EOAssociation

Inherits From: EODelayedObserver : NSObject

Conforms To: NSCoding
EOObserving (EODelayedObserver)
NSObject (NSObject)

Declared In: EOInterface/EOAssociation.h

Class at a Glance

Purpose

An EOAssociation maintains a two-way binding between the properties of a display object, typically a control, and the properties of one or more enterprise objects contained in one or more EODIsplayGroups. You typically set them up using Interface Builder, using the programmatic interface most often when defining subclasses. See these subclass specifications for information on the different kinds of EOAssociations:

- EOActionAssociation
- EOColumnAssociation
- EOControlAssociation
- EODetailSelectionAssociation
- EOMasterDetailAssociation
- EOPopUpAssociation
- EOActionInsertionAssociation
- EOTableViewAssociation
- EOActionCellAssociation
- EOPickTextAssociation
- EOMasterPeerAssociation
- EOTextAssociation

Principal Attributes

- Monitors a single display object
- Has multiple aspects that control state of the display object
- Can be bound to multiple EODisplayGroups

Creation

Interface Builder
– initWithObject: Designated initializer.

Class Description

EOAssociation defines the mechanism that transfers values between EODisplayGroups and the user interface of an application. An EOAssociation instance is tied to a single *display object*, a user-interface object or other kind of object that manages values intended for display. The EOAssociation takes over certain outlets of the display object and sets its value according to the selection in the EODisplayGroup. An EOAssociation also has various *aspects*, which define the different parameters of the display object that it controls, such as the value or values displayed and whether the display object is enabled or editable. Each aspect can be bound to an EODisplayGroup with a key denoting a property of the enterprise objects in the EODisplayGroup. The value or values of this property determine the value for the EOAssociation's aspect.

EOAssociation is an abstract class, defining only the general mechanism for binding display objects to EODisplayGroups. You always instantiate instances of its various subclasses, which define behavior specific to different kinds of display objects. See the listing in the Class at a Glance section for standard EOAssociation subclasses.

You normally set up EOAssociations using Interface Builder; "Setting up an EOAssociation Programmatically" below shows how to set them up in code. EOAssociation's programmatic interface is more important when defining custom EOAssociation subclasses. "Creating a Subclass of EOAssociation" discusses this process in detail.

How EOAssociations Work

An EOAssociation monitors its display object for changes in its value and for other events, and monitors its EODisplayGroups for changes in the selection or in the contents (the enterprise objects). A change at either end causes the EOAssociation to query the object at that end for whatever needs to be done, and to take action on the object at the other end. For example, when the selection changes in the EODisplayGroup, the EOAssociation gets the new value from the selected enterprise object and puts it in the display object. The following sections describe in detail how the EOAssociation works with the object at either end.

The Display Object

An EOAssociation is tied to a single display object, which owns it. Each EOAssociation subclass takes over a particular set of the display object's outlets, such as its target, delegate, or data source, and acts in that role by receiving the messages defined for it. For example, an EOControlAssociation sets itself as the target of its display object, an NSControl, and assigns the control's action method to a method that it implements. When the control is acted upon, it sends its action to the EOAssociation, which takes the control's value and sends it to the EODisplayGroup to update the selected enterprise object. An EOControlAssociation also sets itself as the control's delegate to receive various editing and validation delegate messages.

Because the EOAssociation takes over various outlets of its display object, these outlets can't be set or used for other purposes. EOAssociation's **objectKeysTaken** class method returns the names of these outlets. Interface Builder also disables them in its Connections Inspector when a display object has an EOAssociation assigned to it. For example, a button with an EOControlAssociation has its target outlet dimmed in the inspector.

Although display objects are typically user-interface objects such as text fields and pop-up lists, they can in fact be any kind of object. Two notable examples of non-interface display objects are EODisplayGroups in master-detail and master-peer configurations. These configurations result in the detail or peer EODisplayGroup displaying the destination values for a to-many relationship selected in the master EODisplayGroup. See the EOMasterDetailAssociation, EOMasterPeerAssociation, and EODetailSelectionAssociation class specifications for more information on master-detail and master-peer configurations.

Bindings: Aspects, EODisplayGroups, and Keys

Though an EOAssociation has only one display object, it can have any number of aspects, which define what the EOAssociation monitors in its EODisplayGroups. Aspects are bound to EODisplayGroups through keys describing class properties or other properties of the enterprise objects in the EODisplayGroups. Depending on the type of EOAssociation, there may be only one aspect or several aspects. Where there are several, they can be optional or mandatory; they may all have to be bound to a single EODisplayGroup; they may be bound to different EODisplayGroups; or they may be mutually exclusive, in that if one is used the other must not be used.

An aspect reflects a characteristic of the EOAssociation's display object: the value it displays, the list of possible values it can contain, whether it's enabled or editable, and so on. Most EOAssociations have a "value" aspect, for example. The aspect's value is determined by the values of enterprise object properties in the EODisplayGroup that the aspect is bound to. The value can be taken from all of the enterprise objects in the EODisplayGroup, or from just the selected ones. Some aspects are "read-only" from the EODisplayGroup, merely causing changes in the display object when the EODisplayGroup changes; others affect the properties of enterprise objects in the EODisplayGroup when the EOAssociation's display object changes.

For example, an EOControlAssociation defines the aspects "value" and "enabled". To set up a text field so that it displays the salary for the selected enterprise object, you create an EOControlAssociation for the text field, and bind its "value" aspect to the "salary" key of the EODisplayGroup containing the enterprise objects. You can also bind its "enabled" aspect to some key such as "eligibleForRaise", so that if this property has a non-zero value the user can edit the salary value in the text field. Then, when the user presses the Return key or otherwise finishes editing, the new value is sent to the EODisplayGroup.

A multi-valued aspect can represent the destination of a to-many relationship, or it can define the range of possible values for an individual object's property. One such EOAssociation subclass, EOPopUpAssociation, has a "titles" aspect that's set to all of the possible values for a given key; these values then make up the list of items in the pop-up list. It also has a "selectedObject" aspect, bound to an EODisplayGroup whose entities use those value, that selects the pop-up list item for the value of the selected enterprise object in the EODisplayGroup. Binding the "titles" aspect to the "name" key of a Department entity's EODisplayGroup, for example, results in a pop-up list containing the names of all departments.

EOAssociations register themselves with their EODisplayGroups in order to be notified of changes in their enterprise objects. When an EODisplayGroup changes its selection or contents, its EOAssociations are sent

a **subjectChanged** message. This message doesn't indicate which EODisplayGroup has changed, so the EOAssociation must query each of its EODisplayGroup as described below under "Monitoring Changes from the EODisplayGroup." When the EOAssociation needs to send a change to an EODisplayGroup, it typically does so by invoking **setValue:forAspect:**. This process is described under "Monitoring Changes from the Display Object."

Setting up an EOAssociation Programmatically

Though you normally set up EOAssociations with the Interface Builder application, you can do so programmatically as well. Because EOAssociations coordinate the actions of many objects, linking a display object to a control group is a multi-step process, as shown by this code fragment:

```
NSTextField *salaryField;        /* Assume this exists. */
EODisplayGroup *employeeGroup;   /* Assume this exists. */
EOControlAssociation *myAssoc;

myAssoc = [[EOControlAssociation alloc] initWithObject:salaryField];
[myAssoc bindAspect:@"value" displayGroup:employeeGroup key:@"salary"];
[myAssoc bindAspect:@"enabled" displayGroup:employeeGroup key:@"eligibleForRaise"];
[myAssoc establishConnection];
[myAssoc release];
```

This example first allocates an EOAssociation subclass and initializes it with the display object it monitors, in this case **salaryField**. At this point the EOAssociation is tenuously linked to **salaryField**; it hasn't appropriated any of the field's outlets, and hasn't been retained by the field. Before this can happen, the EOAssociation must have at least one aspect bound to an EODisplayGroup. This is accomplished by the two **bindAspect:displayGroup:key:** messages, which define the nature of the field's interaction with **employeeGroup**. Now all of the potential connections have been put in place, and an **establishConnection** message causes them to be confirmed: **salaryField** is made to assume **myAssoc** as its target and delegate, and **myAssoc** is registered as an observer of **employeeGroup**. At this point **myAssoc** is retained by the field, so it can be safely released.

Creating a Subclass of EOAssociation

If none of the standard EOAssociation subclasses meets your needs, you can create a new one without much effort. To do so, you need to define four areas of functionality:

- What your subclass monitors and which display objects is can work with.
- How your subclass establishes its connections with its display object and its EODisplayGroups.
- How it monitors the EODisplayGroups and updates the display object.
- How it monitors the display object and updates the EODisplayGroups.

The following four sections describe how to do each of these.

Defining Capabilities

A significant part of creating an EOAssociation subclass is defining and advertising what the subclass works with. This allows Interface Builder to make your subclass available in its Connections Inspector pop-up list, just like any other. The characteristics that your subclass should define are:

Aspects (required). Your EOAssociation subclass must define an **aspects** class method that returns an NSArray of aspect names, as NSStrings. Some standard aspects are:

Aspect Name	Use
value	The value of an attribute or relationship
enabled	Whether the control should be enabled
titles	All existing values for an attribute
selectedTitle	The value of the selected attribute (bound to the same key as “titles”)

What the subclass works with (required). Interface Builder asks each EOAssociation subclass if it can work with a given object when it displays its Connections Inspector. Your subclass should implement the **isUsableWithObject:** class method to examine the object provided and return YES if it can work with that object. This method can examine the class of the object provided, or any of its attributes, to determine whether it can work with the object. For example, EOPopUpAssociation verifies that the object is an NSPopUpButton, while EOMasterDetailAssociation checks that the object is an EODisplayGroup whose data source is an EODetailDataSource.

Aspect signatures (optional). Aspects by default are made available for any kind of property—single-valued attributes, to-one relationships, and to-many relationships. If your subclass has aspects that only have meaning for one or two of these, it should define an **aspectSignatures** class method that returns an NSArray of NSStrings corresponding to the aspects. Each string should contain a subset of the string “A1M”, where “A” indicates that the aspect can be used with attributes (where the value is a value-bearing object such as NSString or NSNumber), “1” that it can be used with to-one relationships (where the value is an enterprise object), and “M” indicates that the aspect can be used with to-many relationships (where the value is an array of enterprise objects). EOControlAssociation only displays single attributes, so its aspect signature for “value” and “enabled” is “A”, “A”. EOMasterDetailAssociation only works with relationships, so its aspect signature for “parent” is “1M”.

Which outlets it uses (optional). Interface Builder disables connections to outlets used by an EOAssociation, so if your subclass uses any it should advertise them by defining the **objectKeysTaken** class method to return an NSArray containing the names of the outlets. These are typically the standard “target”, “delegate”, “dataSource”, and so on.

EOAssociation classes superseded (optional). If your EOAssociation subclass applies uniquely to display objects that other kinds of EOAssociations simply happen to work with, it should implement the **associationClassesSuperseded** class method to return an array of these classes. EOPopUpAssociation, for example, works with EOPopUpButton, which as a subclass of NSControl is also eligible for the EOControlAssociation. Since this isn’t a meaningful or useful EOAssociation for a pop-up button, EOPopUpAssociation supersedes it, and Interface Builder doesn’t present it in its Connections Inspector when a pop-up button is selected.

Display name (optional). If you want your subclass to be listed in Interface Builder's Associations pop-up list with a name other than that of its class, it can override the **displayName** to return that name. This is often done to truncate long names so they fit in the pop-up button.

Primary aspect (optional). If your subclass implements the **primaryAspect** class method, Interface Builder automatically selects it the first time the user drags a connection from the display object and chooses your EOAssociation subclass in the Connections Inspector.

Binding ability (optional). If your subclass defines aspects that are mutually exclusive, available only for a particular kind of display object, or are otherwise not always available, you might want to implement the instance method **canBindAspect:displayGroup:key:** to check these types of conditions. Interface Builder uses this information to enable and disable aspects, to guide the user in property setting up EOAssociations.

Priority (optional). EOAssociation uses the default EODelayedObserver priority of EODelayedObserverPriorityThird. If your subclass need a higher or lower priority, it should override the **priority** method appropriately. EOMasterDetailAssociation, for example, uses EODelayedObserverPrioritySecond to catch updates before other EOAssociations based on it.

Setting Up

EOAssociation's designated initializer is **initWithObject:**, but you rarely need to override this method. Instead, you override **establishConnection**, which is where the real initialization takes place, as described above in "Setting up an EOAssociation Programmatically." Your implementation of this method should invoke **super**'s implementation to establish the aspect bindings, which makes the EOAssociation an observer of its EODisplayGroups, then set the outlets and other state of the display object (and any associated objects). This usually takes the form of **setTarget:**, **setDelegate:**, and other messages to plug in the outlets, along with messages such as **setAction:** to register a method that your EOAssociation subclass implements and that should be invoked by the display object. For display objects that work with others, such as NSControls and NSCells, your subclass might even need to get the companion object to set something.

Once your EOAssociation object has tied itself to its EODisplayGroups and its display object, it's ready to work.

Monitoring Changes from the EODisplayGroup

An EOAssociation is notified of changes in its EODisplayGroups through the **subjectChanged** method. This lets the EOAssociation object know that one or more of its EODisplayGroups has changed its selection or its contents, or both. Your EOAssociation can then query each of its EODisplayGroups for what's changed, and update its display object accordingly. To get the EODisplayGroup for a given aspect, you can use the **displayGroupForAspect:** method, and to get the key it's bound to, use **displayGroupKeyForAspect:**.

How your EOAssociation handles this message depends on whether the aspect represents a single value, as a text field does, or multiple values, as a table column or matrix does. A single-value aspect needs only to be updated with the new value. Your EOAssociation can do this by invoking its own **valueForAspect:**

method and passing that value to the display object in whatever way necessary, such as with a **setObjectValue:** or **setEnabled:** message.

A multi-valued aspect requires a bit more care for efficiency's sake. In this case, your EOAssociation should send the EODisplayGroup **contentsChanged** and **selectionChanged** messages to determine what has changed. If the contents have changed, your EOAssociation should reload all values for the aspect in question and update its display object accordingly. If only the selection has changed, your EOAssociation should simply get the selection indexes from the EODisplayGroup and update the selection in its display object, if appropriate. For aspects that don't reflect the selection, your EOAssociation need do nothing.

Monitoring Changes from the Display Object

Changes in the display object make their way to your EOAssociation through whatever messages are defined for the outlets the EOAssociation took over. A control EOAssociation sends its display object a **setAction:** message to register an action method, which is then invoked when the uses clicks the button, edits text and presses Return, or otherwise operates the control. This method should then perform whatever operation is necessary, whether that be to update a value in the EODisplayGroup or send a message to the EODisplayGroup or to its enterprise objects. For example, an EOAssociation used to display a value must update the value of the selected enterprise object by invoking **setValue:forAspect:** with the "value" aspect. Another EOAssociation might be tied to a Grant Raise button that sends a **raiseSalary** message to the selected enterprise objects when the button is clicked.

EOAssociations that work with multiple EODisplayGroups and enterprise objects might need to interact more directly with the EODisplayGroups. They can retrieve the EODisplayGroup for a particular aspect using **displayGroupForAspect:**, and the key it's bound by with **displayGroupKeyForAspect:**. Once it has these, the EOAssociation can send the EODisplayGroup **setSelectedObjectValue:forKey:**, **setValue:forObject:key:**, or **setValue:forObjectAtIndex:key:** messages as needed to effect its changes.

For display objects that must be edited to change their values, such as text fields, the EOAssociation must respond to messages indicating that the display object has begun and ended editing, and inform the EODisplayGroup for its primary or value aspect with **associationDidBeginEditing:** and **associationDidEndEditing:** messages. EODisplayGroups may need to end editing themselves, such as when saving changes. Your EOAssociation subclass should implement **endEditing** to handle this case by sending whatever value it currently has back to the EOAssociation using a **setValue:forAspect:** message. If it can't do this, the EOAssociation should return NO to disallow whatever operation invoked **endEditing**.

EOAssociations for display objects that present multiple values must also monitor the selection in the display object, updating the EODisplayGroup with a **setSelectionIndexes:** message any time the selection changes in the display object.

Validation

Although validation of values entered by the user can happen in several places, EOAssociations generally concern themselves only with data entry errors. These errors are typically caught by the display object or an NSFormatter, and result in a message to the delegate of the display object. For example, an NSControl

sends **control:isValidObject:** and **control:didFailToFormatString:errorDescription:** to its delegate, allowing the delegate to validate values itself or to handle errors caught by an NSFormatter. Your implementation of a method such as **control:isValidObject:** should simply try to save the new value, using EOAssociation's **setValue:forAspect:** or **setValue:forAspect:atIndex:**, returning YES or NO as that message does. For **control:didFailToFormatString:errorDescription:**, the typical response should be to invoke **shouldEndEditingForAspect:invalidInput:errorDescription:** or **shouldEndEditingForAspect:invalidInput:errorDescription:index:**.

Adopted Protocols

NSCoding	– encodeWithCoder: – initWithCoder:

Method Types

Declaring capabilities	+ aspects + aspectSignatures + objectKeysTaken + isUsableWithObject: + associationClassesSuperseded + displayName + primaryAspect – canBindAspect:displayGroup:key:
Getting all possible EOAssociations for a display object	+ associationClassesForObject:
Creating and configuring instances	– initWithObject: – bindAspect:displayGroup:key: – establishConnection – breakConnection – copyMatchingBindingsFromAssociation:
Getting the display object	– object
Examining bindings	– displayGroupForAspect: – displayGroupKeyForAspect:
Updating values	– subjectChanged – endEditing

Accessing enterprise object values	– setValue:forAspect: – setValue:forAspect:atIndex: – valueForAspect: – valueForAspect:atIndex:
Handling validation errors	– shouldEndEditingForAspect:invalidInput:errorDescription: – shouldEndEditingForAspect:invalidInput:errorDescription:index:

Class Methods

aspects

+ (NSArray *)**aspects**

Overridden by subclasses to return the names of the receiving class's aspects, as NSStrings. Subclasses should include their superclass's aspects and add their own when overriding this method.

See also: + **aspectSignatures**

aspectSignatures

+ (NSArray *)**aspectSignatures**

Overridden by subclasses to return the signatures of the receiver's aspects, an array of string objects matching its aspects array index for index. Each signature string can contain the following characters:

Signature Character	Meaning
A	The aspect can be bound to attributes.
1 (one)	The aspect can be bound to to-one relationships.
M	The aspect can be bound to to-many relationships.

An aspect signature string of "A1", for example, means the corresponding aspect can be bound to either attributes or to-one relationships. An empty signature indicates that the corresponding aspect can be bound to an EODisplayGroup without a key (that is, the key is irrelevant). Interface Builder uses aspect signatures to enable and disable keys in its Connections inspectors.

EOAssociation's implementation of this method returns an array of "A1M" of the length of its aspects array.

See also: + **aspects**

associationClassesForObject:

+ (NSArray *)**associationClassesForObject:**(id)*aDisplayObject*

Returns the subclasses of EOAssociation usable with *aDisplayObject*. Sends **isUsableWithObject:** to every loaded subclass of EOAssociation, adding those that respond YES to the array. Subclasses shouldn't override this method; override **isUsableWithObject:** instead.

associationClassesSuperseded

+ (NSArray *)**associationClassesSuperseded**

Overridden by subclasses to return the other EOAssociation classes that the receiver supplants. This allows a subclass to mask its superclasses from the Connection Inspector's pop-up list in Interface Builder, since the subclass always includes the aspects and functionality of its superclasses. For example, EOPopUpAssociation supersedes EOControlAssociation, because for pop-up buttons an EOPopUpAssociation is always more appropriate to use.

displayName

+ (NSString *)**displayName**

Returns the name used by Interface Builder in the Connection Inspector's pop-up list. EOAssociation's implementation simply returns the name of the receiving class.

isUsableWithObject:

+ (BOOL)**isUsableWithObject:**(id)*aDisplayObject*

Overridden by subclasses to return YES if instances of the receiving class are usable with *aDisplayObject*, NO if they aren't. The receiving class can examine any relevant characteristic of *aDisplayObject*—its class, configuration (such as whether an NSMatrix operates in radio mode), and so on.

objectKeysTaken

+ (NSArray *)**objectKeysTaken**

Overridden by subclasses to return the names of display object outlets that instances assume control of, such as "target" and "delegate". Interface Builder uses this information to disable connections from these outlets in its Connections Inspector.

primaryAspect

+ (NSString *)**primaryAspect**

Overridden by subclasses to return the default aspect, usually one denoting the displayed value, which by convention is named “value”. EOAssociation’s implementation returns **nil**.

Instance Methods

bindAspect:displayGroup:key:

– (void)**bindAspect:**(NSString *)*aspectName*
displayGroup:(EODisplayGroup *)*aDisplayGroup*
key:(NSString *)*key*

Defines the receiver’s link between its display object and *aDisplayGroup*. *aspectName* is the name of the aspect it observer in its display object, and *key* is the name of the property it observes in *aDisplayGroup*. Invoke **establishConnection** after this method to finish setting up the binding. See “Setting up an EOAssociation Programmatically” in the class description for more information.

See also: – **initWithObject:**, – **establishConnection**

breakConnection

– (void)**breakConnection**

Removes the receiver from its EODisplayGroup and display object. This causes it to be released, so be sure to retain the EOAssociation before invoking this method if you want to keep it for another use. Subclasses should override this method to remove the receiver from any outlets of the display object, such as target or delegate, and invoke **super**’s implementation at the end.

See also: – **establishConnection**

canBindAspect:displayGroup:key:

– (BOOL)**canBindAspect:**(NSString *)*aspectName*
displayGroup:(EODisplayGroup *)*aDisplayGroup*
key:(NSString *)*key*

Overridden by subclasses to return YES if the receiver can tie an aspect named *aspectName* from its display object to the property identified by *key* in *aDisplayGroup*, NO if it can’t. *aspectName* should name an aspect supported by the receiver’s class.

Interface Builder uses this information to disable aspects in its Connections Inspector. Subclasses can override this method to base their answers on other binds already made, or on characteristics of the receiver's display object or of *aDisplayGroup*. EOAssociation's implementation always returns YES.

See also: + **aspects**, – **localKeys** (EODisplayGroup), – **attributeKeys** (EOClassDescription), – **toOneRelationshipKeys** (EOClassDescription), – **toManyRelationshipKeys** (EOClassDescription)

copyMatchingBindingsFromAssociation:

– (void)**copyMatchingBindingsFromAssociation:**(EOAssociation *)*anAssociation*

Duplicates the bindings of *anAssociation* in the receiver. For each aspect of *anAssociation* that has an EODisplayGroup, invokes **bindAspect:displayGroup:key:** with the EODisplayGroup and key for that aspect.

See also: – **displayGroupForAspect:**, – **displayGroupKeyForAspect:**

displayGroupForAspect:

– (EODisplayGroup *)**displayGroupForAspect:**(NSString *)*aspectName*

Returns the EODisplayGroup bound to the receiver for *aspectName*, or **nil** if there's no such object.

See also: – **displayGroupKeyForAspect:**

displayGroupKeyForAspect:

– (NSString *)**displayGroupKeyForAspect:**(NSString *)*aspectName*

Returns the EODisplayGroup key bound to the receiver for *aspectName*, or **nil** if there's no EODisplayGroup.

See also: – **displayGroupForAspect:**

endEditing

– (BOOL)**endEditing**

Overridden by subclasses to pass the value of the receiver's display object to the EODisplayGroup, by invoking **setValue:forAspect:** with the display object's value and the appropriate aspect (typically "value"). Returns YES if successful, NO if not—specifically if **setValue:forAspect:** returns NO. The receiver should also send an **associationDidEndEditing:** message to its EODisplayGroup.

Subclasses whose display objects immediately pass their changes back to the EOAssociation—such as a button or pop-up list—need not override this method. It's only needed when the display object's value is edited rather than simply set.

EOAssociation's implementation does nothing but return YES.

establishConnection

– (void)**establishConnection**

Overridden by subclasses to attach the receiver to the outlets of its display object, and to otherwise configure the display object (such as by setting its action method). EOAssociation's implementation subscribes the receiver as an observer of its EODisplayGroups and causes the display object to retain the receiver. Subclasses should invoke **super**'s implementation after establishing their own connections.

See "Setting up an EOAssociation Programmatically" in the class description for more information.

See also: – **breakConnection**

initWithObject:

– (id)**initWithObject:**(id)*aDisplayObject*

Initializes the receiver to monitor and update the value in *aDisplayObject*, which is typically a user-interface object or an EODisplayGroup. This is the designated initializer for the EOAssociation class. Returns **self**.

Note: Because of the way that EOAssociations are set up, this method doesn't retain *aDisplayObject*. See "Setting up an EOAssociation Programmatically" in the class description for more information.

See also: – **bindAspect:displayGroup:key:**, – **establishConnection**

object

– (id)**object**

Returns the receiver's display object.

See also: – **initWithObject:**

setValue:forAspect:

– (BOOL)**setValue:**(id)*value* **forAspect:**(NSString *)*aspectName*

Sets a value of the selected enterprise object in the EODisplayGroup bound to *aspectName*. Retrieves the EODisplayGroup and key bound to *aspectName*, and sends the EODisplayGroup a **setSelectedObjectValue:forKey:** message with *value* and the key as arguments. Returns YES if successful, or if there's no EODisplayGroup bound to *aspectName*. Returns NO if there's an EODisplayGroup and it doesn't accept the new value.

See also: – **setValue:forAspect:atIndex:**, – **valueForAspect:**

setValue:forAspect:atIndex:

– (BOOL)**setValue:**(id)*value*
forAspect:(NSString *)*aspectName*
atIndex:(unsigned int)*index*

Sets a value of the enterprise object at *index* in the EODisplayGroup bound to *aspectName*. Retrieves the EODisplayGroup and key bound to *aspectName*, and sends the EODisplayGroup a **setValue:forObjectAtIndex:key:** message with *value*, *index*, and the key as arguments. Returns YES if successful, or if there's no EODisplayGroup bound to *aspectName*. Returns NO if there's an EODisplayGroup and it doesn't accept the new value.

See also: – **setValue:forAspect:**, – **valueForAspect:atIndex:**

shouldEndEditingForAspect:invalidInput:errorDescription:

– (BOOL)**shouldEndEditingForAspect:**(NSString *)*aspectName*
invalidInput:(NSString *)*inputString*
errorDescription:(NSString *)*errorDescription*

Invoked by subclasses when the display object fails to validate its input, this method informs the EODisplayGroup bound to *aspectName* with an **association:failedToValidateValue:forKey:object:errorDescription:** message, using the EODisplayGroup's selected object. Returns the result of that message, or YES if there's no EODisplayGroup.

For example, an EOAssociation tied to an NSControl object receives a **control:didFailToFormatString:errorDescription:** delegate message when the control's formatter fails to format the input string. Its implementation of that method invokes **shouldEndEditingForAspect:invalidInput:errorDescription:**.

See also: – **shouldEndEditingForAspect:invalidInput:errorDescription:index:**

shouldEndEditingForAspect:invalidInput:errorDescription:index:

– (BOOL)**shouldEndEditingForAspect:**(NSString *)*aspectName*
invalidInput:(NSString *)*inputString*
errorDescription:(NSString *)*errorDescription*
index:(unsigned int)*index*

Works in the same manner as **shouldEndEditingForAspect:invalidInput:errorDescription:**, but allows you to specify a particular object by *index* rather than implicitly specifying the selected object.

subjectChanged

– (void)**subjectChanged**

Overridden by subclasses to update its state based on its EODisplayGroups, whose selection or contents may have changed. This method is invoked automatically anytime an EODisplayGroup bound to the receiver changes. The receiver can query its EODisplayGroup with **selectionChanged** and **contentsChanged** messages to determine how it needs to update.

valueForAspect:

– (id)**valueForAspect:**(NSString *)*aspectName*

Returns a value of the selected enterprise object in the EODisplayGroup bound to *aspectName*. Retrieves the EODisplayGroup and key bound to *aspectName*, and sends the EODisplayGroup a **selectedObjectValueForKey:** message with the key. Returns **nil** if there's no EODisplayGroup or key bound to *aspectName*.

See also: – **valueForAspect:atIndex:**, – **setValue:forAspect:**

valueForAspect:atIndex:

– (id)**valueForAspect:**(NSString *)*aspectName* **atIndex:**(unsigned int)*index*

Returns a value of the enterprise object at *index* in the EODisplayGroup bound to *aspectName*. Retrieves the EODisplayGroup and key bound to *aspectName*, and sends the EODisplayGroup a **valueForObjectAtIndex:key:** message with *index* and the key. Returns **nil** if there's no EODisplayGroup or key bound to *aspectName*.

See also: – **valueForAspect:**, – **setValue:forAspect:atIndex:**