EOEntity

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	EOAccess/EOEntity.h

Class Description

An EOEntity describes a table, file or collection in a database, and associates a name internal to the Framework with an external name—by which the table is known to the database. An EOEntity maintains a group of attributes and relationships, which are collectively called properties. These are represented by the EOAttribute and EORelationship classes, respectively; see their specifications for more information.

You usually define entities in your EOModel with the EOModeler application, which is documented in the *Enterprise Objects Framework Developer's Guide*. EOEntity objects are primarily used by the Enterprise Objects Framework for mapping tables in the database to enterprise objects; your code will probably make limited use of them unless you're specifically working with models.

An EOEntity is associated with a specific class whose instances are used to represent records (rows) from the database in applications using layers at or above the database layer of the Enterprise Objects Framework. If an EOEntity doesn't have a specific class associated with it, instances of EOGenericRecord are created.

An EOEntity may be marked as read-only, in which case any changes to rows or objects for that entity made by the database level objects are denied.

You can define an external query for an EOEntity to be used when a selection is attempted with an unrestricted qualifier (one that would select all rows in the entity's table). An external query is sent unaltered to the database server and so can use database-specific features such as stored procedures; external queries are thus useful for hiding records or invoking database-specific features. You can also assign stored procedures to be invoked upon particular database operations through the use of EOEntity's **setStoredProcedure:forOperation:** method.

Like the other major modeling classes, EOEntity provides a user dictionary for your application to store any application-specific information related to the entity.

Class Properties, Primary Keys, and Locking

An EOEntity's EOAttributes are usually intended to have their values fetched from the database and presented to enterprise object. However, there may be times when you want to define an attribute that your enterprise objects never see; for example, you may have a special attribute used only for locking records

during updates that means nothing to your objects. To allow this masking of attributes, the EOEntity can define an EOAttribute as a class property, meaning that an EODatabaseChannel hands enterprise objects the value for the attribute whenever data is fetched for the object. (Note that "class property" here has a more specific meaning than the usual definition of "any quality or characteristic of a class.") If an EOAttribute isn't defined as a class property, the database channel never hands its values to enterprise objects. You use the **setClassProperties:** method to establish the set of attributes used as class properties.

In addition to being defined as a class property, an EOEntity can define an EOAttribute as being part of its primary key. The values for the attributes composing a primary key must uniquely identify their enterprise object (and hence a row in the entity's table). You define an EOEntity's primary key attributes with the **setPrimaryKeyAttributes:** method. Note that a primary key is usually not a class property.

Finally, to support locking of records during updating, an EOEntity defines a set of attributes to be recorded in snapshots any time an enterprise object is fetched. These attributes are compared when an object is updated according to the descriptions in the EODatabaseContext class specification. You establish the set of attributes used for locking with the **setAttributesUsedForLocking:** method. Note that an attribute used for locking doesn't have to be a class property; it will be recorded in the snapshot, but the enterprise object won't see it.

EOEntity provides methods for checking whether an attribute can be used in a primary key or for locking. Only attributes corresponding directly to a column in the database server's table are eligible for these purposes—flattened and derived attributes can't be used as a primary key or for locking. The methods for checking are **isValidPrimaryKeyAttribute:** and **isValidAttributeUsedForLocking:**. You can also check whether a given property can be defined as a class property with the **isValidClassProperty:** method.

Creating an Entity

An EOEntity requires at least the following to be usable:

- A name
- The name of a table in the database (the external name)
- The name of an enterprise object class
- A set of attributes to be used as the primary key

Note that if an entity has no enterprise object class name, the database-level objects use EOGenericRecord. This code excerpt gives an example of creating an EOEntity and adding it to an EOModel:

```
EOModel *myModel;          /* Assume this exists. */
NSArray *keyAttributes;    /* Assume this exists. */
EOEntity *employeeEntity;
BOOL result;

employeeEntity = [[[EOEntity alloc] init] autorelease];
[employeeEntity setName:@"employee"];
[employeeEntity setExternalName:@"EMPLOYEE"];
[employeeEntity setClassName:@"Employee"];
```

```
/* Create at least the primary key attributes. */
result = [employeeEntity setPrimaryKeyAttributes:keyAttributes];

/* Add the entity to the model. */
[myModel addEntity:employeeEntity];
```

Method Types

Setting the name	– beautifyName – name – setName: – validateName:
Getting the model	– model
Getting a qualifier	– isQualifierForPrimaryKey: – qualifierForPrimaryKey: – restrictingQualifier – setRestrictingQualifier:
Accessing attributes	– addAttribute: – anyAttributeNamed: – attributeNamed: – attributes – removeAttribute:
Accessing relationships	– addRelationship: – anyRelationshipNamed: – relationships – relationshipNamed: – removeRelationship:
Checking referential integrity	– externalModelsReferenced – referencesProperty:
Getting primary keys	– globalIDForRow: – isPrimaryKeyValidInObject: – primaryKeyForGlobalID: – primaryKeyForRow:
Getting primary key attributes	– isValidPrimaryKeyAttribute: – primaryKeyAttributeNames – primaryKeyAttributes – primaryKeyRootName – setPrimaryKeyAttributes:

Setting class properties	– classProperties – classPropertyNames – setClassProperties:
Checking whether a property is valid	– isValidClassProperty:
Setting the enterprise object class	– classDescriptionForInstances – className – setClassName:
Setting locking attributes	– attributesUsedForLocking – isValidAttributeUsedForLocking: – setAttributesUsedForLocking:
Setting external information	– externalName – externalQuery – setExternalName: – setExternalQuery:
Setting status	– isAbstractEntity – isReadOnly – setIsAbstractEntity: – setReadOnly:
Setting the user dictionary	– setUserInfo: – userInfo
Working with stored procedures	– setStoredProcedure:forOperation: – storedProcedureForOperation:
Working with entity inheritance hierarchies	– addSubEntity: – parentEntity – removeSubEntity: – subEntities
Specifying fault behavior	– maxNumberOfInstancesToBatchFetch – setMaxNumberOfInstancesToBatchFetch:
Caching objects	– setCachesObjects: – cachesObjects

Instance Methods

addAttribute:

– (void)**addAttribute:**(EOAttribute *)*anAttribute*

Adds *anAttribute* to the receiver. Raises an NSInvalidArgumentException if *anAttribute*'s name is already in use by another attribute or relationship. Sets *anAttribute*'s entity to self.

See also: – **removeAttribute:**, – **attributeNamed:**

addRelationship:

– (void)**addRelationship:**(EORelationship *)*aRelationship*

Adds *aRelationship* to the receiver. Raises an NSInvalidArgumentException if *aRelationship*'s name is already in use by another attribute or relationship. Sets *aRelationship*'s entity to self.

See also: – **removeRelationship:**, – **relationshipNamed:**

addSubEntity:

– (void)**addSubEntity:**(EOEntity *)*child*

Causes the child entity to "inherit" from the receiver. This is the first step in setting up an inheritance hierarchy between entities.

See also: – **subEntities**

anyAttributeNamed:

– (EOAttribute *)**anyAttributeNamed:**(NSString *)*attributeName*

Returns the user-created attribute with the given name. If no such attribute exists, this method looks through the "hidden" attributes created by the Enterprise Objects Framework for one with the given name. If none is found, **nil** is returned. Hidden attributes are used for such things as primary keys on target entities of flattened attributes.

See also: – **attributeNamed:**, – **attributes**

anyRelationshipNamed:

– (EORelationship *)**anyRelationshipNamed:**(NSString *)*relationshipName*

Returns the user-created relationship with the given name. If none exists, this method looks through the "hidden" relationships created by the Enterprise Objects Framework for one with the given name. If none is found, **nil** is returned.

See also: – **relationshipNamed:**

attributeNamed:

– (EOAttribute *)**attributeNamed:**(NSString *)*attributeName*

Returns the attribute named *attributeName*, or **nil** if no such attribute exists.

See also: – **anyAttributeNamed:**, – **attributes**, – **relationshipNamed:**

attributes

– (NSArray *)**attributes**

Returns all of the receiver's attributes, or **nil** if the receiver has none.

See also: – **anyAttributeNamed:**, – **attributeNamed:**

attributesUsedForLocking

– (NSArray *)**attributesUsedForLocking**

Returns an array containing those properties whose values must match a snapshot any time a row is updated.

Attributes used for locking are those whose values are compared when a database-level object performs an update. When the database-level classes fetch an enterprise object, they cache these attributes' values in a snapshot. Later, when the enterprise object is updated, the values of these attributes in the object are checked with those in the snapshot—if they differ, the update fails. See the EODatabaseContext class specification for more information.

beautifyName

– (void)**beautifyName**

Makes the receiver's name conform to a standard convention. Names that conform to this style are all lower-case except for the initial letter of each embedded word other than the first, which is upper case. Thus, "NAME" becomes "name", and "FIRST_NAME" becomes "firstName".

See also: – **setName:**, – **validateName:**

cachesObjects

– (BOOL)**cachesObjects**

Returns YES if all of the objects from the receiver are to be cached in memory and queries are to be evaluated in-memory using this cache rather than in the database. This method should only be used for fairly small tables of read-only objects, since the first access to the receiver will trigger fetching the entire table. You should generally restrict this method to read-only entities to avoid cached data getting out of sync with database data. Also, you shouldn't use this method if your application will be making queries against the entity that can't be evaluated in memory.

See also: – **setCachesObjects:**

classDescriptionForInstances

– (EOClassDescription *)**classDescriptionForInstances**

Returns the EOClassDescription associated with the receiver. The EOClassDescription class provides a mechanism for extending classes by giving them access to the metadata contained in an EOModel (or another external source of information). In an application, EOClassDescriptions are registered on demand for the EOEntity on which an enterprise object is based. For more information, see the class specifications for EOClassDescription and EOEntityClassDescription.

className

– (NSString *)**className**

Returns the name of the enterprise object class associated with the receiver. When a row is fetched for the receiver by a database-level object, it's returned as an instance of this class. This class might not be present in the run-time system, and in fact your application may have to load it on demand. If your application doesn't load a class, EOGenericRecord is used.

An enterprise object class other than EOGenericRecord can be mapped to only one entity.

See also: – **databaseChannel:failedToLookupClassNamed:** (EODatabaseChannel, "Methods Implemented by the Delegate")

classProperties

– (NSArray *)**classProperties**

Returns an array containing the properties that are bound to the receiver's class (so that instances of the class will be passed values corresponding to those properties). This is a subset of the receiver's attributes and relationships.

See also: – **classPropertyNames**

classPropertyNames

– (NSArray *)**classPropertyNames**

Returns an array containing the names of those properties that are bound to the receiver's class (so that instances of the class will be passed values corresponding to those properties). This is a subset of the receiver's attributes and relationships.

See also: – **classProperties**.

externalModelsReferenced

– (NSArray *)**externalModelsReferenced**

Examines each of the receiver's relationships and returns a list of all external models referenced by the receiver.

See also: – **referencesProperty:**

externalName

– (NSString *)**externalName**

Returns the name of the receiver as understood by the database server.

externalQuery

– (NSString *)**externalQuery**

Returns a query statement that's used by an EOAdaptorChannel to select rows for the receiver when a qualifier is empty, or **nil** if the receiver has no external query. An empty qualifier is one that specifies only the entity, and would thus fetch all enterprise objects for that entity.

External queries are useful for hiding records or invoking database-specific features such as stored procedures when an application attempts to select all records for an entity. You can also use the

EOStoredProcedure class to work with stored procedures; for more information see the EOStoredProcedure class specification.

See also: – **isEmpty** (EOQualifier), – **setExternalQuery:**,
– **selectAttributes:describedByQualifier:fetchOrder:lock:** (EOAdaptorChannel)

globalIDForRow:

– (EOGlobalID *)**globalIDForRow:**(NSDictionary *)*aRow*

Constructs a global identifier from the specified row for the receiver.

See also: – **primaryKeyForGlobalID:**

isAbstractEntity

– (BOOL)**isAbstractEntity**

Returns YES to indicate that the receiver is abstract, NO otherwise. An abstract entity is one that has no corresponding enterprise objects in your application. Abstract entities are used to model inheritance relationships. For example, you might have a Person abstract entity that acts as the parent of Customer and Employee entities. Customer and Employee would inherit certain characteristics from Person (such as name and address attributes). However, though your application might have Customer and Employee objects, it would never have a Person object.

See also: – **setIsAbsractEntity:**

isPrimaryKeyValidInObject:

– (BOOL)**isPrimaryKeyValidInObject:**(id)*anObject*

Returns YES if every key attribute is present in *anObject* and has a value that is not **nil**. Returns NO otherwise. This method uses the key-value coding protocol so a dictionary may be substituted for *anObject*.

See also: – **primaryKeyForRow:**

isQualifierForPrimaryKey:

– (BOOL)**isQualifierForPrimaryKey:**(EOQualifier *)*aQualifier*

Returns YES if *aQualifier* describes the primary key and nothing but the primary key, NO otherwise.

isReadOnly

– (BOOL)**isReadOnly**

Returns YES if the receiver can't be modified, NO if it can. If an entity can't be modified, then enterprise objects fetched for that entity also can't be modified (that is, inserted, deleted, or updated).

isValidAttributeUsedForLocking:

– (BOOL)**isValidAttributeUsedForLocking:**(EOAttribute *)*anAttribute*

Returns NO if *anAttribute* isn't an EOAttribute, if the EOAttribute doesn't belong to the receiver, or if *anAttribute* is derived. Otherwise returns YES. An attribute that isn't valid for locking will cause **setAttributesUsedForLocking:** to fail.

See also: – **attributesUsedForLocking**

isValidClassProperty:

– (BOOL)**isValidClassProperty:**(id)*aProperty*

Returns NO if either *aProperty* isn't an EOAttribute or EORelationship, or if *aProperty* doesn't belong to the receiver. Otherwise returns YES. Note that this method doesn't tell you whether *aProperty* is a member of the array returned by **classProperties**. In other words, unlike **classProperties**, **classPropertyNames**, and **setClassProperties:**, this method doesn't interact with the properties bound to the entity's enterprise object class.

isValidPrimaryKeyAttribute:

– (BOOL)**isValidPrimaryKeyAttribute:**(EOAttribute *)*anAttribute*

Returns NO if *anAttribute* isn't an EOAttribute, doesn't belong to the receiver, or is derived. Otherwise returns YES.

See also: – **setPrimaryKeyAttributes:**

maxNumberOfInstancesToBatchFetch

– (unsigned int)**maxNumberOfInstancesToBatchFetch**

Returns the maximum number of to-one EOFaults from the receiver to fire at one time. See the method description for **setMaxNumberOfInstancesToBatchFetch:** for more explanation of what this means.

model

– (EOModel *)**model**

Returns the model that contains the receiver.

See also: **– addEntity:** (EOModel)

name

– (NSString *)**name**

Returns the receiver's name.

parentEntity

– (EOEntity *)**parentEntity**

Returns the entity from which the receiver inherits.

See also: – **subEntities**

primaryKeyAttributeNames

– (NSArray *)**primaryKeyAttributeNames**

Returns an array containing the names of the attributes that make up the receiver's primary key.

See also: – **primaryKeyAttributes**

primaryKeyAttributes

– (NSArray *)**primaryKeyAttributes**

Returns an array of those attributes that make up the receiver's primary key.

See also: – **primaryKeyAttributeNames**

primaryKeyForGlobalID:

– (NSDictionary *)**primaryKeyForGlobalID:**(EOKeyGlobalID *)*globalID*

Returns the primary key for the object identified by *globalID*.

See also: – **globalIDForRow:**

primaryKeyForRow:

– (NSDictionary *)**primaryKeyForRow:**(NSDictionary *)*aRow*

Returns the primary key for *aRow*, or **nil** if the primary key can't be computed. The primary key is an NSDictionary whose keys are attribute names and whose values are values for those attributes.

See also: – **primaryKeyForGlobalID:**

primaryKeyRootName:

– (NSString *)**primaryKeyRootName**

Returns the external name (that is, the name as it's understood by the database) of the receiver's root entity. If the receiver has no parent entity, returns the receiver's external name.

See also: – **externalName**, – **name**, – **parentEntity**

qualifierForPrimaryKey:

– (EOQualifier *)**qualifierForPrimaryKey:**(NSDictionary *)*aRow*

Returns a qualifier for the receiver that can be used to fetch an instance of the receiver with the primary key extracted from *aRow*.

See also: – **isQualifierForPrimaryKey:**, – **restrictingQualifier**

referencesProperty:

– (BOOL)**referencesProperty:**(id)*aProperty*

Returns YES if any of the receiver's attributes or relationships reference *aProperty*, NO otherwise. A property can be referenced by a flattened attribute or by a relationship. For example, suppose a model has an Employee entity with a **toDepartment** relationship. If you flatten the department's name attribute into the Employee entity, creating a **departmentName** attribute, that flattened attribute references the **toDepartment** relationship.

If an entity has any outstanding references to a property, you shouldn't remove the property.

See also: – **removeAttribute:**, – **removeRelationship:**

relationshipNamed:

– (EORelationship *)**relationshipNamed:**(NSString *)*name*

Returns the relationship named *name*, or **nil** if the receiver has no such relationship.

See also: – **anyRelationshipNamed:**, – **attributeNamed:**, – **relationships**

relationships

– (NSArray *)**relationships**

Returns all of the receiver's relationships, or **nil** if the receiver has none.

See also: – **attributes**

removeAttribute:

– (void)**removeAttribute:**(EOAttribute *)*name*

Removes the attribute named *name* if it exists. You should always use **referencesProperty:** to check that an attribute isn't referenced by another property before removing it.

See also: – **addAttribute:**, – **attributes**

removeRelationship:

– (void)**removeRelationship:**(EORelationship *)*name*

Removes the relationship named *name* if it exists. You should always use **referencesProperty:** to check that a relationship isn't referenced by another property before removing it.

See also: – **addRelationship:**, – **relationships**

removeSubEntity:

– (void)**removeSubEntity:**(EOEntity *)*child*

Removes *child* from the receiver's list of sub-entities.

See also: – **subEntities**

restrictingQualifier

– (EOQualifier *)**restrictingQualifier**

Returns the qualifier used to restrict all queries made against the receiver. Restricting qualifiers are useful when there is not a one-to-one mapping between an entity and a particular database table, or when you always want to filter the data that's returned for a particular entity.

For example, if you're using the "one table" inheritance model in which parent and child data is contained in the same table, you'd use a restricting qualifier to fetch objects of the appropriate type. To give a non-inheritance example, for an Employees table you might create a "Sales" entity that has a restricting qualifier that only fetches employees who are in the Sales department.

See also: – **setRestrictingQualifier:**

setAttributesUsedForLocking:

– (BOOL)**setAttributesUsedForLocking:**(NSArray *)*attributes*

Sets *attributes* as the attributes used when an EODatabaseChannel locks enterprise objects for updates. Returns NO and doesn't set the attributes used for locking if any of the attributes in *attributes* responds NO to **isValidAttributeUsedForLocking:**; returns YES otherwise. See the EODatabase, EODatabaseContext, and EODatabaseChannel class specifications for information on locking.

setCachesObjects:

– (void)**setCachesObjects:**(BOOL)*flag*

Sets according to *flag* whether all of the receiver's objects are cached the first time the associated table is queried.

See also: – **cachesObjects**

setClassName:

– (void)**setClassName:**(NSString *)*name*

Sets the name of the class associated with the receiver to *name*. This class need not be present in the run-time system when this message is sent. When an EODatabaseChannel fetches objects for the receiver, they're created as instances of this class. Your application may have to load the class on demand if it isn't present in the run-time system; if it doesn't load the class, EOGenericRecord will be used.

Note: If you set the class name to **nil**, the **className** method returns "EOGenericRecord".

An enterprise object class other than EOGenericRecord can be mapped to only one entity.

See also: – **className**, – **databaseChannel:failedToLookupClassNamed:** (EODatabaseChannel, "Methods Implemented by the Delegate")

setClassProperties:

– (BOOL)**setClassProperties:**(NSArray *)*properties*

Sets the receiver's class properties to the EOAttributes and EORelationships in *properties* and returns YES, unless the receiver responds NO to **isValidClassProperty:** for any of the objects in the array. In this event, the receiver's class properties aren't changed and NO is returned.

setExternalName:

– (void)**setExternalName:**(NSString *)*name*

Sets the name of the receiver as understood by the database server to *name*. For example, though your application may know the entity as "JobTitle" the database may require a form such as "JOB_TTL". An adaptor uses the external name to communicate with the database; your application should never need to use the external name.

setExternalQuery:

– (void)**setExternalQuery:**(NSString *)*aQuery*

Sets the query statement used for selecting rows from the receiver when there is no qualifier.

External queries are useful for hiding records or invoking database-specific features such as stored procedures when an application attempts to select all records for an entity. You can also use the EOStoredProcedure class to work with stored procedures; for more information see the EOStoredProcedure class specification.

An external query is sent unaltered to the database server, and so must contain the external (column) names instead of the names of EOAttributes. However, to work properly with the adaptor the external query must use the columns in alphabetical order by their corresponding EOAttributes' names.

See also: – **columnName** (EOAttribute), – **isEmpty** (EOQualifier), – **externalQuery**, – **selectAttributes:describedByQualifier:fetchOrder:lock:** (EOAdaptorChannel)

setIsAbstractEntity:

– (void)**setIsAbstractEntity:**(BOOL)*flag*

Sets according to *flag* whether the receiver is an abstract entity. For more discussion of abstract entities, see the method description for **isAbstractEntity**.

setMaxNumberOfInstancesToBatchFetch:

– (void)**setMaxNumberOfInstancesToBatchFetch:**(unsigned int)*size*

Sets the maximum number of EOFaults from the receiver to trigger at one time. By default, only one object is fetched from the database when you trigger an EOFault. You can optionally use this method to set to *size* the number of EOFaults of the same entity should be fetched from the database along with the first one. Using this technique helps to optimize performance by taking advantage of round trips to the database.

See also: – **maxNumberOfInstancesToBatchFetch**

setName:

– (void)**setName:**(NSString *)*name*

Sets the receiver's name to *name*. Raises an NSInvalidArgumentException if *name* is already in use by another entity in the same EOModel or if *name* is not a valid entity name.

See also: – **beautifyName:**, – **validateName:**

setPrimaryKeyAttributes:

– (BOOL)**setPrimaryKeyAttributes:**(NSArray *)*keys*

If the receiver responds NO to **isValidPrimaryKeyAttribute:** for any of the objects in the array, returns NO. Otherwise, sets the primary key attributes to the attributes in *keys* and returns YES.

You should exercise care in choosing primary key attributes. Floating-point numbers, for example, can't be reliably compared for equality, and are thus unsuitable for use in primary keys. Integer and string types are the safest choice for primary keys. NSDecimalNumbers will work, but they'll entail more overhead than integers.

See also: – **isValidPrimaryKeyAttribute:**

setReadOnly:

– (void)**setReadOnly:**(BOOL)*flag*

Sets according to *flag* whether the database rows for the receiver can be modified by the database level objects.

setRestrictingQualifier:

– (void)**setRestrictingQualifier:**(EOQualifier *)*qualifier*

Sets the qualifier used to restrict all queries made against the receiver. The restricting qualifier can be used to map an entity to a subset of the rows in a table. For more discussion of this subject, see the description for **restrictingQualifier**.

setStoredProcedure:forOperation:

– (void)**setStoredProcedure:**(EOStoredProcedure *)*storedProcedure*
forOperation:(NSString *)*operation*

Sets *storedProcedure* for *operation*. *operation* can be one of the following:

EOFetchAllProcedureOperation
EOFetchWithPrimaryKeyProcedureOperation
EOInsertProcedureOperation
EODeleteProcedureOperation
EONextPrimaryKeyProcedureOperation

This information is used when changes from the object graph have been transformed into EODatabaseOperations that are being used to construct EOAdaptorOperations. At this point, Enterprise Objects Framework checks the entities associated with the changed objects to see if the entities have any stored procedures defined for the operation being performed.

See also: – **storedProcedureForOperation:**

setUserInfo:

– (void)**setUserInfo:**(NSDictionary *)*dictionary*

Sets the *dictionary* of auxiliary data, which your application can use for whatever it needs. *dictionary* can only contain property list data types (that is, NSDictionaries, NSStrings, NSArrays, and NSDatas).

storedProcedureForOperation:

– (EOStoredProcedure *)**storedProcedureForOperation:**(NSString *)*operation*

Returns the stored procedure for the specified *operation*, if one has been set. Otherwise, returns **nil**.

See also: – **parameterDirection**: (EOEntity), – **storedProcedure** (EOEntity)

subEntities

– (NSArray *)**subEntities**

Returns a list of those entities which inherit from the receiver.

See also: – **parentEntity**

userInfo

– (NSDictionary *)**userInfo**

Returns a dictionary of user data. Your application can use this to store any auxiliary information it needs.

validateName:

– (NSException *)**validateName:**(NSString *)*name*

Validates *name* and returns **nil** if it is a valid name, or an exception if it isn't. A name is invalid if it has zero length; starts with a character other than a letter, a number, or "@", "#", or "_"; or contains a character other than a letter, a number, "@", "#", "_", or "$".

setName: uses this method to validate its argument.