhere within a regular expression in order to group characters.

Logical OR

Any 2 or more constructs within a regular expression can be joined with a logical OR symbol to specify that either of the constructs can be matched. A "|" is used for the logical OR.

Example 19-4. 1

```
The regular expression "Velocity = (500)|(501)" will match both "Velocity =
500" or "Velocity = 501".
```

Example 19-5. 2

```
To match a section in a text file that starts with the string "BEGIN" and
ends with the string "END", the regular expression "BEGIN(.|\n)*?END" is
used. "(.|\n)*" specifies that any character including an new line character
that exists between "BEGIN" and "END" is accepted. The minimal "?" symbol is
used to make sure that the match will stop at the first occurrence of "END"
after a "BEGIN" is found because a "*" quantifier is "greedy" as described in
the quantifiers section above.
```

*** General Note Any character that is considered a special symbol by regular expressions must be escaped in order to be matched. For example, if the character "*" needs to be matched, it must be specified as "*" since "*" is used as a quantifier. Same for "+", "{" , "}", "[" ...

Classes and Methods

parsing-extract-pattern-class [class]

Parses a file and extracts results based on string patterns provided. The following are the input properties associated with this class (the last two are output properties).

Properties:

- **Pattern-begin**

  Start pattern string to match in input file. Values accepted are:

  - a string to be matched exactly
  - a regular expression string to be matched (see Patterns Section)
  - nil to start extracting at beginning of file

- **Pattern-end**

  End pattern string to match in input file. Values accepted are:

  - 'eof
  - a string to be matched exactly
  - a regular expression string to be matched (see Patterns Section)
  - nil

The parsing result is derived mainly according to the value of pattern-begin and pattern-end as follows:

1. pattern-begin = nil and pattern-end is a string: Extract from beginning of file to pattern-end.
2. patern-begin is a string and pattern-end = 'eof: Extract from pattern-begin to end of file.
3. both pattern-begin and pattern-end are strings: Extract between pattern-begin and pattern-end.
4. pattern-begin is a string and pattern-end is nil: Extract pattern-begin.

- **Output-include-patterns?**

  When this property is non-nil, the patterns (pattern-begin and pattern-end) will be included in the output result.

- **Input-file-name**

  Path and name string of the file to parse.

- **Output-file-name**

  File path and name string specifying the file where the parsing results are stored. Default formula appends ".out" to the input-file-name string.

- **Output-file-action**

  This property specifies the behavior with respect to the output file in case the file specified by output-file-name already exists. It can take two values:

  - 'Overwrite - Overwrite the output file if it exists with the new parsing results.
  - 'Append - Append the parsing results to the output file if it exists.

- **Accept-existing-output?**

  When this property is non-nil and the output file specified already exists, the parsing program will abort on its first attempt to run with this object (i.e. the first time the property run-parsing is demanded). This property is particularly useful for saved models to avoid redundant parsing when the model is retrieved.

- **Extract-all?**

  When non-nil, all matches are extracted, not just the first one. In other words, when more than one section of the file matches the desired parsing input, all matching sections are extracted. Default formula is nil. A new line character is inserted in the output file between every match.

- **Output-header**

  String to be inserted at the top of the output file as a file header.

  Default Formula: nil

- **Output-footer**

  String to be inserted at the end of the output file as a file footer.

  Default Formula: nil

- **Output-prefix**

  String to be inserted in the output file at the beginning of every match.

  Default Formula: ""

- **Output-suffix**

  String to be inserted in the output file at the end of every match.

  Default Formula: ""

- **Run-parsing**

  Executes the parsing procedure and writes the result to the output file.

- **Output**

  Reads the parsing result from the output file into a string.

parsing-extract-lines-class [class]

Parses a file and extracts the desired lines into an output file. The following are the input properties associated with this class (the last two are output properties).

Properties:

- **Line-index-begin**

  Index of first line to extract. Starts at 0.

  Default Formula: (default 0)

- **Line-index-end**

  Index of last line to extract. Starts at 0. Can also be equal to 'last to specify the last line.

- **Input-file-name**

  Path and name string of the file to parse.

- **Output-file-name**

  File path and name string specifying the file where the parsing results are stored.

  Default Formula: Default formula appends ".out" to the input-file-name string.

- **Output-file-action**

  This property specifies the behavior with respect to the output file in case the file specified by output-file-name already exists. It can take two values:

  - 'Overwrite - Overwrite the output file if it exists with the new parsing results.
  - 'Append - Append the parsing results to the output file if it exists.

- **Accept-existing-output?**

  When this property is non-nil and the output file specified already exists, the parsing program will abort on its first attempt to run with this object (i.e. the first time the property run-parsing is demanded). This property is particularly useful for saved models to avoid redundant parsing when the model is retrieved.

- **Output-header**

  String to be inserted at the top of the output file as a file header.

  Default Formula: (default nil)

- **Output-footer**

String to be inserted at the end of the output file as a file footer.

Default Formula: (default nil)

- **Output-prefix**

  String to be inserted in the output file at the beginning of every line.

  Default Formula: (default "")

- **Output-suffix**

  String to be inserted in the output file at the end of every line.

  Default Formula: (default "")

- **Run-parsing**

  Executes the parsing procedure and writes the result to the output file.

- **Output**

  Reads the parsing result from the output file into a string.

parsing-extract-columns-class [class]

Parses a file and extracts the desired columns into an output file. The following are the input properties associated with this class (the last two are output properties).

Properties:

- **Column-index-begin**

  Index of the first column to extract.

  Default Formula: (default 0)

- **Column-index-end**

  Index of the last column to extract. Starts at 0. Can also be equal to 'last to specify the last column.

  Default Formula: (default 0)

- **Column-delimiter**

  Delimiter defining columns in input file. Can be 'space, 'tab, 'comma or any string.

  Default Formula: (default 'space)

- **Input-file-name**

  Path and name string of the file to parse.

- **Output-file-name**

  File path and name string specifying the file where the parsing results are stored.

  Default Formula: Default formula appends ".out" to the input-file-name string.

- **Output-file-action**

  This property specifies the behavior with respect to the output file in case the file specified by output-file-name already exists. It can take two values:

  - 'Overwrite - Overwrite the output file if it exists with the new parsing results.
  - 'Append - Append the parsing results to the output file if it exists.

- **Accept-existing-output?**

  When this property is non-nil and the output file specified already exists, the parsing program will abort on its first attempt to run with this object (i.e. the first time the property run-parsing is demanded). This property is particularly useful for saved models to avoid redundant parsing when the model is retrieved.

- **Output-header**

  String to be inserted at the top of the output file as a file header.

  Default Formula: (default nil)

- **Output-footer**

  String to be inserted at the end of the output file as a file footer.

  Default Formula: (default nil)

- **Output-prefix**

  String to be inserted in the output file at the beginning of every line

  Default Formula: (default "")

- **Output-suffix**

  String to be inserted in the output file at the end of every line.

  Default Formula: (default "")

- **Output-column-prefix**

  Prefix string to be inserted in the output file at every column.

  Default Formula: (default "")

- **Output-column-suffix**

  Suffix string to be inserted in the output file at every column.

  Default Formula: (default "")

- **Run-parsing**

  Executes the parsing procedure and writes the result to the output file.

- **Output**

  Reads the parsing result from the output file into a string.

parsing-extract-token-class [class]

Retrieves a token or a set of tokens from a file. Tokens are defined to be the entities delimited by the token-delimiter provided. Consecutive delimiters are considered as one delimiter. The following are the input properties associated with this class (the last two are output properties).

Properties:

- **Token-index**

  Index of the token to extract. Tokens are counted based on the number of token-delimiter encountered in the file. For example if token-index is 2 and 'space is used as the token delimiter, the string delimited by the 2nd and third space character of the file is considered.

Default Formula: (default 0)

- **Column-delimiter**

  Delimiter between tokens of the data file. Can be space, tab, comma, new-line or any string.

  Default Formula: (default 'space)

- **Input-file-name**

  Path and name string of the file to parse.

- **Output-file-name**

  File path and name string specifying the file where the parsing results are stored.

  Default Formula: Default formula appends ".out" to the input-file-name string.

- **Output-file-action**

  This property specifies the behavior with respect to the output file in case the file specified by output-file-name already exists. It can take two values:

  - 'Overwrite - Overwrite the output file if it exists with the new parsing results.
  - 'Append - Append the parsing results to the output file if it exists.

- **Accept-existing-output?**

  When this property is non-nil and the output file specified already exists, the parsing program will abort on its first attempt to run with this object (i.e. the first time the property run-parsing is demanded). This property is particularly useful for saved models to avoid redundant parsing when the model is retrieved.

- **Extract-action**

  This property specifies the desired extraction behavior and can take the following values:

  - 'Self - Extracts the token considered.
  - 'After - Extracts all tokens of the input file that fall after the token considered. A new-line character is inserted before every token extracted in the output file.
  - 'Before - Extracts all tokens of the input file that fall before the token considered. A new-line character is inserted before every token extracted in the output file.

  Default Formula: (default 'self)

- **Output-header**

  String to be inserted at the top of the output file as a file header.

  Default Formula: (default nil)

- **Output-footer**

  String to be inserted at the end of the output file as a file footer.

  Default Formula: (default nil)

- **Output-prefix**

  String to be inserted in the output file at the beginning of every line

  Default Formula: (default "")

- **Output-suffix**

  String to be inserted in the output file at the end of every line.

  Default Formula: (default "")

- **Run-parsing**

  Executes the parsing procedure and writes the result to the output file.

- **Output**

  Reads the parsing result from the output file into a string.

parsing-extract-variable-class [class]

This class extracts a variable value from a text file given a variable name. A search is conducted in the input file for the variable name pattern provided. A "value" corresponding to that variable name is extracted. A variable entry in the input file is defined as follows: (variable-name)(delimiter1)(variable-value)(delimiter2), where delimiter1 separates a variable name from its value and delimiter2 marks the end of a variable value. A variable name pattern can occur more than once in the input file; the match-index provided indicates the desired occurrence index. The following are the input properties associated with this class (the last one is an output property).

Properties:

- **Variable-name-pattern**

  Pattern string specifying the variable name to match in the input file. See Patterns Section.

  Default Formula: (default nil)

- **Match-index**

  Desired occurrence index of the variable-name-pattern in the input file This is useful when there is more than one match for the variable name pattern provided. A value of 0 would extract the value corresponding to the first match, 1 would consider the second match and so on...

  Default Formula: (default 0)

- **Delimiter1**

  Delimiter separating the variable-name-pattern and its corresponding value. Can be space, tab, comma, new-line or any string of length 1.

  Default Formula: (default 'space)

- **Delimiter2**

  Delimiter marking the end of a variable value. Can be space, tab, comma, new-line or any string of length 1.

  Default Formula: (default 'space)

- **String-value?**

  This property controls the type of the output property value. When nil, the value retrieved is converted into its appropriate AML type; otherwise, it is double-quoted as a string. In other words it will be converted into a type corresponding to the way it appeared in the input file. Examples: The value 5 (as it occurs in the input file) is retrieved as "5" if this property is non-nil, as 5 otherwise. The value cylinder (as it occurs in the input file) is retrieved as "cylinder" if this property is non-nil, as 'cylinder

otherwise. The value "High Pressure" (as it occurs in the input file) is retrieved as "\"High Pressure\"" if this property is non-nil, as "High Pressure" otherwise.

- **Input-file-name**

  Path and name string of the file to parse.

- **Output-file-name**

  File path and name string specifying the file where the parsing results are stored.

  Default Formula: Default formula appends ".out" to the input-file-name string.

- **Output-file-action**

  This property specifies the behavior with respect to the output file in case the file specified by output-file-name already exists. It can take two values:

  - 'Overwrite - Overwrite the output file if it exists with the new parsing results.
  - 'Append - Append the parsing results to the output file if it exists.

- **Accept-existing-output?**

  When this property is non-nil and the output file specified already exists, the parsing program will abort on its first attempt to run with this object (i.e. the first time the property run-parsing is demanded). This property is particularly useful for saved models to avoid redundant parsing when the model is retrieved.

- **Output**

  Holds the retrieved "value" of the desired variable. The data type of the property value depends on the string-value? property described above.

parsing-list-from-string-mixin [class]

This class generates a list of tokens from a string based on the format-list specified. The following are the input properties associated with this class (the last one is an output property).

Inheritance:

Properties:

- **Source-string**

  Source string.

- **Format-list**

  Format of the tokens in the source string. The source string is assumed to be a string of tokens and this property defines these token by specifying the type and size (in characters) of every token. This is specified as a consecutive list of couples (one couple per token). Every couple is of the form: '(string n), '(real n n), '(integer n), or '(symbol n), where n is the number of characters of the token. If no n is specified or if n is not an integer, the remainder of the string is taken. Example: (list '(string 3) '(integer 2)) will generate the list '("abc" 56) out of the source string "abc56".

  Default Formula: (default (list '(symbol n)))

- **Ignore-periods?**

  When non-nil, periods in the string will be removed prior to the parsing operation.

Default Formula: (default 'space)

- **Output-list**

  Resulting list of tokens.

parsing-replace-pattern-class [class]

This class searches for a pattern in a file and replaces it with a user-specified string (replacement-string). The result is stored in the output file. The input file is not modified unless the property modify-input-file? is non-nil. Pattern replacement is based on the following: If both pattern-begin and pattern-end are pattern strings, the replacement string will replace the text that occurs between the two pattern matches. If pattern-begin is a pattern string and pattern-end is nil, the replacement-string will replace the text that matches pattern-begin. If more than one match of pattern-begin/pattern-end occurs in the input file, all matches are replaced by the replacement string. The following are the input properties associated with this class (the last one is an output property).

Properties:

- **Input-file-name**

  File path and name string specifying the original source file.

  Default Formula: (default nil)

- **Output-file-name**

  File path and name string specifying the output file.

  Default Formula: Default formula appends ".out" to the input-file-name string.

- **Output-file-action**

  Default formula is 'overwrite to overwrite the output file if it exists. Can also take the value of 'append to append to the output file.

  Default Formula: 'overwrite

- **Pattern-begin**

  Pattern string. See Patterns Section.

- **Pattern-end**

  Pattern string. See Patterns Section. This property can also be nil.

  Default Formula: (default nil)

- **Replacement-string**

  String to replace desired text in input file.

- **Modify-input-file?**

  When non-nil, the input file is also updated based on the parsing results.

  Default Formula: (default nil)

- **Input-file-backup-name**

  Path and name string specifying the name of the original input backup file. This property is used when modify-input-file? is non-nil.

Default Formula: Default formula adds the extension ".bak" to the input file name string.

- **Run-parsing**

  Executes the parsing procedure and generates/updates the output file

parsing-replace-pattern-list-class [class]

This class searches for a set of patterns in a file and replaces them with their corresponding string specified in the replacement-string-list property. The result is stored in the output file. The input file is not modified unless the property modify-input-file? is non-nil. The patterns are specified by the property pattern-list. Each of these patterns, when a match is found in the input file, is replaced by its respective string specified in replacement-string-list. If more than on match for a particular pattern exists, all matches of that pattern are replaced. The following are the input properties associated with this class (the last one is an output property).

Properties:

- **Input-file-name**

  File path and name string specifying the original source file.

  Default Formula: (default nil)

- **Output-file-name**

  File path and name string specifying the output file.

  Default Formula: Default formula appends ".out" to the input-file-name string.

- **Output-file-action**

  Default formula is 'overwrite to overwrite the output file if it exists. Can also take the value of 'append to append to the output file.

  Default Formula: 'overwrite

- **Pattern-list**

  List of pattern strings. See Patterns Section for pattern string generation.

- **Replacement-string-list**

  List of replacement string.

- **Modify-input-file?**

  When non-nil, the input file is also updated based on the parsing results.

  Default Formula: (default nil)

- **Input-file-backup-name**

  Path and name string specifying the name of the original input backup file. This property is used when modify-input-file? is non-nil.

  Default Formula: Default formula adds the extension ".bak" to the input file name string.

- **Run-parsing**

  Executes the parsing procedure and generates/updates the output file

parsing-table-class [class]

This class inherits from the AML class record-table-object. It loads a data (text) file into a record table. Records represent either the column data or row data of the file depending on the orientation property. This class is typically used to manage the output of a standard parsing class. The table created consists of records. Each record will hold the values (strings) of a row or a column from the file depending on the orientation property. The first field of the record (The primary key) is a unique integer identifying the record. (Refer to record-table-object in the AML reference manual to learn more). The second field of the record is the variable name. Variable names are input specified by the user in the variable-name-list property and are mainly used by the retrieve-variable method described below. If the user does not specify variable names, the first value of a row/column in the file will be considered the variable name. Empty lines in the file are always skipped. The following are the input properties associated with this class (the last two are output properties).

Inheritance:

Properties:

- **File-name**

  Data file path and name string.

  Default Formula: (default nil)

- **Orientation**

  Can be any of the following two values

  1. 'horizontal - table records will be generated from the rows of the data file.
  2. 'vertical - table records will be generated from the columns of the data file.

  Default Formula: 'horizontal

- **Fixed-width-columns?**

  When non-nil, columns in the data file are specified by their width in number of characters. Otherwise, columns are identified using a delimiter character or string.

- **Column-delimiter**

  Delimiter separating columns in the data file. Can be space, tab, comma or a string of length 1. This property is effective only if fixed-width-columns? is nil.

  Default Formula: (default 'space)

- **Ignore-consecutive-delimiters?**

  When non-nil, ignores consecutive column delimiters and treats them as one. Otherwise, treats consecutive delimiters as an occurrence of a blank value. Blank values are stored as nil.

  Default Formula: (default nil)

- **Column-width-list**

  List of integers specifying the width (in number of characters) of the columns of the data file. When all columns have the same width, this property can take an integer value instead of a list of integers specifying this width. This property is ignored when fixed-width-columns? is nil.

  Default Formula: (default (list 0))

- **Variable-name-list**

  Optional feature that allows the user to specify a list of variables names to be stored in the second field of each record. These variable names are used as record identifiers by the method retrieve-variable described below. When this property is nil, the first value of the desired column/row in the file will be considered the variable name. Variable names can be strings or symbols.

  Default Formula: (default nil)

- **Start-indices**

  List '(row-index column-index) specifying the row and column of the file where the table should start reading data. This property is used to skip initial rows and columns in the file.

  Default Formula: Default formula is (list 0 0) which will read the whole file.

- **Output**

  List of list representing the records retrieved from the parsing results table. Except for debugging purposes, it is recommended not to demand this property in order to avoid generating the entire table in the form of a list of lists, which can use a lot of memory. Instead, use the select-table method on this class to query the table content or the method retrieve-variable described below.

- **Record-count**

  Integer specifying the number of records retrieved.

retrieve-variable [Method]

Defined on Classes:

parsing-table-class

Returns a value associated with a variable in the table. A value is a field in a record corresponding to the variable name specified. Format: (retrieve-variable instance variable-name [:index 0] [:record? Nil] [:string? t] [:force? Nil])

Arguments:

- **Instance**

  Instance of parsing-table-class.

- **Variable-name**

  Name of the desired variable. A variable name is a string or a symbol.

Keyword Arguments:

- **Index**

  The value 0 specifies the first value of the desired variable, the value 1 is the second and so on. As a reminder, values corresponding to a variable are stored in records and a specified variable-name can correspond to more than one record in the table. When more than one record match the variable name specified, the index treats values as a member of a list constructed by appending all records.

  Default Value: 0

- **String?**

  Optional keyword specifying if values should be turned into a number or a symbol before they are returned. When non-nil, values are returned as is.

- **Record?**

  Optional keyword that allows retrieving a whole record (list of values) corresponding to variable-name. When non-nil, the whole record corresponding to variable-name is returned. In that case, the index keyword is that of the record since a variable-name can correspond to more than one record in the table.

  Default Value: nil

- **Force?**

  When non-nil, resets the table and rereads the source data file before retrieving the desired value(s).

  Default Value: nil

parsing-extract-patterns-table-class [class]

This class parses the input file for the desired pattern, stores the result in the parsing output file, and uses the parsing output file as an input to the table.

Inheritance: parsing-table-class , parsing-extract-pattern-class.

Properties:

- **File-name**

  inherited from parsing-table-class

  Default Formula: set to the output-file-name inherited from parsing-extract-pattern-class

Appendix A. AML Debugger

This appendix describes how to use the AML debugger

Appendix B. Color Names

```
aliceblue       antiquewhite    antiquewhite1   antiquewhite2   antiquewhite3
antiquewhite4
aquamarine      aquamarine1     aquamarine2     aquamarine3     aquamarine4
azure
azure1          azure2          azure3          azure4          beige
bisque
bisque1         bisque2         bisque3         bisque4         black
blanchedalmond
blue            blue1           blue2           blue3           blue4
blueviolet
brown           brown1          brown2          brown3          brown4
burlywood
burlywood1      burlywood2      burlywood3      burlywood4      cadetblue
cadetblue1
cadetblue2      cadetblue3      cadetblue4      chartreuse      chartreuse1
chartreuse2
chartreuse3     chartreuse4     chocolate       chocolate1      chocolate2
chocolate3
chocolate4      coral           coral1          coral2          coral3
coral4
cornflowerblue  cornsilk        cornsilk1       cornsilk2       cornsilk3
cornsilk4
cyan            cyan1           cyan2           cyan3           cyan4
darkgoldenrod
darkgoldenrod1  darkgoldenrod2  darkgoldenrod3  darkgoldenrod4  darkgreen
darkkhaki
darkolivegreen  darkolivegreen1 darkolivegreen2 darkolivegreen3
darkolivegreen4 darkorange
darkorange1     darkorange2     darkorange3     darkorange4     darkorchid
darkorchid1
darkorchid2     darkorchid3     darkorchid4     darksalmon      darkseagreen
darkseagreen1
darkseagreen2   darkseagreen3   darkseagreen4   darkslateblue   darkslategray
darkslategray1
darkslategray2  darkslategray3  darkslategray4  darkslategrey   darkturquoise
darkviolet
deeppink        deeppink1       deeppink2       deeppink3       deeppink4
deepskyblue
deepskyblue1    deepskyblue2    deepskyblue3    deepskyblue4    dimgray
dimgrey
dodgerblue      dodgerblue1     dodgerblue2     dodgerblue3     dodgerblue4
firebrick
firebrick1      firebrick2      firebrick3      firebrick4      floralwhite
forestgreen
gainsboro       ghostwhite      gold            gold1           gold2
gold3
```

```
gold4           goldenrod       goldenrod1      goldenrod2      goldenrod3
goldenrod4
gray            gray0           gray1           gray10          gray100
gray11
gray12          gray13          gray14          gray15          gray16
gray17
gray18          gray19          gray2           gray20          gray21
gray22
gray23          gray24          gray25          gray26          gray27
gray28
gray29          gray3           gray30          gray31          gray32
gray33
gray34          gray35          gray36          gray37          gray38
gray39
gray4           gray40          gray41          gray42          gray43
gray44
gray45          gray46          gray47          gray48          gray49
gray5
gray50          gray51          gray52          gray53          gray54
gray55
gray56          gray57          gray58          gray59          gray6
gray60
gray61          gray62          gray63          gray64          gray65
gray66
gray67          gray68          gray69          gray7           gray70
gray71
gray72          gray73          gray74          gray75          gray76
gray77
gray78          gray79          gray8           gray80          gray81
gray82
gray83          gray84          gray85          gray86          gray87
gray88
gray89          gray9           gray90          gray91          gray92
gray93
gray94          gray95          gray96          gray97          gray98
gray99
green           green1          green2          green3          green4
greenyellow
grey            grey0           grey1           grey10          grey100
grey11
grey12          grey13          grey14          grey15          grey16
grey17
grey18          grey19          grey2           grey20          grey21
grey22
grey23          grey24          grey25          grey26          grey27
grey28
grey29          grey3           grey30          grey31          grey32
grey33
grey34          grey35          grey36          grey37          grey38
grey39
grey4           grey40          grey41          grey42          grey43
grey44
```

```
grey45          grey46          grey47          grey48          grey49
grey5
grey50          grey51          grey52          grey53          grey54
grey55
grey56          grey57          grey58          grey59          grey6
grey60
grey61          grey62          grey63          grey64          grey65
grey66
grey67          grey68          grey69          grey7           grey70
grey71
grey72          grey73          grey74          grey75          grey76
grey77
grey78          grey79          grey8           grey80          grey81
grey82
grey83          grey84          grey85          grey86          grey87
grey88
grey89          grey9           grey90          grey91          grey92
grey93
grey94          grey95          grey96          grey97          grey98
grey99
honeydew        honeydew1       honeydew2       honeydew3       honeydew4
hotpink
hotpink1        hotpink2        hotpink3        hotpink4        indianred
indianred1
indianred2      indianred3      indianred4      ivory           ivory1
ivory2
ivory3          ivory4          khaki           khaki1          khaki2
khaki3
khaki4          lavender        lavenderblush   lavenderblush1
lavenderblush2  lavenderblush3
lavenderblush4  lawngreen       lemonchiffon    lemonchiffon1   lemonchiffon2
lemonchiffon3
lemonchiffon4   lightblue       lightblue1      lightblue2      lightblue3
lightblue4
lightcoral      lightcyan       lightcyan1      lightcyan2      lightcyan3
lightcyan4
lightgoldenrod  lightgoldenrod  lightgoldenrod1 lightgoldenrod2
lightgoldenrod3 lightgoldenrod4
lightgray       lightgrey       lightpink       lightpink1      lightpink2
lightpink3
lightpink4      lightsalmon     lightsalmon1    lightsalmon2    lightsalmon3
lightsalmon4
lightseagreen   lightskyblue    lightskyblue1   lightskyblue2   lightskyblue3
lightskyblue4
lightslateblue  lightslategray  lightslategrey  lightsteelblue
lightsteelblue1 lightsteelblue2
lightsteelblue3 lightsteelblue4 lightyellow     lightyellow1    lightyellow2
lightyellow3
lightyellow4    limegreen       linen           magenta         magenta1
magenta2
magenta3        magenta4        maroon          maroon1         maroon2
maroon3
```

```
maroon4         mediumaquamarine mediumblue     mediumorchid    mediumorchid1
mediumorchid2
mediumorchid3   mediumorchid4   mediumpurple    mediumpurple1   mediumpurple2
mediumpurple3
mediumpurple4  mediumseagreen  mediumslateblue mediumspringgreen
mediumturquoise mediumvioletred
midnightblue    mintcream       mistyrose       mistyrose1      mistyrose2
mistyrose3
mistyrose4      moccasin        navajowhite     navajowhite1    navajowhite2
navajowhite3
navajowhite4    navy            navyblue        oldlace         olivedrab
olivedrab1
olivedrab2      olivedrab3      olivedrab4      orange          orange1
orange2
orange3         orange4         orangered       orangered1      orangered2
orangered3
orangered4      orchid          orchid1         orchid2         orchid3
orchid4
palegoldenrod   palegreen       palegreen1      palegreen2      palegreen3
palegreen4
paleturquoise   paleturquoise1  paleturquoise2  paleturquoise3
paleturquoise4  palevioletred
palevioletred1  palevioletred2  palevioletred3  palevioletred4  papayawhip
peachpuff
peachpuff1      peachpuff2      peachpuff3      peachpuff4      peru
pink
pink1           pink2           pink3           pink4           plum
plum1
plum2           plum3           plum4           powderblue      purple
purple1
purple2         purple3         purple4         red             red1
red2
red3            red4            rosybrown       rosybrown1      rosybrown2
rosybrown3
rosybrown4      royalblue       royalblue1      royalblue2      royalblue3
royalblue4
saddlebrown     salmon          salmon1         salmon2         salmon3
salmon4
sandybrown      seagreen        seagreen1       seagreen2       seagreen3
seagreen4
seashell        seashell1       seashell2       seashell3       seashell4
sienna
sienna1         sienna2         sienna3         sienna4         skyblue
skyblue1
skyblue2        skyblue3        skyblue4        slateblue       slateblue1
slateblue2
slateblue3      slateblue4      slategray       slategray1      slategray2
slategray3
slategray4      slategrey       snow            snow1           snow2
snow3
snow4           springgreen     springgreen1    springgreen2    springgreen3
springgreen4
```

```
steelblue       steelblue1      steelblue2      steelblue3      steelblue4
tan
tan1            tan2            tan3            tan4            thistle
thistle1
thistle2        thistle3        thistle4        tomato          tomato1
tomato2
tomato3         tomato4         turquoise       turquoise1      turquoise2
turquoise3
turquoise4      violet          violetred       violetred1      violetred2
violetred3
violetred4      wheat           wheat1          wheat2          wheat3
wheat4
white           whitesmoke      yellow          yellow          yellow1
yellow2
yellow3         yellow4         yellowgreen
```

Appendix C. VGL Functions

VGL Introduction

This appendix is a listing of the Virtual Geometry Layer (VGL) functions and their parameters but does not contains complete documentaton of VGL. VGL functions are called from within AML, but are written as a layer over a C/C++ based layer. These functions let the user create models with full solid/surface/mixed dimensional/non-manifold geometry. The geometry entities that the VGL deals with are referred to as geoms. Geoms are identified with their unique geom ID. VGL functions are further incorporated into the various geometric objects such as union-object, scale-object, box-object, rotational-sweep-object etc.Detailed documentation on the functions can be obtained from the related objects. For example, detailed documentation on vgl::box-geom can be obtained from box-object.

To invoke VGL functions, use the following format:

```
(vgl::<function-name> arg1 arg2 .....))

;; Example:
> (setf box (vgl::box-geom 1.0 2.0 3.0)
;; returns : 124214233
```

Note:

Certain VGL functions are destructive in that they destroy some or all of the geoms that are passed as arguments to the function. All of the functions in the booleans section are destructive. For example, these functions include replace-sub-geoms, union-geom, union-geoms, intersection-geom, intersection-geoms, difference-geom, difference-geoms, and sewn-geom. In order to not delete the original geoms, the user is encouraged to use vgl::copy-geom. These functions do not contain any memory management functionality, so the user needs to delete any geoms used for construction geometry unless the are automatically destroyed from a destructive function call.

Modeler Functions

Functions

end-modeler [Function]

Terminates the solid modeler.

get-tol-value [Function]

Returns the current solid modeler tolerance.

set-tol-value [Function]

tolerance model-size Sets the solid modeler tolerance. The default modeler tolerance and model size are 0.00001 and 1.0. It is suggested that these values not be modified unless the default values fail. Set the model size to be a little larger than the largest dimension of the model.

solid-modeler-name [Function]

Returns the name of the solid modeler currently being used

start-modeler [Function]

filename Initializes the solid modeler.

Primitives

Functions

arc-geom [Function]

Returns a geom id for a arc geom

arc-geom-from-arc-info [Function]

center radius angle normal start-vector This function takes a set of arguments corresponding to the standard AML arc definition and create an arc geom from these argument. Returns the created geom id.

bezier-geom [Function]

dim degrees points &key rat-p Returns a geom id for bezier geom

box-geom [Function]

length width height &key (solid? nil) Returns a geom id for a box geom

circle-geom [Function]

radius Returns a geom id for a circle geom

cone-geom [Function]

radius height &key (capped? nil) (solid? nil) Returns a geom id for a cone geom

cyclide-geom [Function]

Arguments:

- **a**

  double-float

- **c**

  double-float

- **m**

  double-float

- **theta1**

  double-float

- **theta2**

  double-float

- **phi**

  double-float

- **phi2**

  double-float

- **dim**

  integer

cylinder-geom [Function]

radius height &key (capped? nil) (solid? nil) Returns a geom id for a cylinder geom

disc-geom [Function]

radius Returns a geom id for a disc geom

ellipsoid-geom [Function]

xrad yrad zrad &key (solid? nil) Returns a geom id for an ellipsoid geom

interpolated-curve-geom [Function]

points params &key derivs Returns a geom id for interpolated curve

interpolated-surface-geom [Function]

un vn points uprms vprms &key udr vdr uvdr Returns a geom id for interpolated surface

line-geom [Function]

p1 p2 This returns a geom id for a line segment between point p1 and point p2 where p1 and p2 are specified in (x y z) format.

nurb-geom [Function]

dim rat-p hmg-p degrees knots control-points knot-tol Returns a geom id for nurb geom

point-geom [Function]

coords Returns a geom id for point geom and nil if the coords is nil.

sheet-geom [Function]

length width Returns a geom id for a sheet geom

skin-curve-geom [Function]

points Returns a geom id for curve geom by skinning over a set of points

sphere-geom [Function]

radius &key (solid? nil) Returns a geom id for a sphere geom

surface-geom-skin-from-crvmsh [Function]

ucurves usize vcurves vsize pnt-tol ang-tol avg-factor Returns a geom id from skinning over a mesh of curves

surface-geom-skin-from-curves [Function]

curves size pnt-tol ang-tol u-map-param-values Returns a geom id from skinning over a set of curves

surface-geom-skin-from-points-grid [Function]

points-list Returns a geom id from skinning over a grid of points

surface-skin-curves-with-mesh-geom [Function]

curves npts pnttol ang-tol Returns a geom id from skinning over a set of curves

Geom Construction

Functions

assembly-geom [Function]

geoms geoms is specified as a list of geom ids, the function returns a geom that assembles all the geoms together.

bounded-geom [Function]

geom-list dimension &key copy? normal This function creates a geom as follows: When given a list of coplanar 1 dimensional geoms (curves, lines, ...) that form a closed loop, it will create a surface geom bounded by these geoms. When given a list of surface geoms that surround a closed volume, it will create a 3-dimensional solid geom bounded by these surface geoms. The function returns the geom Id created.

capped-surface-geom [Function]

geom &key solid? This function provides the vgl functionality for capped-surface-object. It takes a geom of something like an open-cylinder, detects its edges that can be capped, caps them, and creates a closed volume, and makes it solid if the solid? flag is t. Usage Notes: a) This only works if the edges to be capped are PLANAR. b) The function is non-destructive and does not effect the input geom.

Example C-1. Capped-surface-geom Example

```
(add-object (The) 'cyl 'open-cylinder-object)
(vgl::k-sub-geoms (the cyl geom) 2) ;; Returns: (87386)
(vgl::geom-dimension (the cyl geom)) ;; Returns: 2
(setf capped (vgl::capped-surface-geom (the cyl geom)))
(vgl::k-sub-geoms capped 2) ;; Returns (87574 87578 87582)
(vgl::geom-dimension capped) ;; Returns 3
```

As shown in this exampl above, 2 additional (cap) faces have been added and the geom is a solid.

complement-halfspace-geom [Function]

geom Returns a geom id for complement of geom

copy-geom [Function]

geom Returns a geom id of the copy of the input geom

copy-sub-geom [Function]

geom

create-polyline-in-surface-geom [Function]

surface-geom points &key normal tolerance Create a polyline curve geom in the domain space of a surface and returns the geom ID of the polyline.

curve-in-surface-geom [Function]

surface-geom points &key normal parampoints smooth? tolerance dimension Create a curve in the domain space of a surface and returns the geom ID of the curve.

disassemble-geom [Function]

geom Returns a list of AML compatible geoms from a Parasolid ASSEMBLE type geom. This function only works with the Parasolid-based solid-modeler.

extrusion-geom [Function]

Returns a geom id for an extruded geom

Arguments:

- **geom**

  Input geom id.

- **vector**

  A list of 3 float numbers identifying a direction of extrusion.

- **dist**

  Extrusion distance

Note: This only works for geoms of dimension < 3.

halfspace-geom [Function]

geom Returns a geom id for halfspaced geom

mirror-geom [Function]

geom mirror-pnt basis &optional (axis-basis-p t) Returns a geom id for mirror of an input geom

replace-sub-geoms [Function]

geom list-of-geoms Replaces the loops of a surface geom with the geoms given in list-of-geoms. This returns a new geom and is destructive.

rotation-sweep-geom [Function]

swept-geom radians axis-point axis-dir Returns a geom id for a rotation sweep geom

sewn-geom [Function]

geoms &optional (maxdim 3) The argument geoms is specified as a list of geom ids, the function returns a geom that sews all the geoms together. Note that the geoms must not touch except at boundaries.

tangential-sweep-geom [Function]

swept-geom path-geom ref-point Returns a geom id for a tangential sweep geom

translation-sweep-geom [Function]

swept-geom path-geom Returns a geom id for a translation sweep geom

Geom Queries

Functions

basic-geom [Function]

geom This function returns a list of basic sub geoms for a geom.

copy-map [Function]

map The argument "map" is the underlying geometry (point, curve or surface).

create-manifold-topology [Function]

This function returns (body topol-list) or (error (fault-index fault)).

Arguments:

- **type-list**

  List of :body :shell :face :loop :edge :vertex or :region :fin (used in creating general topology).

- **relation-list**

List of (parent-index child-index sense), where sense should be 1 (positive), -1 (negative) and 0 (no sense).

- **dimension**

  This value is 0, 1, 2 or 3

create-general-topology [Function]

Arguments:

- **type-list**

  List of :body :shell :face :loop :edge :vertex or :region :fin (used in creating general topology).

- **relation-list**

  List of (parent-index child-index sense), where sense should be 1 (positive), -1 (negative) and 0 (no sense).

- **type**

  0 -- minimum body

  1 -- wire body

  2 -- sheet body

  3 -- solid body

  4 -- general body

  5 -- acorn body

curve-find-min-radius [Function]

curve :param-range This function returns the minimum radius of curvature of the given curve, its position and parameter: (radius position param) param-range needs to be given as (min max), otherwise the whole param range of the curve is the default.

curve-eval-derivatives-at-param [Function]

geom maxOrd param This function computes the value of the derivative at the parameter specified. geom: curve geom. maxOrd: number of derivatives. param: parameter value where the derivatives will be computed.

edge-type [Function]

The function format is: vgl::edge-type(edge-geom). This returns one of the following types for a given edge-geom (subgeom 1): :line :arc :ellipse :nurb :intersection :foreign ;;; foreign geometry :paramcurve :trimcurve :constant-paramcurve ;;isoparameter curve

geom-get-map [Function]

subgeom

face-normal-at-vertex [Function]

geom vertex Returns normal vector to geom when provided a geom id and a vertex id. This usually is relevant only for a 2 dimensional geom.

geom-center [Function]

geom Returns center of geom as a point (x y z).

geom-check [Function]

geom &key filename errors? The function takes as input a geom id, and returns the number of errors found in the geom. Any errors found are also printed out. When filename is given, the output from geom-check will be written to the given file. When errors? is t, vgl::geom-check returns a list of error codes, otherwise, number of errors is returned (-1 means the given geom does not exist). When filename is given, the error information will be written to the given file, otherwise it prints to the stdout.

geom-curvature-radius-at-point [Function]

geom point This function returns the tangent, principal normal, binormal and curvature radius of a 'curve' at the given position: (curvature-radius tangent principal-normal binormal) or returns the normal, principal directions and principal curvature radii of a surface at a given positon: (curvature-radii-uv normal u-direction v-direction).

geom-dimension [Function]

geom Returns dimension of geom.

geom-minmax-box [Function]

geom This returns the bounding box for the input geom as a list of 3 lists, ((xmin xmax) (ymin ymax) (zmin zmax)). The function is defined in the global frame and therefore does not garrantee the smallest box that can contain the geom.

geom-normal [Function]

geom vertex &key underlying-geom Returns normal to geom

geom-print [Function]

geom This prints out the internal data structure of a geom.

geom-tangent-at-point [Function]

geom point

geom-volume [Function]

geom Returns volume of geom provided. If the geom is of dimension 1, the function returns length, for dimension 2, it returns surface area, for dimension 3, it returns solid volume.

get-edge-nurb-data [Function]

edge &key dist-tol This function returns nurb information from an edge. It will return nil when it fails to get the nurb definition for a given curve. The user may need to increase the value of :dist-tol and try again.

get-surface-nurb-data [Function]

edge &key dist-tol This function returns nurb information from a surface. It will return nil when it fails to get the nurb definition for a given surface. The user may need to increase the value of :dist-tol and try again.

get-surface-revolution-data [Function]

This function returns a list of (curve location axis), where curve is the swept curve, location is the axis point and the axis defines the rotation axis. Please note that the curve is actually the underlying geometry not the topology entity (geom). The user needs to call the function vgl::curve-make-wire-body to convert the geometry to topology (geom).

Arguments:

- s
urface
- This should be a geom or sub geom with surface type as :revolution

get-surface-extrusion-data [Function]

This function returns a list of (curve vector), where curve is the swept curve, vector is the extrusion vector. Please note that the curve is actually the underlying geometry not the topology entity (geom). The user needs to call the function vgl::curve-make-wire-body to convert the geometry to topology (geom).

Arguments:

- s
urface
- This should be a geom or sub geom with surface type as :extrusion

intersect-geom-p [Function]

geom1 geom2 &optional (tolerance 0.01) Returns t if geom1 and geom2 intersect within the tolerance provided.

is-face-periodic [Function]

face-geom Returns a couple (2-element list) of t or nil if the face is periodic in the u and v direction respectively. This function only works with the Parasolid-base solid modeler.

is-simple-geom [Function]

geom Returns t if the geom provided is a simple geom, i.e. a geom that was created by the native AML simple geometry engine without using the solid modeler.

k-limits [Function]

geom dim Returns ptr to array of k-limits of a geom. It returns NULL/0 if unsuccessful.

k-sub-geoms [Function]

geom dim This returns a list of sub geoms of a geom of the required dimension, i.e dim=0 returns vertices, 1 returns edges, 2 returns faces, 3 returns solid bodies. It returns 0 if unsuccessful.

map-type [Function]

map The argument "map" is the underlying geometry (point, curve or surface).

null-geom-p [Function]

geom Returns (t or nil) if geom is null or not, respectively.

offset-geom [Function]

geom distance Returns offset-geom to geom provided.

sup-geoms [Function]

geom Returns ptr to array of sup-geoms of a geom. It returns NULL/0 if unsuccessful.

surface-find-min-radii [Function]

face :param-range This function returns the minimum radius of curvature on each side of the given face (in a given param-range), its position and u-v parameters: (radius-list position-list params-list). param-range needs to be given as (minu maxu minv maxv), otherwise the whole param range of the face is the default.

surface-normal-at-point [Function]

geom point :point-on-face? Returns a normal vector to the surface geom provided a geom id and a list of points coordinates. Point has to lie exactly on the surface. When point-on-face? keyword is t, and the geom is single face and the point lies on the face, the normal calculation are done faster.

topology-attach-geometry [Function]

Arguments:

- **geom**

  The geom object.

- **topology-geometry-list**

  This is a list of (topology geometry-map sense), where sense is only used for face/surface, should be 1 (posistive) or -1 (negative).

vertex-coords [Function]

vertex Returns the coordinate list of the vertex geom provided.

vertex-normal [Function]

vertex &optional (ug 0) Returns normal to vertex.

vertex-on-geom [Function]

geom point &optional (tolerance 100.0) This returns a vertex id for a vertex on the geom at the specified point. Note that the point needs to lie within the specified tolerance of the geom, and does not need to lie on the geom. The vertex that is created by this function will not be freed from memory automatically and will need to be deleted.

vertex-params [Function]

vertex This returns a list (u v w) of the coordinates of the vertex in parameter space. Note that for a geom of dimension 0 or 1, only the u value is relevant, for a geom of dimension 2, the u and the v values are relevant, and none are relevant for a solid geom...

vertex-tangent [Function]

vertex Returns tangent to vertex.

The following functions are used to create underlying geometry map (curve, surface, sheet,...) directly without creating the geom body (topology). The arguments used are the same as their regular geom conterpart.

Note: The resulting geometry can not be drawn since it does not have any topology.

point-geom-map

line-geom-map

sheet-geom-map

arc-geom-map

circle-geom-map

cylinder-geom-map

cone-geom-map

sphere-geom-map

ellipsoid-geom-map

nurb-geom-map

ellipse-geom-map

torus-geom-map

Geom Transformations

Functions

invert-geom [Function]

geom

orient-geom [Function]

geom &key underlying-geom Returns a geom id for the oriented geom. The original geom is modified

region-make-hollow [Function]

This function makes a given solid region hollow.

Arguments:

- **region**

  The subgeom 3 entity to make hollow.

rotate-geom [Function]

geom axis-pts &key (deg 0.0) The input geom is rotated about the axis specified as a line given by 2 points '((x1 y1 z1) (x2 y2 z2). Note that the original geom is modified.

scale-geom [Function]

geom factor &optional (point (geom-center geom)) Returns a geom id for the scaled geom. The original geom is modified.

scale-nonuniform-geom [Function]

geom factor &optional (point (geom-center geom)) Returns a geom id for the scaled geom. The Original geom is modified

translate-geom [Function]

geom vector The input geom is translated by the vector specified as (x y z) and returned. Note that the original geom is modified.

Geom Deletion

Functions

delete-geom [Function]

geom Deletes the input geom and frees the memory associated with the geom.

free-id [Function]

geom This function searches the objects drawn in the active display to determine if the geom given equal the geom of one of the drawn objects. If this is the case, it is deleted from the graphics display. This function calls delete-geom on the given geom before exiting.

Boolean Operations

Note that all the boolean operations are destructive, the input geoms may be destroyed and should be copied before the boolean operation if they are to be reused.

Functions

classify-geoms [Function]

geom1 geom2 cl1 cl2 &optional (simplify-mode 0) Creates an array of classified geoms.

difference-geom [Function]

geom1 geom2 &optional (simplify-mode 0) This returns the result of geom2 being subtracted from geom1. simplify-mode=0 implies no simplification and simplify-mode=36 is full simplification.

difference-geoms [Function]

geoms &optional (simplify-mode 0)

intersection-geom [Function]

geom1 geom2 &optional (simplify-mode 0) This returns the intersection of geom1 and geom2. simplify-mode=0 implies no simplification and simplify-mode=36 is full simplification.

intersection-geoms [Function]

geoms &optional (simplify-mode 0) This returns the intersection of all the geoms specified in the list geoms. simplify-mode=0 implies no simplification and simplify-mode=36 is full simplification.

trim-geom [Function]

geom-to-trim-with, geom-to-trim, point Returns the geom ID of the trimmed geom.

union-geom [Function]

geom1 geom2 &optional (simplify-mode 0) This returns the union of the 2 geoms. simplify-mode=0 implies no simplification and simplifymode=36 is full simplification.

union-geoms [Function]

geoms &optional (simplify-mode 0)

Facets

Functions

facet-vertices [Function]

facet Generates an array of vertices for input facet.

geoms-faceted-min-dist [Function]

geom1 geom2 &optional (min-tol 0.0) (max-tol 999999999.99999) Returns min distance between 2 geoms

geom-facets [Function]

geom &key (point-tol 0.0) (ang-tol 0.0) Generates array of facets for input geom

geom-facets-by-edge-length [Function]

geom &key (edge-length 1.0) Generates array of facets for input geom based on edge length

geom-facets-minmax-box [Function]

geom Returns facets minmax box of geom. This function is an approximation and depends heavily on the faceting settings of the geometry. Please see GEOM-MINMAX-BOX above. The function does not garrantee the smallest box possible.

geom-release-facets [Function]

geom Releases and frees memory of the facets generated for the geom provided.

get-angular-facet-tolerance [Function]

This function is used to query the facetting tolerance for graphics on an angular-facet. Negative values mean the setting is disabled.

get-curve-angular-facet-tolerance [Function]

This function returns the tolerance of the curve-angular-facet.

get-curve-edgelength-facet-tolerance [Function]

This function returns the tolerance of the curve-edgelength-facet.

get-curve-point-facet-tolerance [Function]

This function returns the tolerance of the curve-point-facet.

get-edgelength-facet-tolerance [Function]

This function is used to query the facetting tolerance for graphics on an edgelength-facet. Negative values mean the setting is disabled.

get-point-facet-tolerance [Function]

This function is used to query the facetting tolerance for graphics on an point-facet. Negative values mean the setting is disabled.

get-vgl-facet-method [Function]

get-vgl-facet-value [Function]

This returns the angular tolerance used for faceting.

set-angular-facet-tolerance [Function]

This function sets the tolerance of the angular-facet. A negative value will disable the setting.

Arguments:

- **tolerance**

  The desired tolerance value.

set-curve-angular-facet-tolerance [Function]

This function sets the tolerance of the curve-angular-facet.

Arguments:

- **tolerance**

  The desired tolerance value.

set-curve-edgelength-facet-tolerance [Function]

This function sets the tolerance of the curve-edgelength-facet.

Arguments:

- **tolerance**

  The desired tolerance.

set-curve-point-facet-tolerance [Function]

This function returns the tolerance of the curve-point-facet.

Arguments:

- **tolerance**

  The desired tolerance.

set-edgelength-facet-tolerance [Function]

This function sets the tolerance of the edgelength-facet. A negative value will disable the setting.

Arguments:

- **tolerance**

  The desired tolerance value.

set-point-facet-tolerance [Function]

This function sets the tolerance of the point-facet. A negative value will disable the setting.

Arguments:

- **tolerance**

  The desired tolerance value.

set-vgl-facet-method [Function]

rough?

set-vgl-facet-value [Function]

value Takes a real value that specifies the vgl angular tolerance.Faceting is controlled using the angular tolerance specified here. Faceting controls how fine the facets for shaded/faceted rendering will be.

Geom Saving and Retrieving

Functions

retrieve-geom [Function]

file-name Retrieves geom from specified file and returns the geom id. A complete path for the file can be specified.

save-geom [Function]

geom file &optional (logical-path :home) (mode 1) Saves the geom to the specified file. The logical-path determines where the file will be saved. All geoms are saved in binary format if using the AML SHAPES-base solid modeler and in ascii if using the AML Parasolid-based solid modeler. Geoms saved under one solid modeler cannot be retrieved using the other solid modeler.

Parameter Functions

Functions

cell-eval-pnt-for-prms [Function]

geom params Returns point on geom for specified params

find-cell-prms-range [Function]

geom Returns params range for geom

generate-points-on-curve-geom [Function]

geom n

geom-map-eval-normal-space [Function]

geom params Returns normal on map for specified params

Higher-Level Functions

Functions

circular-array-geom [Function]

geom n center ref-point diameter start-angle repeat-angle rotate-clones? translate-axis rotate-axis assembly? Returns a geom id for assembly of circular array

linear-array-geom [Function]

geom n start-d spacing vector start-point ref-point assembly? Returns a geom id for assembly of linear array

offset-profile-geom [Function]

profile-geom distance vector

surface-halfspace-geom [Function]

surface-geom point

wrap-around-geom [Function]

geom1 geom2

Ray-Intersection Functions

Functions

delete-ray-comps [Function]

raycomp

geom-intersect-ray [Function]

geom ray Returns a geom id.

ray-comp-get-coords [Function]

raycomp

ray-comp-get-dim [Function]

raycomp Returns an integer. (0 for a point, 1 for a curve).

ray-comp-get-ends [Function]

raycomp

ray-get-nth [Function]

raycomp n Returns a ray component id.

Appendix D. Glossary

Class

A class is a template that defines an entity "type" with properties and subobjects. A class is meant to be instantiated into an object. A class can be instantiated an indefinite number of times. Methods are also defined for a class and considered part of the character of the class even though they are not defined within the class. A class can be thought of as a recipe for creating instances. For example the same cake recipe may be used many times, and each time, a different cake instance is created.

Demand-Driven Calculation

Demand driven calculation means that properties and subobjects are not created until the first time they are needed/demanded. If a property is not referred to (demanded) it will not calculate its value. The first time a property is demanded the value is calculated and that value is retained by the property. Subsequent demanding of the same property will not cause re-calculation because the first calculation is retained, unless the value of that property has been smashed. A property value is smashed when other properties or objects that it depends on are changed or smashed.

Dependency

A property is said to depend on another property if its formula (or any method/function called from the formula) references that other property. When a depndency is established, the property value smashes anytime the property it depends on changes value, formula or smashes.

Dependency Backtracking

Dependency backtracking is the mechanism that causes properties to recalculate their retained values (from their formula) when other properties that were used to calculate the value change. That is if volume is calculated by multiplying height by width by depth and the height, width, or depth change, the volume property value will be smashed (forgotten). The value will not be calculated until it is needed again because of demand driven calculations.

Expanding

Expandingan AML object is the action of demanding all its subobjects, and can also include expanding the subobjects too. When an instance of a class is created, if this class contains defined subobjects, the suobject instances are not created until referenced/demanded.

Inheritance

Inheritance is a mechanism for class (code) reuse. Through inheritance a class will have the same properties, subobjects, and methods as the class (superclass) that is being inherited from. The tree structure created by class inheritance is referred to as the class hierarchy.

Instance

An instance is an object that is created using the class definition. All instances of the same class have the same properties although the values of the properties may be different depending on the operations that have been performed on the instances. For example each time a cake is baked makes an instance of the cake recipe which could be thought of as a class. Throughout the manual, the expression "instance of Class-A" means an instance of Class-A or of any other class that directly or indirectly inherit from Class-A.

Method

A method is an operation or function that is defined specifically for a class. A method name may have definitions for many classes that must perform the same operation but in different ways. For example suppose the method volume is defined for each of the classes box, cylinder, and sphere. By calling the volume method with an instance of one of those classes the correct code will be executed automatically and the volume of the instance returned. Inheriting from a class that has methods defined for it will also inherit the methods.

Object

An object is a term that is used to refer to an instance of a class and all of the methods of that class and its superclasses. It may be considered to be a "packet" of software characterized by a set of operations (methods), a set of variables (properties) to store the results of operations, and a set of objects (subobjects) that define the structure. In fact an object oriented system does not have objects, but instead classes and instances. Defining an object is really defining a class and making an object is really creating an instance.

Overloading

Overloading refers to having the same property or method name defined for more than one class. Defining a volume method for each of the classes box, cylinder, and sphere is overloading of the method volume.

Overriding

Overriding is a special case of overloading that refers to having a property or method name defined for classes that are superclass and subclass with respect to each other. When this is the case the method or property defined at the lowest subclass will be retained. For example consider that a basketball class has a volume method defined that calculates the volume of material and it inherits from sphere that has a

volume method defined for total volume. Then an instance of a basketball will calculate the volume of the material and not the total volume. In this case the basketball volume method overrides the sphere volume method. The same is true for properties.

Smashing

Smashing is the event of marking a property value invalid or "unbound". A smashed property value will be recalculated from its formula when needed.

Subclass

Subclass is a term that refers to any class that inherits from a given class. Subclasses are thought of as down the class hierarchy. For example the class baseball is a subclass of a class sphere.

Superclass

Superclass is a term that refers to any class that is inherited by a given class. A Superclass of a class C is a class that can be found by tracing up the class hierarchy of C. For example the class sphere is a superclass of the class baseball.

Chapter 20. Index Chapter

Index

Index of functions

Index of methods

Index of classes