The EOInterface Framework

Framework:	System/Library/Frameworks/EOInterface.framework
Header File Directories:	System/Library/Frameworks/EOInterface.framework/Headers

Introduction

The EOInterface framework defines one of the layers of the Enterprise Objects Framework architecture—the interface layer.

The relationship between user interface objects and enterprise objects is managed by an instance of the EODisplayGroup class. EODisplayGroups are used by EOAssociation objects to mediate between enterprise objects and the user interface. EOAssociations link a single user interface object to one ore more class properties (keys) of the objects managed by an EODisplayGroup. The properties' values are displayed in the association's user interface object.

In the Interface layer, EOAssociation objects "observe" EODisplayGroups to make sure that the data displayed in the user interface remains consistent with enterprise object data. EODisplayGroups interact with a data source, which supplies them with enterprise objects.

The interface layer's associations are listed in the following table:

Association	Yellow Box	Java Client	Description
EOActionAssociation	Yes	Yes	Allows you to set up an interface object, such as a button, to send a message to the objects selected in the association's display group when the interface object is acted on
EOActionCellAssociation	Yes	No	The default association class for use with NSActionCells
EOActionInsertionAssociation	Yes	Yes	Inserts objects from one display group into another.
EOAssociation	Yes	Yes	Defines the mechanism that transfers values between EODisplayGroups and the user interface of an application.
EOColumnAssociation	Yes	No	Cooperates with an EOTableViewAssociation to display values in a column of an NSTableView
EOComboBoxAssociation	Yes	Yes	Displays an attribute or to-one relationship value in a combo box

Association	Yellow Box	Java Client	Description
EOControlAssociation	Yes	No	The default EOAssociation subclass for use with NSControl objects
EODetailSelectionAssociation	Yes	No	Binds two EODisplayGroups together through a relationship, so that the destination display group acts as an editor for that relationship.
EOGenericControlAssociation	Yes	No	the abstract superclass of EOControlAssociation and EOActionCellAssociation.
EOMasterCopyAssociation	Yes	No	Synchronizes two EODisplayGroups that share the same data source but have different qualifiers.
EOMasterDetailAssociation	Yes	Yes	Binds one EODisplayGroup (the detail) to a relationship in another (the master), so that the detail display group contains the destination objects for the object selected in the master.
EOMasterPeerAssociation	Yes	No	Binds two EODisplayGroups together in a master-detail relationship, where the detail EODisplayGroup shows the destination objects for the relationship of the master EODisplayGroup.
EOMatrixAssociation	Yes	No	Allows you to populate an NSMatrix's cells.
EOPickTextAssociation	Yes	No	Allows the user to perform a similarity search based on whole or partial values.
EOPopUpAssociation	Yes	No	Displays an attribute or to-one relationship value in an NSPopUpButton
EORadioMatrixAssociation	Yes	No	Displays a string or an integer in an NSMatrix.
EORecursiveBrowserAssociation	Yes	No	The default association for use with a multi-column NSBrowser.
EOTableAssociation	No	Yes	Associates a display group with a Swing JTable.
EOTableColumnAssociation	No	Yes	Associates a single attribute of all enterprise objects in a display group with a Swing JTable TableColumn.
EOTableViewAssociation	Yes	No	Manages the individual EOColumnAssociations between an NSTableView (Application Kit) and an EODisplayGroup.

Association	Yellow Box	Java Client	Description
EOTextAssociation	Yes	Yes	Displays a plain or rich text attribute in an NSText object (Application Kit) or an EOTextField, EOTextArea, or EOFormCell (Java Client) by binding the text object to a string or NSData attribute.

In addition to the above association classes, EOInterface defines the following classes for use exclusively with Java Client applications:

- EOApplet
- EOApplication
- EOArchive
- EOInterfaceController
- EOViewLayout

EOActionAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOActionAssociation.h

Class Description

An EOActionAssociation object allows you to set up an interface object, such as a button, to send a message to the objects selected in the association's display group when the interface object is acted on.

Usable With
Any control object (in the Yellow Box, any NSControl or NSActionCell)

Aspects

action	Bound to a key that names the method to invoke on the selected objects. If the **argument** aspect isn't bound, the method must take no arguments. If the **argument** aspect is bound, then the method must take exactly one argument.
argument	An object attribute or relationship of the selected object, passed as an argument to the action method. (Usually bound to a different EODisplayGroup than the one bound to **action**.)
enabled	A boolean attribute of the selected object, which determines whether the display object is enabled.

Object Keys Taken

target	On receiving an action message from the display object, an EOActionAssocation sends its action to the selected objects.

Examples

Suppose you have an application that manages member accounts, each of which has a restriction on the outstanding balance allowed. You want a user to be able to increase the restriction limit by selecting one or more members and then clicking a button. To do this, you define a **boostRestrictions** method in the Member class that increases the limit by 20%. In Interface Builder, control-drag a connection from the button to the Member display group. Select EOActionAssociation in the Connections inspector, and bind the association's **action** aspect to the "boostRestrictions" key.

In another scenario, one EODisplayGroup shows Members, while another shows video tapes available for rent. Here, you want a user to be able to select a member, select a video tape, and then click a Rent button that checks the selected tape out to the selected member. To do this, define a **rentVideoTape:** method in the Member class that takes a VideoTape as an argument and handles the accounting involved in a video rental. Then, in Interface Builder, control-drag a connection from the button to the Members display group. Select EOActionAssociation in the Connections inspector, and bind the association's **action** aspect to Member's **rentVideoTape:** action. Similarly, control-drag a connection from the button to the VideoTape display group. Select EOActionAssociation in the Connections inspector, and bind the association's **argument** aspect to the VideoTape display group. Now, when the user selects a Member, selects a VideoTape, and clicks the button, the selected Member is sent a **rentVideoTape:** message with the selected VideoTape.

Instance Methods

action:

– (void)**action:**(id)*sender*

Invoked when the receiver's display object is acted upon. Sends the method identified by the receiver's **action** aspect (with an argument, if the **argument** aspect is bound) to the selected objects.

EOActionCellAssociation

Inherits From:	EOGenericControlAssociation : EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOControlAssociation.h

Class Description

EOActionCellAssociation is the default association class for use with NSActionCells (Application Kit). An EOActionCellAssociation object displays the value of the selected object in its NSActionCell, and updates the object when the NSActionCell's value changes. A sibling class, EOControlAssociation, can be used with independent controls such as NSButtons and NSTextFields. Other associations, such as EOPopUpAssociation and EOColumnAssociation, supersede these classes for more specialized behavior.

When multiple EOActionCellAssociations are bound to cells in the same control (such as in an Application Kit NSMatrix), one of them becomes the delegate of the control and forwards appropriate messages, such as **control:isValidObject:**, to the others. This eliminates the need to add an EOControlAssociation just to handle delegate messages.

EOActionCellAssociations access values using NSActionCell's **setObjectValue:** method, which allows values with non-string representations to be displayed. An EOActionCellAssociation can be bound to an NSImageCell, for example, with an attribute whose class is NSImage.

Usable With
Any NSActionCell

Aspects	
value	An attribute of the selected object, displayed in the NSActionCell.
enabled	A boolean attribute of the selected object, which determines whether the NSActionCell is enabled.

Object Keys Taken	
target	On receiving an action message from the NSActionCell, an EOActionCellAssociation sends the NSActionCell's value to the EODisplayGroup.
delegate	See the class description.

Examples

To display a movie's budget in an NSTextFieldCell, in Interface Builder, control-drag a connection from the text field to the Movie display group. Select EOActionCellAssociation in the Connections inspector, and bind the **value** aspect to the "budget" key. Then, if the NSTextFieldCell is editable, when the user types a new value and presses Enter or Tab, the selected movie's **budget** attribute is changed.

Assuming that Movie objects implement an **isBudgetNegotiable** method, you can make the NSTextFieldCell uneditable depending on the selected movie. To do so, bind the **enabled** aspect to the "isBudgetNegotiable" key.

Instance Methods

control

– (NSControl *)**control**

Returns the NSControl that owns the receiver's display object.

See also: – **object** (EOAssociation), **– controlView** (NSActionCell class of the Application Kit)

editingAssociation

– (EOGenericControlAssociation *)**editingAssociation**

For EOActionCellAssociations in an NSMatrix (defined in the Application Kit) or other multi-celled control, returns the selected EOActionCellAssociation (or the one that's editing text).

EOActionInsertionAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOActionInsertionAssociation.h

Class Description

An EOActionInsertionAssociation object inserts objects from one display group into another. In the Yellow Box, the EOActionInsertion object uses NSControl's **action** method as a signal to perform the insertion.

Usable With

Any object that responds to **setAction:**, in the Yellow Box typically an NSControl.

Aspects

source	Bound to the EODisplayGroup containing objects to insert. This aspect doesn't use a key.
destination	A relationship of the selected object into which objects from the source EODisplayGroup are inserted. Usually bound to a different EODisplayGroup than **source**.
enabled	A boolean attribute of the selected object (usually in the destination EODisplayGroup), which determines whether the NSControl is enabled.

Object Keys Taken

target	On receiving an action message from the display object, an EOActionInsertionAssociation inserts objects from the source EODisplayGroup into the destination EODisplayGroup.

Example

Suppose an application shows Talent in one display group and Movies in another. You want a user to be able to select a talent, select a movie, and then click an Assign Director button that assigns the selected talent as one of the movie's directors. To do this, in Interface Builder, control-drag a connection from the button to the Talent display group. Select EOActionInsertionAssociation in the Connections inspector, and double-click the association’s **source** aspect, binding it to the Talent display group. Similarly, control-drag a connection from the button to the Movie display group. Select EOActionAssociation in the Connections inspector, and bind the association’s **destination** aspect to the “directors” key. Now, when the user clicks the button, the selected Talent is added to the **directors** relationship of the selected Movie. If more than one talent is selected, both are added to the relationship. If more than one Movie is selected, the selected talent are added to the relationship of the first Movie in the selection.

EOApplet

Inherits From: com.sun.java.swing.JApplet

Package: com.apple.client.eointerface

Class Description

EOApplet is the default Applet class embedded in WebObjects pages containing a WOJavaClientApplet WOElement. EOApplet is actually something of a shell, as all application initialization logic actually resides in EOApplication. For maximum flexibility, any application specifics should be implemented in EOApplication's **finishInitialization** rather than EOApplet's **init**.

EOApplet is for use in Java Client applications only; there isn’t an equivalent class for Yellow Box.

Instance Methods

init

public void **init**()

Instantiates an EODistributionChannel before passing it to EOApplication's **application** method, using the receiver's **contentPane** as *container* and *className*,*languages* and *controllerClassName* retrieved via Applet's **getParameter**.

EOApplication

Inherits From:	com.sun.java.swing.JApplet
Package:	com.apple.client.eointerface

Class Description

Java programs typically execute either as either an Applet running in a browser or care of a class implementing the static method **main** (often referred to as an "application"). EOApplication insulates the developer from the details of this distinction by serving as an execution-mode-independent repository for application-level client-side logic. The provided JApplet subclass EOApplet is essentially nothing more than an EOApplication-invoking shell.

EOApplication is for use in Java Client applications only; there isn’t an equivalent class for Yellow Box.

Constructors

protected **EOApplication**()

This constructor is exposed simply so that subclasses may refer to it without compiler error. EOApplication should always be instantiated through one of the two static initializers.

Static Methods

application

public static com.apple.client.eointerface.EOApplication
application(com.apple.client.eodistribution.EODistributionChannel *channel*)

One of two **sharedApplication** initializers, this variant should be invoked in contexts such as **main** where the parameters required by the following edition are unavailable and must be requested from a server-side WOJavaClientApplet via *channel*. Once these parameters have been retrieved this method performs the same initialization as the following method.

public static com.apple.client.eointerface.EOApplication **application**(java.lang.String *className*,
com.apple.client.eodistribution.EODistributionChannel *channel*
com.apple.client.foundation.NSArray *languages*,
java.lang.String *controllerClassName*,
java.awt.container *container*)

Sets **sharedApplication** to a new instance of the EOApplication with *className*. After initializing an EODistributedObjectStore with *channel* and establishing it as EOEditingContext's **defaultParentObjectStore**, this method creates an instance of the EOInterfaceController with *controllerClassName* and runs it in *container* (see **createInterfaceController** and **languages** for further details).

Instance Methods

applicationWillExit

protected boolean **applicationWillExit**()

Invoked whenever the last window registered with the receiver via **registerWindow** has been closed in a non-Applet execution context, this method gives custom subclasses an opportunity to perform any necessary cleanup before exit or simply reject EOApplication's window management logic by returning **false** (see **registerWindow**). The default implementation returns **true**.

createInterfaceController

public static com.apple.client.eointerface.EOInterfaceController
createInterfaceController(java.lang.String *className*, java.awt.Container *container*)

This convenience method creates and runs a new instance of the EOInterfaceController with *className*. If *container* is non-null, it is passed to the new controller's implementation of **runInContainer**, otherwise the "presentation-neutral" method **run** is invoked. Returns the new instance.

finishInitialization

protected void **finishInitialization**()

Invoked as the final step in the receiver's bootstrapping, this method represents a subclass initialization hook somewhat analogous to Applet's **init**. By the time **finishInitialization** is invoked, EOEditingContext's **defaultParentObjectStore** has been set, connecting the receiver to the server, and any specified EOInterfaceController has been instantiated and run.

languages

public void com.apple.client.foundation.NSArray **languages**()

Returns the array of languages supported by the receiver's localization as defined by developer and retrieved from the server (note that only the first element of this array, typically defined by WOJavaClientApplet's WOAppletLanguageKey, is currently used).

main

public static void **main**(java.lang.String *arguments*)

EOApplication's implementation of this method allows client-side programs to execute from the command line in addition to running as an Applet. The first, required element of *arguments* must be the same application URL users would enter to access the equivalent EOApplet in a browser. The optional second element should be the name of any initial entry page other than Main. After instantiating an EODistributionChannel on the basis of these two parameters,main simply invokes the channel-driven version of **application**.

registerWindow

public void **registerWindow**(java.awt.Window *window*)

Disposing of all windows in a non-Applet-based user interface launched via **main** does not cause the program to exit because the Java virtual machine continues to run. EOApplication attempts to manage this problem for the consumer by maintaining a registry of all Windows. When the last remaining Window registered with this method is disposed of in a non-Applet execution context, EOApplication invokes **shouldTerminate**. If this method returns **true**, System's **exit** method is invoked with a zero *status*. Also see **windowRegistryEnabled** and EOInterfaceController's **shouldRegisterWindow**.

Note the use of "disposing" rather than "closing" in the previous paragraph. Despite its name, the AWT WindowListener message **windowClosed** is only sent by java.awt.Window within **dispose**, not **setVisible**. As a result, the exit logic EOApplication provides is only activated by Window disposal.

setLanguages

public void **setLanguages**(com.apple.client.foundation.NSArray *languages*)

Sets the array of languages supported by the receiver's localization as defined by the developer. Note that only the first element of this array is currently referenced.

setWindowRegistryEnabled

public boolean **setWindowRegistryEnabled**(boolean *enabled*)

Sets whether or not the window registration logic described in **registerWindow** should be used by the receiver.

shouldTerminate

protected boolean **shouldTerminate**()

This method acts as a subclass hook for application cleanup logic and refinement of the exit condition described in **registerWindow**. A return value of **false** will inhibit invocation of **exit** when all registered windows have been disposed. The default implementation returns **true**.

sharedApplication

public static void **sharedApplication**()

Returns the EOApplication instance initialized via one the two **application** methods, throwing an IllegalStateException if neither has been invoked.

windowRegistryEnabled

public boolean **windowRegistryEnabled**()

Returns whether or not window registration logic described in **registerWindow** is used by the receiver. The default is **true**, but this may be overridden via **setWindowRegistryEnabled**.

EOArchive

Inherits From:	java.lang.Object
Package:	com.apple.client.eointerface

Class Description

An EOArchive is a client-side rendering of an Interface Builder archive document. By default, the EOInterfaceController for a WOJavaClientApplet manages its associated archive transparently and the developer never needs to interact directly with this class. The **loadArchiveNamed** method is exposed for developers who do not wish to use EOInterfaceControllers. Please note that EOArchive's implementation is intentionally opaque because it is guaranteed to change in the next release.

Constructors

public **EOArchive**(Object *owner*)

Creates an EOArchive instance initialized with *owner*, the "File's Owner" for the Interface Builder archive document.

Static Methods

loadArchiveNamed

public static boolean **loadArchiveNamed**(String *archiveName*, Object *owner*, String *packageName*)

Loads a new instance of the EOArchive *name*, where *name* identifies an InterfaceBuilder document (localization is driven by WOJavaClientApplet's EOLanguageKey) and *owner* indicates the "File's Owner.""The *packageName* argument identifies a package and is used in constructing the complete path to the archive. If *packageName* is **null**, the package of *owner* is used.

EOAssociation

Inherits From:	EODelayedObserver (EOControl) : 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, such as a text field or combo box, and the properties of one or more enterprise objects contained in one or more EODisplayGroups. You typically create and configure associations in Interface Builder, using the programmatic interface only when you write your own EOAssociation subclasses. For information on the different kinds of associations you can use, see the following subclass specifications:

EOActionAssociation	EOActionCellAssociation
EOActionInsertionAssociation	EOColumnAssociation
EOComboBoxAssociation	EOControlAssociation
EODetailSelectionAssociation	EOGenericControlAssociation
EOMasterCopyAssociation	EOMasterDetailAssociation
EOMasterPeerAssociation	EOMatrixAssociation
EOPickTextAssociation	EOPopUpAssociation
EORadioMatrixAssociation	EORecursiveBrowserAssociation
EOTableViewAssociation	EOTextAssociation

Principal Attributes

- A display object (such as a text field or combo box)

- Aspects that control different parameters of the display object (such as **value** and **enabled**)
- One or more EODisplayGroups (no more than one per aspect)
- One or more keys (enteprise object properties) (as many as one key per aspect)

Creation

Interface Builder	
– initWithObject:	Designated initializer.

Class at a Glance

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 create 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; each of the class specifications for EOAssociation's subclasses provide an example using Interface Builder to set them up. EOAssociation's programmatic interface is more important when defining custom EOAssociation subclasses. For more information on EOAssociations, see the sections:

- How EOAssociations Work
- Setting up an EOAssociation Programmatically
- Creating a Subclass of EOAssociation

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 an array of string objects. Subclasses should include their superclass's aspects and add their own when overriding this method.

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.

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: – **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.

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 display group and key bound to *aspectName*, and sends the display group a **setSelectedObjectValue: forKey:** message with *value* and the key as arguments. Returns YES if successful, or if there's no display group bound to *aspectName*. Returns NO if there's an display group and it doesn't accept the new value.

See also: – **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 display group and key bound to *aspectName*, and sends the display group a **setValue:forObjectAtIndex: key:** message with *value*, *index*, and the key as arguments. Returns YES if successful, or if there's no display group bound to *aspectName*. Returns NO if there's a display group and it doesn't accept the new value.

See also: – **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 display group's selected object. Returns the result of that message, or YES if there's no display group.

For example, an association bound to an NSControl object (Application Kit) 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 state based when an EODisplayGroup’s selection or contents changes. This method is invoked automatically anytime a display group that’s bound to the receiver changes. The receiver can query its display group 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 display group and key bound to *aspectName*, and sends the display group a **selectedObjectValueForKey:** message with the key. Returns **nil** if there’s no display group or key bound to *aspectName*.

See also: – **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 display group and key bound to *aspectName*, and sends the display group a **valueForObjectAtIndex: key:** message with *index* and the key. Returns **nil** if there’s no display group or key bound to *aspectName*.

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

EOAssociation

How EOAssociations Work

An EOAssociation monitors its display object for user input or other events while also observing changes in the selection or contents of its EODisplayGroups. The basic purpose of an EOAssociation is to assure that changes at one end are reflected on the other. When the selection in a display group changes, for example, the association updates the state of its display object to reflect this new selection. The following sections describe this process in detail.

The Display Object

In the Yellow Box, an EOAssociation is tied to a single display object. Each EOAssociation assumes the roles defined for one or more outlets of this object. An EOControlAssociation, for example, appropriates the target and action outlets of the NSControl it is bound to. When the user activates the control or changes its value, the action is fired and the EOAssociation correspondingly updates a property of the display group's selected enterprise object. An EOControlAssociation also sets itself as the control's delegate in order to receive various editing and validation messages.

In the Yellow Box, any outlets an association claims cannot be used for other purposes. The class method **objectKeysTaken** returns the names of any outlets a given EOAssociation subclass appropriates, and InterfaceBuilder disables them in its Connections Inspector if the inspected object has been associated. A button acting as an EOControlAssociation's display object, for example, has its target outlet dimmed.

Although display objects are typically user-interface controls such as text fields and pop-up menus, they can be any kind of object. A notable example of this is an EOMasterDetailAssociation, where the display object is a "detail" EODisplayGroup populated with the destination enterprise objects of a relationship in the "master" display group. See the EOMasterDetailAssociation class specification for more information on master-detail configurations.

Bindings: Aspects, EODisplayGroups, and Keys

Although an EOAssociation has only one display object it may have any number of aspects. Aspects define the EODisplayGroup characteristics that the association observes. Aspects are bound to a display group by a key of the enterprise objects contained by the association. Depending upon a given EOAssociation subclass, aspects may be optional or mandatory. They might all have to be bound to a single EODisplayGroup or they may span several. Some aspects can be mutually exclusive.

On the display side, aspects are typically bound to visible facets of the EOAssociation's display object, such as the value or values it displays and any interactive state. Each aspect's value is determined by the contents of the enterprise-object property in the EODisplayGroup that the aspect is bound to. This value may be taken from all enterprise objects in the EODisplayGroup or only those in the current selection. Some aspects are "read-only" in that they merely reflect the contents of the display group, but others change enterprise-object values when the display object is manipulated.

An EOControlAssociation, for example, defines “value” and “enabled” aspects. To configure a text field to display the salary for the selected enterprise object you must create an EOControlAssociation with the text field as its display object and bind the EOControlAssociation’s “value” aspect to the appropriate display group’s “salary” key. You might also bind the EOControlAssociation’s “enabled” aspect to some key such as “eligibleForRaise” so that the text field is made editable if this property evaluates to non-zero. When focus leaves the text field, the newly entered value is sent to the EODisplayGroup.

A multi-valued aspect can represent the destination of a to-many relationship or it can define a range of possible values for an enterprise object’s property. EOComboBoxAssociation, for example, has a “titles” aspect that defines all possible values for a key, and all these values then appear in the pop-up menu. If, for example, you bind the “titles” aspect to the “name” key of an EODisplayGroup containing Departments, you get a pop-up menu containing the names of all departments. EOComboBoxAssociation also has a “selectedObject” aspect which, when bound to a relationship property of an enterprise object, determines the selection in the “titles” display group.

As EODelayedObservers, EOAssociations add themselves to the list of objects observing the display groups they are bound to. When a display group changes its selection or contents, observing EOAssociations are sent a **subjectChanged** message. This message does not indicate which EODisplayGroup has changed, so the receiver must query each one. When an EOAssociation wishes to modify the contents of a EODisplayGroup, it typically does so through the **setValue:forAspect:**. This process and the querying of display groups are described under ““Monitoring Changes from the Display Object”.”

Setting up an EOAssociation Programmatically

Although you normally use the Interface Builder application (and the EOPalette palette) to set up EOAssociations, you can do so programmatically as well. Because EOAssociation coordinates the actions of many objects, linking a display object to a display group is a multi-step process, as shown by the following code fragment; this fragment assumes that `salaryText` and `employeeGroup` already exist.

```
NSTextField *salaryText;
EODisplayGroup *employeeGroup;
EOControlAssociation *association;

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

Although an association is initialized with the display object it monitors, this really represents only half of the required initialization; the association and therefore the display object have yet to be bound to any display group. The two invocations of **bindAspect:displayGroup:key:** define the specifics of the field’s interaction with `employeeGroup`. Once these aspects have been bound, **establishConnection** causes the association to register as an observer of `employeeGroup` and complete its internal initialization. Note that in the Yellow Box you can safely release a newly instantiated association once you invoke **establishConnection** because this method retains the association for the lifespan of the display object.

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 it can work with.
- How your subclass establishes its connections with its display object and its EODisplayGroups
- How it updates the display object to reflect display group changes.
- How it monitors the display object and updates the EODisplayGroups.

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

Defining Capabilities

If you're creating a Yellow Box subclass, a significant part of creating an EOAssociation subclass is defining and advertising what the subclass works with. The characteristics that your subclass should define are:

Characteristic	Optional/ Required	Description
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")

Characteristic	Optional/ Required	Description
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 defined for the class. 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 the array (“A”, “A”). EOMasterDetailAssociation only works with relationships, so the aspect signature for its aspect “parent” is the array (“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.

Characteristic	Optional/ Required	Description
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 subclass's implementation of this method should first invoke the superclass implementation to initialize the observation of bound EODisplayGroups and then establish their notification relationship with the display object. Once the association has been bound to its display groups and appropriately attached to its display object it is ready to perform real work.

Monitoring Changes from the EODisplayGroup

An EOAssociation is notified of changes in EODisplayGroup selections and changes through EODelayedObserver's **subjectChanged** method. An EOAssociation sublcass, in its implementation of this method, propagates these changes to the display object. Because **subjectChanged** provides no additional information about the change that triggered its invocation, associations must query their bound display groups for details. The EOAssociation method **displayGroupForAspect:**, in conjunction with EODisplayGroup's **contentsChanged** and **selectionChanged**, faciliate efficient aspect-by-aspect change analysis. Once you have determined the set of affected aspects, your subclass must update its display object to reflect their new values. How this is done is specific to the class of display object and to the aspects your EOAssocation subclass supports.

Monitoring Changes from the Display Object

When an EOAssociation is notified of a change to the state of its display object, it must update the affected display groups so that they reflect the new state. Updating can involve changing a display-group value, sending messages to the display group, or sending messages to some set of the enterprise objects the display group contains. As a simple example, an association with a “value” aspect would update the value of the bound display group’s selected enterprise object by invoking **setValue:forAspect:** with the display object's new contents. Complex associations might set enterprise object values more directly via EODisplayGroup's **setSelectedObjectValue:forKey:** , **setValue:forObject:key:**, or **setValue:forObjectAtIndex:key:** in conjunction with EOAssocation”s **displayGroupKeyForAspect:**. An association with a button as its display object might go even further, sending the message defined by its “action” aspect to the enterprise objects selected in a display group whenever the button is clicked.

For display objects that support editing, such as text fields, an association must observe events signifying the beginning or end of an editing operation and then inform the appropriate display groups using EODisplayGroup's **associationDidBeginEditing:** and **associationDidEndEditing:**. This operation is important because a display group requests an end to editing when it is asked to perform tasks such as the insertion of a new enterprise object or a save. It requests and end to editing by sending an **endEditing** message to the association it believes currently has an edit in progress. Implementations of **endEditing** should attempt to propagate the current state of the display object to the receiver’s display groups and return NO if this attempt fails, indicating that the request has been disallowed. EOAssociations that support the display of multiple values and the notion of a selection must also propagate changes in this selection to the appropriate display groups using EODisplayGroup’s **setSelectionIndexes:**.

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:**.

EOColumnAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOColumnAssociation.h

Class Description

An EOColumnAssociation object cooperates with an EOTableViewAssociation to display values in a column of an NSTableView (Application Kit). A column association links an NSTableColumn (Application Kit) to a single attribute of all displayed objects in an EODisplayGroup. The value of each object is displayed in its corresponding row.

The EOTableViewAssociation receives target, delegate, and data source messages from the table view, and forwards them as needed to the appropriate column association. Column associations provide values for the cells of each NSTableColumn, and also accept edited values to set in their display groups.

EOColumnAssociations provide values using NSTableView's DataSource methods **tableView: setObjectValue:forTableColumn:row:** and **tableView:objectValueForTableColumn:**. This allows values with non-string representations to be displayed. For example, if an NSImageCell (Application Kit) is used as an NSTableColumn's data cell, an EOColumnAssociation can be used to display NSImages (Application Kit) in the NSTableView.

Usable With

NSTableColumn (Application Kit)

Aspects

value	An attribute of the objects, displayed in each row of the NSTableColumn.
enabled	A boolean attribute of the objects, which determines whether each object's value cell is editable. Note that because EOTableViewAssociation also uses this aspect, you can use it with different keys to limit editability to the whole row or to an individual cell (column) in that row.

Object Keys Taken	
identifier	An EOColumnAssociations sets itself as the identifer of its NSTableColumn. (Note: This key isn't formally reserved by the **objectKeysTaken** method, as Interface Builder doesn't treat it as an outlet.)

Example

To display the last and first names of objects in a Talent display group, in Interface Builder, Control-drag a connection from the last name column to the display group. Select EOColumnAssociation in the Connections inspector, and bind the **value** aspect to the "lastName" key (this automatically creates an EOTableViewAssociation to manage the individual columns). Repeat to set up a column association for the first name. Now when you run the application, the last and first names of each Talent object in the display group's **displayedObjects** array are put in the correponding row.

Method Types

Sorting rows	
	– setSortingSelector:
	– sortingSelector
Table view data source methods	
	– tableView:setObjectValue:forTableColumn:row:
	– tableViewObjectValueForLocationtableView: objectValueForTableColumn:row:
Table view delegate methods	
	– tableView:shouldEditTableColumn:row:
	– tableView:willDisplayCell:forTableColumn:row:
Control delegate methods	
	– controlDidFailToFormatStringErrorDescriptioncontrol: didFailToFormatString:errorDescription:
	– control:isValidObject:
	– control:textShouldBeginEditing:

Instance Methods

setSortingSelector:

– (void)**setSortingSelector:**(SEL)*aSelector*

Sets the method selector used to sort rows to *aSelector*, one of:

- EOCompareAscending
- EOCompareDescending
- EOCompareCaseInsensitiveAscending
- EOCompareCaseInsensitiveDescending
- **nil** (to tell the receiver not to sort)

For more information on these selectors, see the section “Comparison Methods” in the EOSortOrdering class specification (EOControl). Additionally, for details on the sorting methods themselves, see the **compare...** methods in the Value Class Additions specification.

If the EOTableViewAssociation for the receiver’s NSTableView (Application Kit) sorts its rows, it applies this method as needed to sort them. The default sorting selector is EOCompareAscending.

sortingSelector

– (SEL)**sortingSelector**

Returns the method selector used to sort rows, or **nil** if the column isn’t sorted.

Data Source and Delegate Methods

These methods are forwarded by the corresponding NSTableView’s EOTableViewAssociation to the appropriate EOColumnAssociation.

control:didFailToFormatString:errorDescription:

– (BOOL)**control:**(NSControl *)*aTableView*
  didFailToFormatString:(NSString *)*aString*
  errorDescription:(NSString *)*errorDescription*

Invokes **shouldEndEditingForAspect:invalidInput:errorDescription:** (defined by EOAssociation) and returns the result.

control:isValidObject:

– (BOOL)**control:**(NSControl *)*aTableView*
  isValidObject:(id)*anObject*

Saves the value of any cell being edited using **setValue:forAspect:**, and if successful sends an **associationDidEndEditing:** message to the receiver’s EODisplayGroup. Returns YES if successful (or if no changes need be saved), NO if unsuccessful.

control:textShouldBeginEditing:

– (BOOL)**control:**(NSControl *)*aTableView*
  textShouldBeginEditing:(NSText *)*fieldEditor*

Sends an **associationDidBeginEditing:** message to the receiver's EODisplayGroup and returns YES.

tableView:objectValueForTableColumn:row:

– (id)**tableView:**(NSTableView *)*aTableView*
  objectValueForTableColumn:(NSTableColumn *)*aTableColumn*
  row:(int)*rowIndex*

Returns the value of the property of the object at *rowIndex* bound to the **value** aspect.

tableView:setObjectValue:forTableColumn:row:

– (void)**tableView:**(NSTableView *)*aTableView*
  setObjectValue:(id)*value*
  forTableColumn:(NSTableColumn *)*aTableColumn*
  row:(int)*rowIndex*

Sets the property of the object at *rowIndex* bound to the **value** aspect to *value*.

tableView:shouldEditTableColumn:row:

– (BOOL)**tableView:**(NSTableView *)*aTableView*
  shouldEditTableColumn:(NSTableColumn *)*aTableColumn*
  row:(int)*rowIndex*

Returns NO if the **enabled** aspect is bound and its value for the object at *rowIndex* is NO. Otherwise returns YES. Note that because the **enabled** aspects of EOTableViewAssociation and EOColumnAssociation can be bound to different keys, you can limit editability to the whole row or to an individual cell (column) in that row.

tableView:willDisplayCell:forTableColumn:row:

– (void)**tableView:**(NSTableView *)*aTableView*
  willDisplayCell:(id)*aCell*
  forTableColumn:(NSTableColumn *)*aTableColumn*
  row:(int)*rowIndex*

Alters the display characteristics for *aCell* according to the values for the **enabled** aspect of the object at *rowIndex*.

EOComboBoxAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOComboBoxAssociation.h

Class Description

An EOComboBoxAssociation object displays an attribute or to-one relationship value in an NSComboBox (Application Kit), or JComboBox (Swing). The items in the ComboBox can be entered manually, or for a relationship, constructed dynamically from values supplied by an EODisplayGroup. EOComboBoxAssociation is very similar to the EOPopUpAssociation (Yellow Box only).

Usable With
NSComboBox (Application Kit), JComboBox (Swing)

Aspects
(defined in EOAssociation for Java Client applications)

titles (Yellow Box) TitlesAspect (Java Client)	Property of the enterprise objects in an EODisplayGroup that supplies the titles for the items in the combo box list.
selectedTitle (Yellow Box) SelectedTitlesAspect (Java Client)	String property of the enterprise object supplying the title to display in the c ombo box. When the value of the combo box changes either because a new value is typed in or a selection is made using the pop up menu, the new text value is assigned to this property.
selectedObject (Yellow Box) SelectedObjectAspect (Java Client)	Relationship property of the enterprise object containing the enterprise object to select from the **titles** EODisplayGroup. **selectedObject** is usually mutually exclusive with **selectedTitle**. When the value of the combo box changes, the association updates the relationship to point to the new object.
enabled (Yellow Box) EnabledAspect (Java Client)	A boolean attribute of the selected object that determines whether the combo box is enabled.

Object Keys Taken
(none forJava Client)

target	When the user chooses an item in the pop-up menu, the EOComboBoxAssociation updates the selected object's property with the item's title or object.
dataSource	When the NSComboBox requests values for its list, the EOComboBoxAssociation provides them by querying the appropriate EODisplayGroup or groups.
delegate	An EOComboBoxAssociation accepts the message **comboBoxSelectionDidChange:**.

Examples

There are three basic ways to configure a combo box and it's association. Each is described below.

Selecting a String from a Static List

Suppose you have a Movie display group and you want to provide a combo box for setting the rating from a static list of strings. In this example, a Movie object's rating is a string property rather than a relationship to a Rating object). To do this, in Interface Builder, type the list of ratings into the combo box. Control-drag a connection from the combo box to the Movie display group. Choose EOComboBoxAssociation in the Connections inspector, and bind the **selectedTitle** aspect to the "rating" key.

Selecting a String from a Dynamic List

This example is similar to the previous one, except in this example, a Movie object's rating is chosen from strings in a Rating database table. There's a Rating EODisplayGroup that fetches the ratings into Rating objects, and the combo box is filled from the "ratingString" property of the rating display group's Rating objects. To do this, in Interface Builder, control-drag a connection from the combo box to the Ratings display group. Choose EOComboBoxAssociation in the Connections inspector, and bind the **titles** aspect to the "ratingString" key. Similarly, control-drag a connection from the combo box to the Movie display group. Again choose EOComboBoxAssociation in the Connections inspector, and bind the **selectedTitle** aspect to the "rating" key.

Selecting the Destination of a To-One Relationship

Suppose you have a list of employees and want to assign each employee a department. In terms of the object model, you want to assign a Department object as the destination of an Employee object's **department** relationship. To do this, in Interface Builder, control-drag a connection from the combo box to a Department display group. Choose EOComboBoxAssociation in the Connections inspector, and bind the **titles** aspect to the "name" key. Similarly, control-drag a connection from the combo box to the Employee display group. Again choose EOComboBoxAssociation in the Connections inspector, and bind the **selectedObject** to the "department" key.

If the **selectedObject** aspect is bound and the user types a value that doesn’t match any of those currently in the list, an error panel is displayed.

EOControlAssociation

Inherits From:	EOGenericControlAssociation : EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOControlAssociation.h

Class Description

EOControlAssociation is the default EOAssociation subclass for use with NSControl objects (Application Kit). A control association displays the value of the selected object in its control, and updates the object when the control's value changes. A sibling class, EOActionCellAssociation, can be used with individual cells in an NSMatrix or NSForm (both defined in the Application Kit). Some other subclasses of EOAssociation, such as EOPopUpAssociation and EOColumnAssociation, supersede these classes for more specialized behavior.

EOControlAssociations access values using NSControl's **setObjectValue:** method, which allows values with non-string representations to be displayed. An EOControlAssociation can be bound to an NSImageView, for example, with an attribute whose class is NSImage (both NSImageView and NSImage are defined in the Application Kit).

Usable With
Any NSControl (Application Kit)

Aspects

value	An attribute of the selected object, displayed in the NSControl.
enabled	A boolean attribute of the selected object, which determines whether the NSControl is enabled.

Object Keys Taken	
target	On receiving an action message from the NSControl, an EOControlAssociation sends the NSControl's value to the EODisplayGroup.
delegate	An EOControlAssociation accepts messages related to editing and validation of text, such as **control: textShouldBeginEditing:** and **controlDidFailToFormatStringErrorDescriptioncontrol: didFailToFormatString:errorDescription:**.

Examples

To display a movie's budget in an NSTextField, in Interface Builder, control-drag a connection from the text field and a Movie display group. In the Connections inspector, choose EOControlAssociation, and bind the **value** aspect to the "budget" key. Then, if the NSTextField is editable, when the user types a new value and presses Enter or Tab, the selected movie's **budget** attribute is changed.

Assuming that Movie objects implement an **isBudgetNegotiable** method, you can make the NSTextField uneditable depending on the selected movie. To do so, bind the **enabled** aspect to the "isBudgetNegotiable" key.

Instance Methods

control

– (NSControl *)**control**

Returns the receiver's control object. For EOControlAssociation, this method is equivalent to EOAssociation's **object** method.

editingAssociation

– (EOGenericControlAssociation *)**editingAssociation**

Returns **self**.

EODetailSelectionAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EODetailSelectionAssociation.h

Class Description

An EODetailSelectionAssociation binds two EODisplayGroups together through a relationship, so that the destination display group acts as an editor for that relationship. The destination display group shows all possible values for the relationship and indicates the actual members of the relationship by selecting them. The user can change the objects included in the relationship of the source by selecting and deselecting them in the destination.

EODetailSelectionAssociation is a useful alternative to EOMasterDetailAssociation and EOMasterPeerAssociation when it's more important to add and remove objects from a relationship than it is to edit the attributes of those objects.

Usable With
EODisplayGroup

Aspects	
selectedObjects	A relationship from objects in the source EODisplayGroup.

Object Keys Taken
None

Example

Suppose that an employee can be assigned any number of projects. Your application displays employees in one table view and projects in another. When an employee is selected in the first table view, the employee's assigned projects are selected in the other. To change the employee’s project assignments, a user changes the selection in the project table view: to add a project to the set, the user selects it, and to remove a project from the set, the user deselects it. To do this, in Interface Builder control-drag a connection from the Projects display group to the Employee display group. Choose EODetailSelectionAssociation in the Connections inspector, and bind the **selectedObjects** aspect to the “projects” key.

EODisplayGroup

Inherits From:	NSObject
Conforms To:	NSCoding NSObject (NSObject)
Declared In:	EOInterface/EODisplayGroup.h

Class at a Glance

Purpose

An EODisplayGroup collects an array of objects from an EODataSource, and works with a group of EOAssociation objects to display and edit the properties of those objects.

Principal Attributes

- Array of objects supplied by an EODataSource
- EOQualifier and EOSortOrderings to filter the objects for display
- Array of selection indexes
- Delegate

Creation

Interface Builder

– init	Designated initializer.

Commonly Used Methods

– allObjects	Returns all objects in the EODisplayGroup.
– displayedObjects	Returns the subset of all objects made available for display.
– selectedObjects	Returns the selected objects.
– setQualifier:	Sets a filter that limits the objects displayed.
– setSortOrderings:	Sets the ordering used to sort the objects.
– updateDisplayedObjects	Filters, sorts, and redisplays the objects.
– insertObjectAtIndex:	Creates a new object and inserts it into the EODataSource.

Class at a Glance

Class Description

An EODisplayGroup is the basic user interface manager for an Enterprise Objects Framework or Java Client application. It collects objects from an EODataSource, filters and sorts them, and maintains a selection in the filtered subset. It interacts with user interface objects and other display objects through EOAssociations, which bind the values of objects to various aspects of the display objects.

An EODisplayGroup manipulates its EODataSource by sending it **fetchObjects**, **insertObject:**, and other messages, and registers itself as an editor and message handler of the EODataSource's EOEditingContext. The EOEditingContext allows the EODisplayGroup to intercede in certain operations, as described in the EOEditors and EOMessageHandlers informal protocol specifications. EODisplayGroup implements all the methods of these informal protocols; see the descriptions for **editingContextWillSaveChanges:**, **editorHasChangesForEditingContext:**, and **editingContext:presentErrorMessage:** (**editingContextPresentException** for Java Client applications) for more information.

Most of an EODisplayGroup's interactions are with its associations, its EODataSource, and its EOEditingContext. See the EOAssociation, EODataSource, and EOEditingContext class specifications for more information on these interactions.

Creating an EODisplayGroup

You create most EODisplayGroups in Interface Builder, by dragging an entity icon from the EOModeler application, which creates an EODisplayGroup with an EODatabaseDataSource (EODistributedDataSource, for Java Client applications), or by dragging an EODisplayGroup with no EODataSource from the EOPalette. EODisplayGroups with EODataSources operate independent of other EODisplayGroups, while those without EODataSources must be set up in a master-detail association with another EODisplayGroup.

To create an EODisplayGroup programmatically, simply initialize it and set its EODataSource:

```
EODataSource *myDataSource;    /* Assume this exists. */
EODisplayGroup *myDisplayGroup;

myDisplayGroup = [[EODisplayGroup alloc] init];
[myDisplayGroup setDataSource:myDataSource];
```

After creating the EODisplayGroup, you can add associations as described in the EOAssociation class specification.

Getting Objects

Since an EODisplayGroup isn't much use without objects to manage, the first thing you do with an EODisplayGroup is send it a fetch message. You can use the basic **fetch** method; the **fetch:** action method, which can be invoked by a control in the EODisplayGroup's nib file; or, you can configure the EODisplayGroup in Interface Builder to fetch automatically when its nib file is loaded. These methods all ask the EODisplayGroup's EODataSource to fetch from its persistent store with a **fetchObjects** message.

Filtering and Sorting

An EODisplayGroup's fetched objects are available through its **allObjects** method. These objects are treated only as candidates for display, however. The array of objects actually displayed is filtered and sorted by the EODisplayGroup's delegate, or by a qualifier and sort ordering array. You set the qualifier and sort orderings using the **setQualifier:** and **setSortOrderings:** methods. The **displayedObjects** method returns this filtered and sorted array; index arguments to other EODisplayGroup methods are defined in terms of this array.

If the EODisplayGroup has a delegate that responds to **displayGroup:displayArrayForObjects:**, it invokes this method rather than using its own qualifier and sort ordering array. The delegate is then responsible for filtering the objects and returning a sorted array. If the delegate only needs to perform one of these steps, it can get the qualifier or sort orderings from the EODisplayGroup and apply either itself using the NSArray methods **filteredArrayUsingQualifier:** and **sortedArrayUsingKeyOrderArray:**, which are added by the control layer.

If you change the qualifier or sort ordering, or alter the delegate in a way that changes how it filters and sorts the EODisplayGroup's objects, you can send **updateDisplayedObjects** to the EODisplayGroup to get it to refilter and resort its objects. Note that this doesn't cause the EODisplayGroup to refetch.

Changing and Examining the Selection

An EODisplayGroup keeps a selection in terms of indexes into the array of displayed objects. EOAssociations that display values for multiple objects are responsible for updating the selection in their EODisplayGroups according to user actions on their display objects. This is typically done with the **setSelectionIndexes:** method. Other methods available for indirect manipulation of the selection are the action methods **selectNext** and **selectPrevious**, as well as **selectObjectsIdenticalTo:** and **selectObjectsIdenticalTo:**.

To get the selection, you can use the **selectionIndexes** method, which returns an array of NSNumbers, or **selectedObjects**, which returns an array containing the selected objects themselves. Another method, **selectedObject**, returns the first selected object if there is one.

The Delegate

EODisplayGroup offers a number of methods for its delegate to implement; if the delegate does, it invokes them as appropriate. Besides the aforementioned **displayGroup:displayArrayForObjects:**, there are methods that inform the delegate that the EODisplayGroup has fetched, created an object (or failed to create one), inserted or deleted an object, changed the selection, or set a value for a property. There are also methods that request permission from the delegate to perform most of these same actions. The delegate can return YES to permit the action or NO to deny it. For more information, see each method's description in the EODisplayGroupDelegate protocol specification.

Methods for Use by EOAssociations

While most of your application code interacts with objects directly, EODisplayGroup also defines methods for its associations to access properties of individual objects without having to know anything about which methods they implement. Accessing properties through the EODisplayGroup offers associations the benefit of automatic validation, as well.

Associations access objects by index into the displayed objects array, or by object identifier. **valueForObjectAtIndex:key:** returns the value of a named property for the object at a given index, and **setValue:forObjectAtIndex:key:** sets it. Similarly, **valueForObject:key:** and **setValue:forObject:key:** access the objects by object identifer. EOAssociations can also get and set values for the first object in the selection using **selectedObjectValueForKey:** and **setSelectedObjectValue:forKey:**.

Adopted Protocols

NSCoding

– encodeWithCoder:
– initWithCoder:

Method Types

Creating instances

– init

Configuring behavior

– defaultStringMatchFormat
– defaultStringMatchOperator
– fetchesOnLoad
– queryBindingValues
– queryOperatorValues
– selectsFirstObjectAfterFetch
– setDefaultStringMatchFormat:
– setDefaultStringMatchOperator:
– setFetchesOnLoad:
– setQueryBindingValues:
– setQueryOperatorValues:
– setSelectsFirstObjectAfterFetch:
– setUsesOptimisticRefresh:
– setValidatesChangesImmediately:
– usesOptimisticRefresh
– validatesChangesImmediately

Setting the data source	
	– setDataSource:
	– dataSource
Setting the qualifier and sort ordering	
	– setQualifier:
	– qualifier
	– setSortOrderings:
	– sortOrderings
Managing queries	
	– qualifierFromQueryValues
	– setEqualToQueryValues:
	– equalToQueryValues
	– setGreaterThanQueryValues:
	– greaterThanQueryValues
	– setLessThanQueryValues:
	– lessThanQueryValues
	– qualifyDisplayGroup
	– qualifyDisplayGroup:
	– qualifyDataSource
	– qualifyDataSource:
	– enterQueryMode:
	– inQueryMode
	– setInQueryMode:
	– enabledToSetSelectedObjectValueForKey:
Fetching objects from the data source	
	– fetch
	– fetch:
Getting the objects	– allObjects
	– displayedObjects
Updating display of values	
	– redisplay
	– updateDisplayedObjects
Setting the objects	
	– setObjectArray:

Changing the selection	
	– setSelectionIndexes:
	– selectObjectsIdenticalTo:
	– selectObjectsIdenticalTo:selectFirstOnNoMatch:
	– selectObject:
	– clearSelection
	– selectNext
	– selectNext:
	– selectPrevious
	– selectPrevious:
Examining the selection	
	– selectionIndexes
	– selectedObject
	– selectedObjects
Inserting and deleting objects	
	– delete:
	– deleteObjectAtIndex:
	– deleteSelection
	– insert:
	– insertedObjectDefaultValues
	– insertObjectAtIndex:
	– insertObject:atIndex:
	– setInsertedObjectDefaultValues:
Adding keys	
	– setLocalKeys:
	– localKeys
Getting the associations	
	– observingAssociations
Setting the delegate	
	– setDelegate:
	– delegate
Changing values from associations	
	– setSelectedObjectValue:forKey:
	– selectedObjectValueForKey:
	– setValue:forObject:key:
	– valueForObject:key:
	– setValue:forObjectAtIndex:key:
	– valueForObjectAtIndex:key:

Editing by associations	
	– associationDidBeginEditing:
	– association:failedToValidateValue:forKey:object:errorDescription:
	– associationDidEndEditing:
	– editingAssociation
	– endEditing
Querying changes for associations	
	– contentsChanged
	– selectionChanged
	– updatedObjectIndex
Interacting with the EOEditingContext	
	– editorHasChangesForEditingContext:
	– editingContextWillSaveChanges:
	– editingContext:presentErrorMessage:

Instance Methods

allObjects

– (NSArray *)**allObjects**

Returns all of the objects collected by the receiver.

See also: – **displayedObjects**, – **fetch**

associationDidBeginEditing:

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

Invoked by *anAssociation* when its display object begins editing to record that EOAssociation as the editing association.

See also: – **editingAssociation**, – **endEditing**, – **association:failedToValidateValue:forKey:object:errorDescription:**

associationDidEndEditing:

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

Invoked by *anAssociation* to clear the editing association. If *anAssociation* is the receiver's editing association, clears the editing association. Otherwise does nothing.

See also: – **editingAssociation**, – **endEditing**, – **association:failedToValidateValue:forKey:object: errorDescription:**

association:failedToValidateValue:forKey:object:errorDescription:

– (BOOL)**association:**(EOAssociation *)*anAssociation*
  failedToValidateValue:(NSString *)*value*
  forKey:(NSString *)*key*
  object:(id)*anObject*
  errorDescription:(NSString *)*errorDescription*

Invoked by *anAssociation* from its **shouldEndEditingForAspect:invalidInput:errorDescription:index:** method to let the receiver handle a validation error. This method opens an attention panel with *errorDescription* as the message and returns NO.

See also: – **displayGroup:shouldDisplayAlertWithTitle:message:** (EODisplayGroupDelegate)

clearSelection

– (BOOL)**clearSelection**

Invokes **setSelectionIndexes:** to clear the selection, returning YES on success and NO on failure.

contentsChanged

– (BOOL)**contentsChanged**

Returns YES if the receiver's array of objects has changed and not all observers have been notified, NO otherwise. EOAssociations use this in their **subjectChanged** methods to determine what they need to update.

See also: – **selectionChanged**, – **updatedObjectIndex**

dataSource

– (EODataSource *)**dataSource**

Returns the receiver's EODataSource.

See also: – **setDataSource:**

defaultStringMatchFormat

– (NSString *)**defaultStringMatchFormat**

Returns the format string that specifies how pattern matching will be performed on string values in the **queryMatch** dictionary. If a key in the **queryMatch** dictionary does not have an associated operator in the **queryOperatorValues** dictionary, then its value is matched using pattern matching, and the format string returned by this method specifies how it will be matched.

See also: – **defaultStringMatchOperator**, – **setDefaultStringMatchFormat:**

defaultStringMatchOperator

– (NSString *)**defaultStringMatchOperator**

Returns the operator used to perform pattern matching for string values in the **queryMatch** dictionary. If a key in the **queryMatch** dictionary does not have an associated operator in the **queryOperatorValues** dictionary, then the operator returned by this method is used to perform pattern matching.

See also: – **defaultStringMatchFormat**, – **setDefaultStringMatchOperator:**

delegate

– (id)**delegate**

Returns the receiver's delegate.

See also: – **setDelegate:**

delete:

– (void)**delete:**(id)*sender*

This action method invokes **deleteSelection**.

deleteObjectAtIndex:

– (BOOL)**deleteObjectAtIndex:**(unsigned int)*index*

Attempts to delete the object at *index*, returning YES if successful and NO if not. Checks with the delegate using **displayGroup:shouldDeleteObject:**. If the delegate returns NO, this method fails and returns NO. If successful, sends the delegate a **displayGroup:didDeleteObject:** message.

This method performs the delete by sending **deleteObject:** to the EODataSource. If that message raises an exception, this method fails and returns NO.

deleteSelection

– (BOOL)**deleteSelection**

Attempts to delete the selected objects, returning YES if successful and NO if not.

displayedObjects

– (NSArray *)**displayedObjects**

Returns the objects that should be displayed or otherwise made available to the user, as filtered by the receiver's delegate or by its qualifier and sort ordering.

See also: – **allObjects**, – **updateDisplayedObjects**, – **displayGroup:displayArrayForObjects:** (EODisplayGroupDelegate), – **qualifier**, – **sortOrderings**

editingAssociation

– (EOAssociation *)**editingAssociation**

Returns the EOAssociation editing a value if there is one, NO if there isn't.

See also: – **associationDidBeginEditing:**, – **associationDidEndEditing:**

editingContext:presentErrorMessage:

– (void)**editingContext:**(EOEditingContext *)*anEditingContext*
presentErrorMessage:(NSString *)*errorMessage*

Invoked by *anEditingContext* as part of the EOMessageHandlers informal protocol, this method presents an attention panel with *errorMessage* as the message to display.

editingContextWillSaveChanges:

– (void)**editingContextWillSaveChanges:**(EOEditingContext *)*anEditingContext*

Invoked by *anEditingContext* in its **saveChanges** method as part of the EOEditors informal protocol, this method allows the EODisplayGroup to prohibit a save operation. EODisplayGroup's implementation of this method invokes **endEditing**, and raises an NSInternalInconsistencyException if it returns NO. Thus, if there's an association that refuses to end editing, *anEditingContext* doesn't save changes.

editorHasChangesForEditingContext:

– (BOOL)**editorHasChangesForEditingContext:**(EOEditingContext *)*anEditingContext*

Invoked by *anEditingContext* as part of the EOEditors informal protocol, this method returns NO if any association is editing, YES otherwise.

enabledToSetSelectedObjectValueForKey:

– (BOOL)**enabledToSetSelectedObjectValueForKey:**(NSString *)*key*

Returns YES to indicate that a single value association (such as an EOControlAssociation for a NSTextField) should be enabled for setting *key*, NO otherwise. Normally this is the case if the receiver has a selected object. However, if *key* is a special query key (for example, "@query=.name"), then the control should be enabled even without a selected object.

endEditing

– (BOOL)**endEditing**

Attempts to end any editing taking place. If there's no editing association or if the editing association responds YES to an **endEditing** message, returns YES. Otherwise returns NO.

See also: – **editingAssociation**

enterQueryMode:

– (void)**enterQueryMode:**(id)*sender*

This action method invokes **setInQueryMode:** with an argument of YES.

equalToQueryValues

– (NSDictionary *)**equalToQueryValues**

Returns the receiver’s dictionary of equalTo query values. This dictionary is typically manipulated by associations bound to keys of the form @query=.*propertyName*. The **qualifierFromQueryValues** method uses this dictionary along with the lessThan and greaterThan dictionaries to construct qualifiers.

See also: – **setEqualToQueryValues:**, – **greaterThanQueryValues**, – **lessThanQueryValues**,

fetch

– (BOOL)**fetch**

Attempts to fetch objects from the EODataSource, returning YES on success and NO on failure.

Before fetching, invokes **endEditing** and sends **displayGroupShouldFetch:** to the delegate, returning NO if either of these methods does. If both return YES, sends a **fetchObjects** message to the receiver’s EODataSource to replace the object array, and if successful sends the delegate a **displayGroup: didFetchObjects:** message.

fetch:

– (void)**fetch:**(id)*sender*

This action method invokes **fetch**.

fetchesOnLoad

– (BOOL)**fetchesOnLoad**

Returns YES if the receiver fetches automatically after being loaded from a nib file, NO if it must be told explicitly to fetch. The default is NO. You can set this behavior in Interface Builder using the Inspector panel.

See also: – **fetch**, – **setFetchesOnLoad:**

greaterThanQueryValues

– (NSDictionary *)**greaterThanQueryValues**

Returns the receiver’s dictionary of greaterThan query values. This dictionary is typically manipulated by associations bound to keys of the form @query>.*propertyName*. The **qualifierFromQueryValues** method uses this dictionary along with the lessThan and equalTo dictionaries to construct qualifiers.

See also: – **setGreaterThanQueryValues:**, – **lessThanQueryValues**, – **equalToQueryValues**

init

– (id)**init**

Initializes a newly allocated EODisplayGroup. The new display group then needs to have an EODataSource set with **setDataSource:**. This is the designated initializer for the EODisplayGroup class. Returns **self**.

See also: – **bindAspect:displayGroup:key:** (EOAssociation)

inQueryMode

– (BOOL)**inQueryMode**

Returns YES to indicate that the receiver is in query mode, NO otherwise. In query mode, user interface controls that normally display values become empty, allowing users to type queries directly into them (this is also known as a “Query By Example” interface). In effect, the receiver’s “displayedObjects” are replaced with an empty equalTo query values dictionary. When **qualifyDisplayGroup** or **qualifyDataSource** is subsequently invoked, the query is performed and the display reverts to displaying values—this time, the objects returned by the query.

See also: – **setInQueryMode:**, – **enterQueryMode:**

insert:

– (void)**insert:**(id)*sender*

This action method invokes **insertObjectAtIndex:** with an index just past the first index in the selection, or 0 if there’s no selection.

insertedObjectDefaultValues

– (NSDictionary *)**insertedObjectDefaultValues**

Returns the default values to be used for newly inserted objects. The keys into the dictionary are the properties of the entity that the display group manages. If the dictionary returned by this method is empty, the **insert...** method adds an object that is initially empty. Because the object is empty, the display group has no value to display on the HTML page for that object, meaning that there is nothing for the user to select and modify. Use the **setInsertedObjectDefaultValues:** method to set up a default value so that there is something to display on the page.

insertObjectAtIndex:

– (id)**insertObjectAtIndex:**(unsigned int)*anIndex*

Asks the receiver's EODataSource to create a new object by sending it a **createObject** message, then inserts the new object using **insertObject:atIndex:**. The EODataSource **createObject** method has the effect of inserting the object into the EOEditingContext.

If a new object can't be created, this method sends the delegate a **displayGroup: createObjectFailedForDataSource:** message or, if the delegate doesn't respond, opens an attention panel to inform the user of the error.

See also: – **insert:**

insertObject:atIndex:

– (void)**insertObject:**(id)*anObject*
  atIndex:(unsigned int)*index*

Inserts *anObject* into the receiver's EODataSource and **displayedObjects** array at *index*, if possible. This method checks with the delegate before actually inserting, using **displayGroup:shouldInsertObject: atIndex:**. If the delegate refuses, *anObject* isn't inserted. After successfully inserting the object, this method informs the delegate with a **displayGroup:didInsertObject:** message, and selects the newly inserted object. Raises an NSRangeException if *index* is out of bounds.

Unlike the **insertObjectAtIndex:** method, this method does not insert the object into the EOEditingContext. If you use this method, you're responsible for inserting the object into the EOEditingContext yourself.

See also: – **insert:**

lessThanQueryValues

– (NSDictionary *)**lessThanQueryValues**

Returns the receiver's dictionary of lessThan query values. This dictionary is typically manipulated by associations bound to keys of the form @query<.*propertyName*. The **qualifierFromQueryValues** method uses this dictionary along with the greaterThan and equalTo dictionaries to construct qualifiers.

See also: – **setLessThanQueryValues:**, – **greaterThanQueryValues**, – **equalToQueryValues**

localKeys

– (NSArray *)**localKeys**

Returns the additional keys that EOAssociations can be bound to. An EODisplayGroup's basic keys are typically those of the attributes and relationships of its objects, as defined by their EOClassDescription through an EOEntity in the model. Local keys are typically used to form associations with key paths, with arbitrary methods of objects, or with properties of objects not associated with an EOEntity. Interface Builder allows the user to add and remove local keys in the EODisplayGroup Attributes Inspector panel.

See also: – **setLocalKeys:**

observingAssociations

– (NSArray *)**observingAssociations**

Returns all EOAssociations that observe the receiver's objects.

qualifier

– (EOQualifier *)**qualifier**

Returns the receiver's qualifier, which it uses to filter its array of objects for display when the delegate doesn't do so itself.

See also: – **updateDisplayedObjects**, – **displayedObjects**, – **setQualifier:**

qualifierFromQueryValues

– (EOQualifier *)**qualifierFromQueryValues**

Builds a qualifier constructed from entries in the three query dictionaries: equalTo, greaterThan, and lessThan. These, in turn, are typically manipulated by associations bound to keys of the form @query=.firstName, @query>.budget, @query<.budget.

See also: – **qualifyDisplayGroup**, – **qualifyDataSource**

qualifyDataSource

– (void)**qualifyDataSource**

Takes the result of **qualifierFromQueryValues** and applies to the receiver's data source. The receiver then sends itself a **fetch** message. If the receiver is in query mode, query mode is exited. This method differs from **qualifyDisplayGroup** as follows: whereas **qualifyDisplayGroup** performs in-memory filtering of already fetched objects, **qualifyDataSource** triggers a new qualified fetch against the database.

qualifyDataSource:

– (void)**qualifyDataSource:**(id)*sender*

This action method invokes **qualifyDataSource**.

qualifyDisplayGroup

– (void)**qualifyDisplayGroup**

Takes the result of **qualifierFromQueryValues** and applies to the receiver using **setQualifier:**. The method **updateDisplayedObjects** is invoked to refresh the display. If the receiver is in query mode, query mode is exited.

See also: – **qualifyDataSource**

qualifyDisplayGroup:

– (void)**qualifyDisplayGroup:**(id)*sender*

This action method invokes **qualifyDisplayGroup:**.

queryBindingValues

– (NSDictionary *)**queryBindingValues**

Returns a dictionary containing the actual values that the user wants to query upon. You use this method to perform a query stored in the model file. Bind keys in this dictionary to elements on your component that specify query values, then pass this dictionary to the fetch specification that performs the fetch.

queryOperatorValues

– (NSDictionary *)**queryOperatorValues**

Returns a dictionary of operators to use on items in the **queryMatch** dictionary. If a key in the **queryMatch** dictionary also exists in **queryOperatorValues**, that operator for that key is used.

See also: – **qualifierFromQueryValues**

redisplay

– (void)**redisplay**

Notifies all observing associations to redisplay their values.

See also: – **observingAssociations**

selectedObject

– (id)**selectedObject**

Returns the first selected object in the displayed objects array, or **nil** if there's no such object.

See also: – **displayedObjects**, – **selectionIndexes**

selectedObjects

– (NSArray *)**selectedObjects**

Returns the objects selected in the receiver's displayed objects array.

See also: – **displayedObjects**, – **selectionIndexes**

selectedObjectValueForKey:

– (id)**selectedObjectValueForKey:**(NSString *)*key*

Returns the value corresponding to *key* for the first selected object in the receiver's displayed objects array, or **nil** if exactly one object isn't selected.

See also: – **valueForObject:key:**

selectionChanged

– (BOOL)**selectionChanged**

Returns YES if the selection has changed and not all observers have been notified, NO otherwise. EOAssociations use this in their **subjectChanged** methods to determine what they need to update.

See also: – **contentsChanged**

selectionIndexes

– (NSArray *)**selectionIndexes**

Returns the indexes of the receiver's selected objects as NSNumbers , in terms of its displayed objects array.

See also: – **displayedObjects**, – **selectedObjects**, – **selectedObject**, – **setSelectionIndexes:**

selectNext

– (BOOL)**selectNext**

Attempts to select the object just after the currently selected one, returning YES if successful and NO if not. The selection is altered in this way:

- If there are no objects, does nothing and returns NO.
- If there's no selection, selects the object at index zero and returns YES.
- If the first selected object is the last object in the displayed objects array, selects the first object and returns YES.
- Otherwise selects the object after the first selected object.

See also: – **selectPrevious**, – **setSelectionIndexes:**

selectNext:

– (void)**selectNext:**(id)*sender*

This action method invokes **selectNext**.

See also: – **selectPrevious:**, – **setSelectionIndexes:**

selectObject:

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

Returns YES to indicate that the receiver has found and selected *anObject*, NO if it can't find a match for *anObject* (in which case it clears the selection). The selection is performed on the receiver's **displayedObjects**, not on **allObjects**.

selectObjectsIdenticalTo:

– (BOOL)**selectObjectsIdenticalTo:**(NSArray *)*objects*

Attempts to select the objects in the receiver's displayed objects array whose **id**s are equal to those of *objects*, returning YES if successful and NO otherwise.

See also: – **setSelectionIndexes:**

selectObjectsIdenticalTo:selectFirstOnNoMatch:

– (BOOL)**selectObjectsIdenticalTo:**(NSArray *)*objects*
    selectFirstOnNoMatch:(BOOL)*flag*

Selects the objects in the receiver's displayed objects array whose **id**s are equal to those of *objects*, returning YES if successful and NO otherwise. If no objects in the displayed objects array match *objects* and *flag* is YES, attempts to select the first object in the displayed objects array.

See also: – **setSelectionIndexes:**

selectPrevious

– (BOOL)**selectPrevious**

Attempts to select the object just before the presently selected one, returning YES if successful and NO if not. The selection is altered in this way:

- If there are no objects, does nothing and returns NO.
- If there's no selection, selects the object at index zero and returns YES.
- If the first selected object is at index zero, selects the last object and returns YES.
- Otherwise selects the object before the first selected object.

See also: – **selectNext**, – **redisplay**

selectPrevious:

– (void)**selectPrevious:**(id)*sender*

This action method invokes **selectPrevious**.

See also: – **selectNext:**, – **redisplay**

selectsFirstObjectAfterFetch

– (BOOL)**selectsFirstObjectAfterFetch**

Returns YES if the receiver automatically selects its first displayed object after a fetch if there was no selection, NO if it leaves an empty selection as-is.

See also: – **displayedObjects**, – **fetch**, – **setSelectsFirstObjectAfterFetch:**

setDataSource:

– (void)**setDataSource:**(EODataSource *)*aDataSource*

Sets the receiver's EODataSource to *aDataSource*. In the process, it performs these actions:

- Unregisters **self** as an editor and message handler for the previous EODataSource's EOEditingContext, if necessary, and registers **self** with *aDataSource's* editing context. If the new editing context already has a message handler, however, the receiver doesn't assume that role.
- Registers **self** for EOObjectsChangedInEditingContextNotification and EOInvalidatedAllObjectsInStoreNotification from the new editing context.
- Clears the receiver's array of objects.
- Sends **displayGroupDidChangeDataSource:** to the delegate if there is one.

See also: – **dataSource**

setDefaultStringMatchFormat:

– (void)**setDefaultStringMatchFormat:**(NSString *)*format*

Sets how pattern matching will be performed on NSString values in the **queryMatch** dictionary. This format is used for properties listed in the **queryMatch** dictionary that have NSString values and that do not have an associated entry in the **queryOperatorValues** dictionary. In these cases, the value is matched using pattern matching and *format* specifies how it will be matched.

The default format string for pattern matching is "**%@***" which means that the string value in the **queryMatch** dictionary is used as a prefix. For example, if the **queryMatch** dictionary contains a value "Jo" for the key "Name", the query returns all records whose name values begin with "Jo".

See also: – **defaultStringMatchFormat**, – **setDefaultStringMatchOperator:**

setDefaultStringMatchOperator:

– (void)**setDefaultStringMatchOperator:**(NSString *)*operator*

Sets the operator used to perform pattern matching for NSString values in the **queryMatch** dictionary. This operator is used for properties listed in the **queryMatch** dictionary that have NSString values and that do not have an associated entry in the **queryOperatorValues** dictionary. In these cases, the operator *operator* is used to perform pattern matching.

The default value for the query match operator is **caseInsensitiveLike**, which means that the query does not consider case when matching letters. The other possible value for this operator is **like**, which matches the case of the letters exactly.

See also: – **defaultStringMatchOperator**, – **setDefaultStringMatchFormat:**

setDelegate:

– (void)**setDelegate:**(id)*anObject*

Sets the receiver's delegate to *anObject*, without retaining it.

See also: – **delegate**

setEqualToQueryValues:

– (void)**setEqualToQueryValues:**(NSDictionary *)*values*

Sets to *values* the receiver's dictionary of equalTo query values. The **qualifierFromQueryValues** method uses this dictionary along with the lessThan and greaterThan dictionaries to construct qualifiers.

See also: – **equalToQueryValues**, – **setLessThanQueryValues:**, – **setGreaterThanQueryValues:**

setFetchesOnLoad:

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

Controls whether the receiver automatically fetches its objects after being loaded from a nib file. If *flag* is YES it does; if *flag* is NO the receiver must be told explicitly to fetch. The default is NO. You can also set this behavior in Interface Builder using the Inspector panel.

See also: – **fetch**, – **fetchesOnLoad**

setGreaterThanQueryValues:

– (void)**setGreaterThanQueryValues:**(NSDictionary *)*values*

Sets to *values* the receiver’s dictionary of greaterThan query values. The **qualifierFromQueryValues** method uses this dictionary along with the lessThan and equalTo dictionaries to construct qualifiers.

See also: – **greaterThanQueryValues**, – **setLessThanQueryValues:**, – **setEqualToQueryValues:**

setInQueryMode:

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

Sets according to *flag* whether the receiver is in query mode.

See also: – **inQueryMode**, – **enterQueryMode:**

setInsertedObjectDefaultValues:

– (void)**setInsertedObjectDefaultValues:**(NSDictionary *)*defaultValues*

Sets default values to be used for newly inserted objects. When you use the **insert...** method to add an object, that object is initially empty. Because the object is empty, there is no value to be displayed on the HTML page, meaning there is nothing for the user to select and modify. You use this method to provide at least one field that can be displayed for the newly inserted object. The possible keys into the dictionary are the properties of the entity managed by this display group. For example, a component that displays a list of movie titles and allows the user to insert new movie titles might contain these statements to ensure that all new objects have something to display as a movie title:

```
[defaultValues setObject:@"New title" forKey:@"title"];
[movies setInsertedObjectDefaultValues:defaultValues];
```

See also: – **insertedObjectDefaultValues**

setLessThanQueryValues:

– (void)**setLessThanQueryValues:**(NSDictionary *)*values*

Sets to *values* the receiver’s dictionary of lessThan query values. The **qualifierFromQueryValues** method uses this dictionary along with the greaterThan and equalTo dictionaries to construct qualifiers.

See also: – **lessThanQueryValues**, – **setGreaterThanQueryValues:**, – **setEqualToQueryValues:**

setLocalKeys:

– (void)**setLocalKeys:**(NSArray *)*keys*

Sets the additional keys to which EOAssociations can be bound to the strings in *keys*. Instead of invoking this method programmatically, you can use Interface Builder to add and remove local keys in the EODisplayGroup Attributes Inspector panel.

See also: **– localKeys**

setObjectArray:

– (void)**setObjectArray:**(NSArray *)*objects*

Sets the receiver's objects to *objects*, regardless of what its EODataSource provides. This method doesn't affect the EODataSource's objects at all; specifically, it results in neither inserts or deletes of objects in the EODataSource. *objects* should contain objects with the same property names or methods as those accessed by the receiver. This method is used by **fetch** to set the array of fetched objects; you should rarely need to invoke it directly.

After setting the object array, this method restores as much of the original selection as possible by invoking **selectObjectsIdenticalTo:**. If there's no match and the receiver selects after fetching, then the first object is selected.

See also: – **allObjects**, – **displayedObjects**, – **selectsFirstObjectAfterFetch**

setQualifier:

– (void)**setQualifier:**(EOQualifier *)*aQualifier*

Sets the receiver's qualifier to *aQualifier*. This qualifier is used to filter the receiver's array of objects for display when the delegate doesn't do so itself. Use **updateDisplayedObjects** to apply the qualifier.

If the receiver's delegate responds to **displayGroup:displayArrayForObjects:**, that method is used instead of the qualifier to filter the objects.

See also: – **displayedObjects**, – **qualifier**, – **qualifierFromQueryValues**

setQueryBindingValues:

– (void)**setQueryBindingValues:**(NSDictionary *)*values*

setQueryOperatorValues:

– (void)**setQueryOperatorValues:**(NSDictionary *)*values*

setSelectedObjectValue:forKey:

– (BOOL)**setSelectedObjectValue:**(id)*value*
    forKey:(NSString *)*key*

Invokes **setValue:forObject:key:** with the first selected object, returning YES if successful and NO otherwise. This method should be invoked only by EOAssociation objects to propagate changes from display objects.

See also: – **setValue:forObjectAtIndex:key:**, – **valueForObject:key:**

setSelectionIndexes:

– (BOOL)**setSelectionIndexes:**(NSArray *)*indexes*

Selects the objects at *indexes* in the receiver's array if possible, returning YES if successful and NO if not (in which case the selection remains unaltered). *indexes* is an array of NSNumbers . This method is the primitive method for altering the selection; all other such methods invoke this one to make the change.

This method invokes **endEditing** to wrap up any changes being made by the user. If **endEditing** returns NO, this method fails and returns NO. This method then checks the delegate with a **displayGroup: shouldChangeSelectionToIndexes:** message. If the delegate returns NO, this method also fails and returns NO. If the receiver successfully changes the selection, its observers (typically EOAssociations) each receive a **subjectChanged** message.

setSelectsFirstObjectAfterFetch:

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

Controls whether the receiver automatically selects its first displayed object after a fetch when there were no selected objects before the fetch. If *flag* is YES it does; if *flag* is NO then no objects are selected. By default, display groups select the first object after a fetch when there was no previous selection.

See also: – **displayedObjects**, – **fetch**, – **selectsFirstObjectAfterFetch**

setSortOrderings:

– (void)**setSortOrderings:**(NSArray *)*orderings*

Sets the EOSortOrdering objects that **updateDisplayedObjects** uses to sort the displayed objects to *orderings*. Use **updateDisplayedObjects** to apply the sort orderings.

If the receiver's delegate responds to **displayGroup:displayArrayForObjects:**, that method is used instead of the sort orderings to order the objects.

See also: – **displayedObjects**, – **sortOrderings**

setUsesOptimisticRefresh:

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

Controls how the receiver redisplays on changes to objects. If *flag* is YES it redisplays only when elements of its displayed objects array change; if *flag* is NO it redisplays on any change in its EOEditingContext. Because changes to other objects can affect the displayed objects (through flattened attributes or custom methods, for example), EODisplayGroups by default use the more pessimistic refresh technique of redisplaying on any change in the EOEditingContext. If you know that none of the EOAssociations for a particular EODisplayGroup display derived values, you can turn on optimistic refresh to reduce redisplay time.

The default is NO. You can also change this setting in Interface Builder's Inspector panel using the Refresh All check box.

See also: – **usesOptimisticRefresh**

setValidatesChangesImmediately:

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

Controls the receiver's behavior on encountering a validation error. Whenever an EODisplayGroup sets a value in an object, it sends the object a **validateValue:forKey:** message, allowing the object to coerce the value's type to a more appropriate one or to return an exception indicating that the value isn't valid. If this method is invoked with a *flag* of YES, the receiver immediately presents an attention panel indicating the validation error. If this method is invoked with a *flag* of NO, the receiver leaves validation errors to be handled when changes are saved. By default, display groups don't validate changes immediately.

See also: – **saveChanges** (EOEditingContext), – **validatesChangesImmediately**

setValue:forObject:key:

– (BOOL)**setValue:**(id)*value*
  forObject:(id)*anObject*
  key:(NSString *)*key*

Sets a property of *anObject*, identified by *key*, to *value*. Returns YES if successful and NO otherwise. If a new value is set, sends the delegate a **displayGroup:didSetValue:forObject:key:** message.

This method should be invoked only by EOAssociation objects to propagate changes from display objects. Other application code should interact with the objects directly.

If the receiver validates changes immediately, it sends *anObject* a **validateValue:forKey:** message, returning NO if the object refuses to validate *value*. Otherwise, validation errors are checked by the EOEditingContext when it attempts to save changes.

See also: – **setValue:forObjectAtIndex:key:**, – **setSelectedObjectValue:forKey:**, – **valueForObject: key:**, – **validatesChangesImmediately**

setValue:forObjectAtIndex:key:

– (BOOL)**setValue:**(id)*value*
  forObjectAtIndex:(unsigned int)*index*
  key:(NSString *)*key*

Invokes **setValue:forObject:key:** with the object at *index*, returning YES if successful and NO otherwise. This method should be invoked only by EOAssociation objects to propagate changes from display objects.

See also: – **setSelectedObjectValue:forKey:**,– **valueForObjectAtIndex:key:**

sortOrderings

– (NSArray *)**sortOrderings**

Returns an array of EOSortOrdering objects that **updateDisplayedObjects** uses to sort the displayed objects, as returned by the **displayedObjects** method.

See also: – **setSortOrderings:**

updateDisplayedObjects

– (void)**updateDisplayedObjects**

Recalculates the receiver's displayed objects array and redisplays. If the receiver's delegate responds to **displayGroup:displayArrayForObjects:**, it's sent this message and the returned array is set as the display group's displayed object. Otherwise, the receiver applies its qualifier and sort ordering to its array of objects. In either case, any objects that were selected before remain selected in the new displayed objects array.

See also: – **redisplay**, – **displayedObjects**, – **selectedObjects**, – **qualifier**, – **sortOrderings**

updatedObjectIndex

– (int)**updatedObjectIndex**

Returns the index in the displayed objects array of the most recently updated object, or –1 if more than one object has changed. The return value is meaningful only when **contentsChanged** returns YES. EOAssociations can use this method to optimize redisplay of their user interface objects.

usesOptimisticRefresh

– (BOOL)**usesOptimisticRefresh**

Returns YES if the receiver redisplays only when its displayed objects change, NO if it redisplays on any change in its EOEditingContext.

See also: – **setUsesOptimisticRefresh:**

validatesChangesImmediately

– (BOOL)**validatesChangesImmediately**

Returns YES if the receiver immediately handles validation errors, or NO if it leaves errors for the EOEditingContext to handle when saving changes.

See also: – **setValidatesChangesImmediately:**

valueForObject:key:

– (id)**valueForObject:**(id)*anObject*
    key:(NSString *)*key*

Returns *anObject*'s value for the property identified by *key*.

valueForObjectAtIndex:key:

– (id)**valueForObjectAtIndex:**(unsigned int)*index*
    key:(NSString *)*key*

Returns the value of the object at *index* for the property identified by *key*.

EODisplayGroupDelegate

(informal protocol)

Category Of:	NSObject
Declared In:	EOInterface/EODisplayGroup.h

Category Description

The EODisplayGroupDelegate informal protocol defines methods that an EODisplayGroup can invoke in its delegate. Delegates are not required to provide implementations for all of the methods in the informal protocol. Instead, declare and implement any subset of the methods declared in the informal protocol that you need, and use the EODisplayGroup method **setDelegate:** method to assign your object as the delegate. A display group can determine if the delegate doesn't implement a delegate method and only attempts to invoke the methods the delegate actually implements.

Method Types

Fetching objects

– displayGroupShouldFetch:
– displayGroup:didFetchObjects:
– displayGroupShouldRefetchdisplayGroup:
shouldRefetchForInvalidatedAllObjectsNotification:

Inserting, updating, and deleting objects

– displayGroup:shouldInsertObject:atIndex:
– displayGroup:didInsertObject:
– displayGroup:createObjectFailedForDataSource:
– displayGroup:didSetValue:forObject:key:
– displayGroup:shouldDeleteObject:
– displayGroup:didDeleteObject:

Managing the display

– displayGroup:shouldDisplayAlertWithTitle:message:
– displayGroupShouldRedisplaydisplayGroup:
shouldRedisplayForChangesInEditingContext:
– displayGroup:displayArrayForObjects:

Managing the selection

– displayGroup:shouldChangeSelectionToIndexes:
– displayGroupDidChangeSelection:
– displayGroupDidChangeSelectedObjects:

Changing the data source	– displayGroupDidChangeDataSource:

Instance Methods

displayGroup:createObjectFailedForDataSource:

– (void)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
  createObjectFailedForDataSource:(EODataSource *)*aDataSource*

Invoked from **insertObjectAtIndex:** to inform the delegate that *aDisplayGroup* has failed to create a new object for *aDataSource*. If the delegate doesn't implement this method, the EODisplayGroup instead runs an alert panel to inform the user of the failure.

displayGroupDidChangeDataSource:

– (void)**displayGroupDidChangeDataSource:**(EODisplayGroup *)*aDisplayGroup*

Informs the delegate that *aDisplayGroup*'s EODataSource has changed.

displayGroupDidChangeSelectedObjects:

– (void)**displayGroupDidChangeSelectedObjects:**(EODisplayGroup *)*aDisplayGroup*

Informs the delegate that *aDisplayGroup*'s set of selected objects has changed, regardless of whether the selection indexes have changed.

displayGroupDidChangeSelection:

– (void)**displayGroupDidChangeSelection:**(EODisplayGroup *)*aDisplayGroup*

Informs the delegate that *aDisplayGroup*'s selection has changed.

displayGroup:didDeleteObject:

– (void)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
  didDeleteObject:(id)*anObject*

Informs the delegate that *aDisplayGroup* has deleted *anObject*.

displayGroup:didFetchObjects:

– (void)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
didFetchObjects:(NSArray *)*objects*

Informs the delegate that *aDisplayGroup* has fetched *objects*.

displayGroup:didInsertObject:

– (void)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
didInsertObject:(id)*anObject*

Informs the delegate that *aDisplayGroup* has inserted *anObject*.

displayGroup:didSetValue:forObject:key:

– (void)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
didSetValue:(id)*value*
forObject:(id)*anObject*
key:(NSString *)*key*

Informs the delegate that *aDisplayGroup* has altered a property value of *anObject*. *key* identifies the property, and *value* is its new value.

displayGroup:displayArrayForObjects:

– (NSArray *)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
displayArrayForObjects:(NSArray *)*objects*

Invoked from **updateDisplayedObjects**, this method allows the delegate to filter and sort *aDisplayGroup*'s array of objects to limit which ones get displayed. *objects* contains all of *aDisplayGroup*'s objects. The delegate should filter any objects that shouldn't be shown and sort the remainder, returning a new array containing this group of objects. You can use the added NSArray methods **filteredArrayUsingQualifier:** and **sortedArrayUsingKeyOrderArray:** to create the new array.

If the delegate doesn't implement this method, the EODisplayGroup uses its own qualifier and sort ordering to update its displayed objects array.

See also: – **sortOrderings**, – **qualifier**, – **displayedObjects**

displayGroup:shouldChangeSelectionToIndexes:

– (BOOL)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
shouldChangeSelectionToIndexes:(NSArray *)*newIndexes*

Allows the delegate to prevent a change in selection by *aDisplayGroup*. *newIndexes* is the proposed new selection, an array of NSNumbers . If the delegate returns YES, the selection changes; if the delegate returns NO, the selection remains as it is.

displayGroup:shouldDeleteObject:

– (BOOL)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
shouldDeleteObject:(id)*anObject*

Allows the delegate to prevent *aDisplayGroup* from deleting *anObject*. If the delegate returns YES, *anObject* is deleted; if the delegate returns NO, the deletion is abandoned.

displayGroup:shouldDisplayAlertWithTitle:message:

– (BOOL)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
shouldDisplayAlertWithTitle:(NSString *)*title*
message:(NSString *)*message*

Allows the delegate to prevent *aDisplayGroup* from displaying an attention panel with *title* and *message*. The delegate can return YES to allow *aDisplayGroup* to display the panel, or NO to prevent it from doing so (perhaps displaying a different attention panel).

displayGroupShouldFetch:

– (BOOL)**displayGroupShouldFetch:**(EODisplayGroup *)*aDisplayGroup*

Allows the delegate to prevent *aDisplayGroup* from fetching. If the delegate returns YES, *aDisplayGroup* performs the fetch; if the delegate returns NO, *aDisplayGroup* abandons the fetch.

displayGroup:shouldInsertObject:atIndex:

– (BOOL)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
shouldInsertObject:(id)*anObject*
atIndex:(unsigned int)*anIndex*

Allows the delegate to prevent *aDisplayGroup* from inserting *anObject* at *anIndex*. If the delegate returns YES, *anObject* is inserted; if the delegate returns NO, the insertion is abandoned.

displayGroup:shouldRedisplayForChangesInEditingContext:

– (BOOL)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
  shouldRedisplayForEditingContextChangeNotification:(NSNotification *)*aNotification*

Invoked whenever *aDisplayGroup* receives an EOObjectsChangedInEditingContextNotification, this method allows the delegate to suppress redisplay based on the nature of the change that has occurred. If the delegate returns YES, *aDisplayGroup* redisplays; if it returns NO, *aDisplayGroup* doesn't. *aNotification* supplies the EOEditingContext that has changed, as well as which objects have changed and how. See the EOEditingContext class specification for information on EOObjectsChangedInEditingContextNotification.

See also: – **redisplay**

displayGroup:shouldRefetchForInvalidatedAllObjectsNotification:

– (BOOL)**displayGroup:**(EODisplayGroup *)*aDisplayGroup*
  shouldRefetchForInvalidatedAllObjectsNotification:(NSNotification *)*aNotification*

Invoked whenever *aDisplayGroup* receives an EOInvalidatedAllObjectsInStoreNotification, this method allows the delegate to suppress refetching of the invalidated objects. If the delegate returns YES, *aDisplayGroup* immediately refetches its objects. If the delegate returns NO, *aDisplayGroup* doesn't immediately fetch, instead delaying until absolutely necessary. *aNotification* is an NSNotification. See the EOObjectStore and EOEditingContext class specifications for information on this notification.

EOGenericControlAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOControlAssociation.h

Class Description

EOGenericControlAssociation is the abstract superclass of EOControlAssociation and EOActionCellAssociation. You never use instances of this class directly; its **isUsableWithObject:** method always returns NO. See the subclass specifications for more information.

Usable With	Aspects	Object Keys Taken
Nothing	value	target
	enabled	delegate

Instance Methods

control

– (NSControl *)**control**

Overridden by subclasses to return the receiver's display object—an NSControl (Application Kit).

editingAssociation

– (EOGenericControlAssociation *)**editingAssociation**

Overridden by subclasses to return the association responsible for handling text delegation messages. For example, if the display object is a NSMatrix or NSTableView (Application Kit), this method returns the association for the cell being edited.

EOInterfaceController

Inherits From:	java.lang.Object
Implements:	EOKeyValueCodingAdditions (EOControl) EOObserving (EOControl) java.awt.event.WindowListener
Package:	com.apple.client.eointerface

Class Description

EOInterfaceController serves as a convenient base class for logic related to the interface of client-side applications. When the WebObjectsApplication wizard in Project Builder creates a new client-side interface, it adds to the client-side subproject an Interface Builder nib file representing this interface and a skeletal EOInterfaceController subclass defined as the nib file's root object or "owner."

In an application constructed in conformance to the Model-View-Controller paradigm, EOInterfaceController plays the role of controller. It has four special outlets: its **editingContext**, its **component**, its **masterDisplayGroup**, and its **controllerDisplayGroup**, all of which you can configure using Interface Builder. The object identified by **component** is an AWT Component that functions as the view, since it is the main entry point into the user interface. Because an enterprise objects must always inhabit an editing context, **editingContext** and its contents serve as the "model." The **masterDisplayGroup** is an EODisplayGroup containing the "master" enterprise objects manipulated by the controller's user interface (which may well involve many other display groups). The **controllerDisplayGroup** is a convenience instance containing nothing but the interface controller itself.

Constructors

public **EOInterfaceController**()
public **EOInterfaceController**(com.apple.client.eocontrol.EOEditingContext *substitutionContext*)
public **EOInterfaceController**(com.apple.client.eocontrol.EOEditingContext *substitutionContext*,
java.lang.String *archiveName*)

The two-argument constructor is EOInterfaceController's designated constructor. It initializes a new instance and then attempts to load the associated EOArchive indentified by *archiveName*, using *substitutionContext* as EOEditingContext's substitution editing context during the load. Archive loading results in the restoration of any controller outlets such as **editingContext** and **component** connected within InterfaceBuilder. The remaining constructors are convenience constructions that invoke the designated constructor with **null**s substituted for the missing arguments. These **null** values are passed along in the calls to **loadArchive** and EOEditingContext's **setSubstitutionEditingContext** .

Instance Methods

closeWindow

public void **closeWindow**()

Puts any window created through the **runInFrame** or **runInModalDialog** methods offscreen and disposes of the window once the Component has been removed.

See also: – **component**

collectChangesFromServer

public void **collectChangesFromServer**()

Updates the receiver's editing context to reflect any changes to enterprise objects pending on the server.

component

public java.awt.Component **component**()

Returns the main entry point into the receiver's user interface, which is always an AWT Component.

controllerDisplayGroup

public EODisplayGroup **controllerDisplayGroup**()

Returns an EODisplayGroup containing nothing but the receiver (EOInterfaceController implements EOKeyValueCodingAdditions). You typically instantiate and connect this display group with Interface Builder. This display group facilitates the use of the receiver with EOAssociations, as any properties defined in a controller subclass may then be used as association aspect keys.

For example, to enable or disable the entire user interface, you could add a "uiEnabled" property to an EOInterfaceController subclass and bind the "enabled" aspect of any EOAssociation with display objects in its component to the key of the controller display group.

displayGroupDidChangeDataSource

public void **displayGroupDidChangeDataSource**(EODisplayGroup *aDisplayGroup*)

Overriden by subclasses to respond to a change in *aDisplayGroup*'s data source. In the **setMasterDisplayGroup** and **setControllerDisplayGroup** methods EOInterfaceController sets itself as the delegate of its master and controller display groups if no other object has claimed delegate status. The default implementation of this delegation method simply invokes **redisplayControllerDisplayGroup**.

displayGroupDidChangeSelection

public void **displayGroupDidChangeSelection**(EODisplayGroup *aDisplayGroup*)

Overriden by subclasses to respond to a selection change in *aDisplayGroup*. In the **setMasterDisplayGroup** and **setControllerDisplayGroup** methods EOInterfaceController sets itself as the delegate of its master and controller display groups if no other object has claimed delegate status. The default implementation of this delegation method simply invokes **redisplayControllerDisplayGroup**.

displayGroupDidDeleteObject

public void **displayGroupDidDeleteObject**(EODisplayGroup *aDisplayGroup*,
java.lang.Object *anObject*)

Overriden by subclasses to respond to the deletion of *anObject* in *aDisplayGroup*. In the **setMasterDisplayGroup** and **setControllerDisplayGroup** methods EOInterfaceController sets itself as the delegate of its master and controller display groups if no other object has claimed delegate status. The default implementation of this delegation method simply invokes **redisplayControllerDisplayGroup**.

displayGroupDidFetchObjects

public void **displayGroupDidFetchObjects**(EODisplayGroup *aDisplayGroup*, NSArray *anArray*)

Overriden by subclasses to respond to the fetch of objects in *anArray* by *aDisplayGroup*. In the **setMasterDisplayGroup** and **setControllerDisplayGroup** methods EOInterfaceController sets itself as the delegate of its master and controller display groups if no other object has claimed delegate status. The default implementation of this delegation method simply invokes **redisplayControllerDisplayGroup**.

displayGroupDidInsertObject

public void **displayGroupDidInsertObject**(EODisplayGroup *aDisplayGroup*,
java.lang.Object *anObject*)

Overriden by subclasses to respond to the insertion of object *anObject* into *aDisplayGroup*. In the **setMasterDisplayGroup** and **setControllerDisplayGroup** methods EOInterfaceController sets itself as the delegate of its master and controller display groups if no other object has claimed delegate status. The default implementation of this delegation method simply invokes **redisplayControllerDisplayGroup**.

displayGroupDidSetValueForObject

public void **displayGroupDidSetValueForObject**(EODisplayGroup *aDisplayGroup*, java.lang.Object *anObject*, java.lang.Object *anObject*, java.lang.String *aString*)

Overriden by subclasses to *aDisplayGroup*. In the **setMasterDisplayGroup** and **setControllerDisplayGroup** methods EOInterfaceController sets itself as the delegate of its master and controller display groups if no other object has claimed delegate status. The default implementation of this delegation method simply invokes **redisplayControllerDisplayGroup**.

editingContext

public com.apple.client.eocontrol.EOEditingContext **editingContext**()

Returns the EOEditingContext established through **setEditingContext**, which should also be that of the master display group's data source. All manipulation of enterprise objects must occur within an editing context.

See also: – **masterDisplayGroup**

handleEditingContextChanges

public void **handleEditingContextChanges**(NSNotification *aNotification*)

Implemented by observers of EOEditingContext's ObjectsChangedInEditingContext and EditingContextDidSaveChanges notifications; when **setEditingContext** is invoked, the receiver is automatically registered as an observer. The default implementation simply invokes **redisplayControllerDisplayGroup** and **updateWindowTitle**.

handleWindowClosing

protected void **handleWindowClosing**()

Invoked when the window containing the receiver's component is about to close. The default implementation invokes **saveIfUserConfirmsAndCloseWindow**.

See also: – **component**

insertIntoControllerDisplayGroup

protected void **insertIntoControllerDisplayGroup**()

Makes the receiver the only object in the controller display group and selects it. This method is invoked whenever **setControllerDisplayGroup** is invoked.

isEdited

public boolean **isEdited**()

Returns whether the receiver's editing context has changes and thus whether the receiver is "dirty."

See also: – **editingContext**

isRunning

public boolean **isRunning**()

Returns whether **component** currently has a parent.

isRunningInContainer

public boolean **isRunningInContainer**()

Returns **true** if no JFrame or JDialog has been instantiated by the receiver but **component** still has a parent.

isRunningInFrame

public boolean **isRunningInFrame**()

Returns **true** if **runInFrame** has previously been invoked but **closeWindow** has not.

isRunningInModalDialog

public boolean **isRunningInModalDialog**()

Returns **true** if **runInModalDialog** has previously been invoked but **closeWindow** has not.

loadArchive

protected void **loadArchive**()

Convenience method equivalent invoking the following form with a null parameter.

protected void **loadArchive**(java.lang.String *archiveName*)

Loads a new instance of the EOArchive named *archiveName* with the receiver as its owner and a null archive package name, throwing if the attempt fails (see EOArchive's **loadArchiveNamed**). If *archiveName* is null the name of the receiver's class will be used instead.

locateWindow

protected void **locateWindow**(java.awt.Window *window*)

Invoked within **runInFrame** and **runInModalDialog**, this method positions *window* at its appropriate initial location, center screen in the default implementation.

masterDisplayGroup

public EODisplayGroup **masterDisplayGroup**()

Returns an EODisplayGroup containing the "master" enterprise objects primarily manipulated by the receiver's user interface (the EOArchive associated with the receiver may well contain additional display groups).

A component containing a display of Studios, for example, might contain additional displays of Movies the selected Studio has produced and Talent in its stable, but the Studio EODisplayGroup would drive these details and therefore be the "master."

masterObject

public com.apple.client.eocontrol.EOEnterpriseObject **masterObject**()

Returns that single enterprise object currently selected in the receiver's **masterDisplayGroup**.

masterObjectGlobalID

public com.apple.client.eocontrol.EOGlobalID **masterObjectGlobalID**()

Returns the EOGlobalID of the **masterObject**.

objectWillChange

public void **objectWillChange**(java.lang.Object *object*)

Actually EOObserverCenter's notification hook, this method is implemented by EOInterfaceController in order to invoke **redisplayControllerDisplayGroup** whenever *object* is the receiver.

redisplayControllerDisplayGroup

public void **redisplayControllerDisplayGroup**()

Invoked whenever the contents or selection of **controllerDisplayGroup** changes (see **objectWillChange**), this method sends the display group a **redisplay** message.

run

public void **run**()

A "presentation-neutral" form of the following three more specific editions, this method is intended to be invoked when the consumer is content to leave **component** presentation details to the receiver. The default implementation invokes **runInFrame**.

runInContainer

public void **runInContainer**(java.awt.Container *container*)

Adds the receiver's **component** to *container.*

runInFrame

public void **runInFrame**()

Instantiates a JFrame containing the receiver's **component** and makes it visible atop the window stack.

runInModalDialog

public void **runInModalDialog**()

Instantiates a modal JDialog containing the receiver's **component** and makes it visible atop the window stack.

save

public boolean **save**()

Sends **editingContext** a **saveChanges** message with the receiver as its sender and presents an error dialog containing any exception if this invocation fails. Returns **true** if **saveChanges** succeeds, **false** otherwise.

saveAndCloseWindow

public boolean **saveAndCloseWindow**()

Invokes **save** and, upon success, **closeWindow**, returning **save**'s result.

saveIfUserConfirms

public boolean **saveIfUserConfirms**()

Convenience method invoking the following two parameter form with a null *dialogTitle* and *message*.

public boolean **saveIfUserConfirms**(java.lang.String *dialogTitle*, java.lang.String *message*)

Invokes **save** once the user has confirmed this operation in a dialog titled *dialogTitle* containing *message* (both arguments take on default values if null). Returns **true** if the operation is confirmed and succeeds, **false** otherwise.

saveIfUserConfirmsAndCloseWindow

public boolean **saveIfUserConfirmsAndCloseWindow**()

Invokes the two parameter form of **saveIfUserConfirms** with the *dialogTitle* "Close" and a null *message*. If the operation is confirmed, **closeWindow** is invoked. Returns the result of **saveIfUserConfirms**.

setComponent

public void **setComponent**(java.awt.Component *component*)

Establishes *component* as the main entry point into the receiver's user interface. If component is a Window or RootPaneContainer, the receiver's **component** will actually wind up being a new EOView containing its subcomponents rather than *component*, as the appropriate root container will be determined by which **run...** method is subsequently invoked. If *component* is not a Window it will be removed from any existing parent.

setControllerDisplayGroup

public void **setControllerDisplayGroup**(EODisplayGroup *displayGroup*)

Typically invoked only by EOArchive in re-establishing a connection made in the receiver's corresponding InterfaceBuilder document, this method establishes *displayGroup* as the EODisplayGroup which will vend the receiver's keys (also see **controllerDisplayGroup**). The default implementation invokes **insertIntoControllerDisplayGroup** before making the receiver both an observer of *displayGroup* and its delegate in the absence of any other.

setEditingContext

public void **setEditingContext**(com.apple.client.eocontrol.EOEditingContext *editingContext*)

Typically invoked only by EOArchive in re-establishing a connection made in the receiver's corresponding InterfaceBuilder document, this method establishes *editingContext* as the EOEditingContext for any manipulated enterprise objects (also see **editingContext**). This should be the same as that of **masterDisplayGroup**'s **dataSource**. The default implementation adds the receiver as a recipient of ObjectsChangedInEditingContext and EditingContextDidSaveChanges notifications sent to **handleEditingContextChanges**.

setMasterDisplayGroup

public void **setMasterDisplayGroup**(EODisplayGroup *displayGroup*)

Establishes *displayGroup* as the EODisplayGroup containing the "master" enterprise objects manipulated in **component** (also see **masterDisplayGroup**).

setMasterWithGlobalID

public void **setMasterWithGlobalID**(com.apple.client.eocontrol.EOGlobalID *gid*)

Attempts to retrieve that enterprise object with *gid* from the EOEditingContext of **masterDisplayGroup**'s **dataSource**. If successful, this object is set as **masterDisplayGroup**'s new contents and selection. If not, the **masterDisplayGroup** will be emptied.

setMasterWithObject

public void **setMasterWithObject**(com.apple.client.eocontrol.EOEnterpriseObject *anEO*)

Retrieves *anEO*'s EOGlobalID from its editing context and invokes **setMasterWithGlobalID**.

setTitle

public void **setTitle**(java.lang.String *title*)

Sets the receiver's title to *title*. Note that this value will be used in constructing the receiver's **windowTitle**. Invocations of this method will have no effect on window titles until changes occur in the receiver's editing context.

showWindow

public void **showWindow**()

If the receiver's root container is a Window, this method makes it visible atop the window stack.

title

public java.lang.String **title**()

Returns any title explicitly set for the receiver via **setTitle**.

updateWindowTitle

public void **updateWindowTitle**()

Sets the title of any Window created by the receiver to the current value of **windowTitle**.

window

public java.awt.Window **window**()

Returns the Window created to contain the receiver's **component**, which will only be non-null if **runInFrame** or **runInModalDialog** have previously been invoked.

windowTitle

protected java.lang.String **windowTitle**()

Returns **title** if it has been explicitly set or an de-packaged, prettified edition of the receiver's class name. Both will be prefixed by an asterisk if **isEdited** currently returns **true**.

EOMasterCopyAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOMasterCopyAssociation.h

Class Description

An EOMasterCopyAssociation object synchronizes two EODisplayGroups that share the same data source but have different qualifiers. By binding two display groups with an EOMasterCopyAssociation, any changes performed in one display group are immediately reflected in the other. Similarly, changing the selection in one display group immediately changes it in the other one.

Usable With
EODisplayGroup

Aspects	
parent	An EODisplayGroup with which the association's display group should be synchronized.

Object Keys Taken
None

Examples

Suppose you have an EODisplayGroup for displaying Talent objects (actors and directors) and another display group for displaying the pictures of the Talents who are actors. When a Talent is selected in the first display group, you want the "actor" display group to select that Talent's picture if the selected Talent is an actor. Since both display groups manage Talent objects, they can share the same EODataSource. However, the first display group is unqualified—it fetches all Talent objects; the second display group is qualified to fetch only the Talents who are actors.

To do this, in Interface Builder, start with an unqualified display group for displaying all the Talents. Drag a second display group from the Enterprise Objects palette into your nib. Control-drag a connection from the new display group to the unqualified Talent display group. In the Connections inspector, choose EOMasterCopyAssociation, select the **parent** aspect, and click Connect. This action automatically sets the second display group's data source. Initially, the data source is set to an EODetailDataSource—that's what you'll see in Interface Builder. However, at runtime, the association switches the second display group's data source to that of the **parent** display group.

Now when you run the application, the display groups will be synchronized with one another. (You'll programmatically assign a qualifier to the second display group so that it filters out non-actor Talents.)

EOMasterDetailAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOMasterDetailAssociation.h

Class Description

An EOMasterDetailAssociation object binds one EODisplayGroup (the detail) to a relationship in another (the master), so that the detail display group contains the destination objects for the object selected in the master. The display groups' data sources also operate in a master-detail arrangement, meaning changes to one are immediately reflected in the other. In this arrangement, the detail EODisplayGroup's data source must be an EODetailDataSource. The detail objects are taken directly from the selected object in the master EODisplayGroup, so that changes to the objects in one EODisplayGroup are instantly reflected in the other.

In Yellow Box, by contrast, with an EOMasterPeerAssociation, the two EODisplayGroups are independent of each other (EOMasterPeerAssociation is not a Java Client class). In a master-peer setup, insertions and deletions in the detail EODisplayGroup don't affect the corresponding relationship property of the selected object in the master EODisplayGroup. Master-peer setups are more appropriate when no insertions or deletions will be made in the detail EODisplayGroup. See the EOMasterPeerAssociation class specification for more information.

Usable With

EODisplayGroups whose data sources are EODetailDataSources

Aspects

parent (Yellow Box) ParentAspect (Java Client)	A relationship from the master EODisplayGroup.

Example

Suppose you have a master EODisplayGroup displaying Movie objects and a detail display group displaying Talent objects. The two display groups are bound to one another through Movie's **directors** relationship—a to-many relationship from Movie to Talent. When a Movie is selected, you want the Talent display group to display the Talents who directed the Movie. Inserting a new director into the Talent display group should add the director to the selected Movie's **directors** relationship; and similarly, deleting a director from the Talent display group should remove the director from the selected Movie's **directors** relationship.

To do this, in Interface Builder, control-drag a connection from the Talent display group to the Movie display group. In the Connections inspector, choose EOMasterDetailAssociation, and bind **parent** aspect to the "directors" key.

Instance Methods

EOMasterPeerAssociation

Inherits From:	EOMasterDetailAssociation : EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOMasterDetailAssociation.h

Class Description

An EOMasterPeerAssociation binds two EODisplayGroups together in a master-detail relationship, where the detail EODisplayGroup shows the destination objects for the relationship of the master EODisplayGroup. In a master-peer arrangement, the detail display group's data source is independent. Detail objects are fetched independently from the detail's data source, which means that changes to one display group aren't automatically reflected in the other. To update the other display group, it's necessary to save the changes made and then have the other display group fetch its objects anew.

Contrast this with a master-detail setup using an EOMasterDetailAssociation. With an EOMasterDetailAssociation, the display groups' data sources also operate in a master-detail arrangement, meaning changes to one are immediately reflected in the other. The detail objects are taken directly from the selected object in the master display group, so that changes to the objects in one display group are instantly reflected in the other. Master-peer setups display these advantages over master-detail setups:

- You can use them to display the destination objects for relationships that are defined in the model but not declared as class properties. This is typically done for rarely accessed information—or information that's costly to access. By not defining the relationship as a class property, the destination objects aren't stored as instance variables in the source objects, which saves memory and the cost of constructing faults for the relationship.
- Because the detail display group fetches objects with its own data source, you can configure the detail data source with an auxiliary EOQualifier to limit the objects fetched. This further reduces the cost of fetching data.
- You can use an EOMasterPeerAssociation to fetch detail information that may be updated in another editing context or even in another application; thus this association helps you to remain "up to date" with the database.

Generally, master-peer setups are only appropriate when no insertions or deletions will be made in the detail display group. For a master-detail relationship that reflects changes between two display groups, including insertions and deletions, use an EOMasterDetailAssociation.

Usable With

EODisplayGroups whose data sources are not EODetailDataSources

Aspects

parent	A relationship from the master EODisplayGroup.

Object Keys Taken

None

Example

Suppose you have a database of salesmen and their associated sales. Each salesman has a city ID. The sales are related to the salesmen by salesman ID, but also have a city ID. You want a list of all the sales in a salesman's city so you could evaluate it against other salesmen. For this, you create a relationshipo between salesman and sales based on city ID (the relationship is not a class property). You can then display that information using an EOMasterPeerAssociation.

EOMatrixAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOMatrixAssociation.h

Class Description

An EOMatrixAssociation allows you to populate an NSMatrix's cells (Application Kit). EOMatrixAssociation supports connections for both cell titles and icons, depending on the matrix's prototype cell. You define the prototype in Interface Builder (to display an icon only, text only, or both).

Usable With

NSMatrix (Application Kit)

Aspects

enabled	A boolean attribute of the objects, which determines whether the matrix is enabled.
image	An NSImage attribute of the objects to display in the cell.
title	An attribute of the objects to display in the cell.

Object Keys Taken

target	On receiving an action message from the matrix, an EOMatrixAssociation updates its display group's selection.

Examples

Suppose that you want to display actors' names and pictures in an NSMatrix. Start with a TalentPhoto display group (where a TalentPhoto object has a relationship to its Talent object). In interface builder, create a button containing both an image and text. Then, alternate-drag to create a matrix of buttons. Control-drag from the matrix to the photo display group. In the Connections inspector, choose EOMatrixAssociation, and bind the **image** aspect to the **photo** attribute. Repeat, binding the **title** aspect to the **talent.lastName** attribute.

Note that you can group the matrix in a scroll view. An EOMatrixAssociation will automatically manage the size of the matrix for this (for vertical scrolling only).

EOPickTextAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOPickTextAssociation.h

Class Description

An EOPickTextAssociation takes the value of its display object, an NSControl (Application Kit), and uses it to form a qualifier with up to three LIKE operators, each compared to a different key of the EODisplayGroup. This allows the user to perform a similarity search based on whole or partial values.

EOPickTextAssociations are most often used with a table view to qualify a list of fetched objects that is too long for convenient scrolling.

Usable With
Any NSControl

Aspects

matchKey1	An attribute to match using a LIKE qualifier.
matchKey2	An attribute to match using a LIKE qualifier.
matchKey3	An attribute to match using a LIKE qualifier.

Object Keys Taken

target	The EOPickTextAssociation applies its qualifier when sent an action message from the NSControl.
delegate	The EOPickTextAssociation applies its qualifier when sent a **controlTextDidChange:** message, causing dynamic update as the user types.

Example

Make an EOPickTextAssociation between an NSTextField and an EODisplayGroup of People objects. Bind the **matchKey1** and **matchKey2** aspects to the “lastName” and “firstName” keys. If the user types “Bi” in the field, the EOPickTextAssociation applies the following qualifier to the EODisplayGroup:

```
(lastName like "*Bi*") OR (firstName like "*Bi*")
```

which matches names like “Bill Smith” and “Joe Biggs”. The list of objects displayed in the display group is restricted to those that match the qualifier.

EOPopUpAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOPopUpAssociation.h

Class Description

An EOPopUpAssociation object displays an attribute or to-one relationship value in an NSPopUpButton (Application Kit). The items in the NSPopUpButton can be entered manually, or for a relationship, constructed dynamically from values supplied by the destination entity's EODisplayGroup. The value displayed by the NSPopUpButton can be bound by one of three aspects: **selectedTitle**, which is useful for values representable as strings; **selectedTag**, for integer values; and **selectedObject**, for the destination object of a relationship.

Usable With

NSPopUpButton (Application Kit)

Aspects

titles	An attribute of the objects in an EODisplayGroup whose values can be represented as strings.
selectedTitle	An attribute of the selected object whose values can be represented as strings.
selectedTag	An integer attribute of the selected object.
selectedObject	A to-one relationship of the selected object; the value displayed is that for the attribute bound to the **titles** aspect.
enabled	A boolean attribute of the selected object, which determines whether the NSPopUpButton is enabled.

Object Keys	Taken
target	When the user chooses an item in the pop-up list, the EOPopUpAssociation updates the selected object's property with the item's title, tag, or object.

Examples

There are several basic ways to configure a combo box and it's association. They are described below.

Selecting a String from a Static List

Suppose you have a Movie display group and you want to provide a pop-up list for setting the rating from a static list of strings. In this example, a Movie object's rating is a string property rather than a relationship to a Rating object. To do this, in Interface Builder, type the list of ratings into the pop-up list. Control-drag a connection from the pop-up list to the Movie display group. Choose EOPopUpAssociation in the Connections inspector, and bind the **selectedTitle** aspect to the "rating" key. With this configuration, if an object's string attribute value isn't in the pop-up list, it's temporarily added while the object is selected.

Selecting a String from a Dynamic List

This example is similar to the previous one, except in this example, a Movie object's rating is chosen from strings in a Rating database table. There's a Rating EODisplayGroup that fetches the ratings into Rating objects, and the pop-up list is filled from the "ratingString" property of the rating display group's Rating objects. To do this, in Interface Builder, control-drag a connection from the pop-up list to the Ratings display group. Choose EOPopUpAssociation in the Connections inspector, and bind the **titles** aspect to the "ratingString" key. Similarly, control-drag a connection from the pop-up list to the Movie display group. Again choose EOComboBoxAssociation in the Connections inspector, and bind the **selectedTitle** aspect to the "rating" key.

Selecting an Integer Tag from a Static List

Suppose you have a Customer enterprise object whose credit card type (Visa, MasterCard, and so on) is indicated by an integer tag. You want a user to be able to choose a customer's card type from a pop-up list. To do this, in Interface Builder, set the credit card names and tags for the pop-up list. Control-drag a connection from the pop-up list to the Customer display group. Choose EOPopUpAssociation in the Connections inspector, and bind the **selectedTag** aspect to the "cardType" key. You can also allow for a general "other" value by defining a special tag and setting it in the EOPopUpAssociation using **setTagValueForOther:**. Credit card tags from the database not matching any in the pop-up list are then displayed as the "other" value. (It would also make sense to disable the pop-up list in this case, to avoid writing the meaningless tag back to the database.)

Selecting the Destination of a To-One Relationship

Suppose you have a list of employees and want to assign each employee a department. In terms of the object model, you want to assign a Department object as the destination of an Employee object's **department** relationship. To do this, in Interface Builder, control-drag a connection from the pop-up list to a Department display group. Choose EOComboBoxAssociation in the Connections inspector, and bind the **titles** aspect to the "name" key. Similarly, control-drag a connection from the pop-up list to the Employee display group. Again choose EOComboBoxAssociation in the Connections inspector, and bind the **selectedObject** to the "department" key. This fills the pop-up list with the names of departments, and causes the name of the selected Employee's Department to be selected in the pop-up list.

Instance Methods

setTagValueForOther:

– (void)**setTagValueForOther:**(int)*tag*

Records *tag* as the "unknown" tag. When a property value doesn't match any other tag in the pop-up list, the EOPopUpAssociation automatically selects the item for this tag. If there's no item for this tag, the pop-up list's selection isn't changed. This tag value is by default –1.

tagValueForOther

– (int)**tagValueForOther**

Returns the "unknown" tag.

EORadioMatrixAssociation

Inherits From:	EOAssociation : EODelayedObserver : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EORadioMatrixAssociation.h

Class Description

EORadioMatrixAssociation displays a string or an integer in an NSMatrix. EORadioMatrixAssociation includes three aspects: **selectedTitle**, which is useful for values representable as strings; **selectedTag**, for integer values; and **enabled** for enabling and disabling the NSMatrix.

Purpose
An EORadioMatrixAssociation binds titles or tags of controls in an NSMatrix to string or integer attributes.

Aspects

selectedTitle	An attribute of the selected object whose values can be represented as strings.
selectedTag	An integer attribute of the selected object.
enabled	A boolean attribute of the selected object, which determines whether the matrix is enabled.

Object Keys Taken

target	When the user chooses an item in the matrix, the EORadioMatrixAssociation updates the selected object's property with the item's title or tag.

Instance Methods

setTagValueForOther:

– (void)**setTagValueForOther:**(int)*tag*

Records *tag* as the “unknown” tag. When a property value doesn’t match any other tag in the matrix, the EORadioMatrixAssociation automatically selects the item for this tag. If there’s no item for this tag, the radio button selection isn’t changed. This tag value is by default –1.

tagValueForOther

– (int)**tagValueForOther**

Returns the “unknown” tag.

EORecursiveBrowserAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EORecursiveBrowserAssociation.h

Class Description

An EORecursiveBrowserAssociation is the default association for use with a multi-column NSBrowser (Application Kit). EORecursiveBrowserAssociation manages hierarchical structures, such as a company's management chain—the first column is filled with top-level managers, the second column is filled with the employees who report directly to the selected top-level manager, and so on.

Usable With

NSBrowser (Application Kit)

Aspects

rootChildren	An array of objects with which to fill the browser's first column.
title	An attribute of objects to display in the browser's cells.
isLeaf	A boolean attribute of objects that determines whether the corresponding browser cell is a leaf (YES) or a branch (NO).
children	An NSArray attribute of the selected object, with which to fill the next column. This aspect is only used when the selected object is a branch (responds NO to **isLeaf**).

Object Keys Taken

target	used to handle user click actions within the browser. The association sends the proper synchronization msg to the DG.
delegate	used to fill in the values of the browser

Example

Suppose you want to display a company's management structure in a browser. Start with a display group for Employee objects. Programmatically qualify this display group to fetch only the top-level management (the Employees with which to fill the browser's first column).

Drag a browser into a window. Be sure to set it to "Allow branch selection." Control-drag from the browser to your Employee display group. In the Interface Builder's Connections Inspector (EORecursiveBrowserAssociation—labeled EORecBrowser—is chosen by default), bind the **rootChildren** aspect to Employee's **directReports** relationship (a recursive, to-many relationship). Making this binding has the effect of:

- Creating a new display group named "LastEmployeeColumn." More generally, the new display group has a name of the form, "Last*NameOfFirstDisplayGroup*Column."
- Preconnecting the new display group to a data source.
- Binding the EORecursiveBrowserAssociation's **children** aspect to the **directReports** relationship—the same relationship used for the **rootChildren** aspect.

Now bind the **title** and **isLeaf** aspects. (Note that if you try to bind these aspects before you bind the **rootChildren** aspect, you'll bypass work that the association can do for you automatically.) Control-drag from the browser to either of the display groups, and bind the association's **title** aspect to the **fullName** key and the **isLeaf** aspect to the **isIndividualContributor** key (a method that returns NO if the Employee is a manager with direct reports). It doesn't matter what display group you make these bindings to, because the association expects **rootChildren** and **children** to reference the same kind of objects (have the same keys).

Now the association populates the browser's columns based on the selection in the previous column. You might want to create a master-detail association between the *LastColumn* display group and another display group. For example, the Employees application might display information about the employee selected in the browser's right-most column.

The rootChildren Aspect

When you bind an EORecursiveBrowserAssociation's **rootChildren** aspect, the association assumes that **children** will be bound to the same key. However, it's possible for you to bind these aspects to different keys. If you want to do this, you'll have to disconnect the **children** binding that the association creates automatically, and then rebind it to the key you want to use. Note that you only have this freedom with the first column. Subsequent columns must all use the same key to satisfy the **children** aspect.

EOTableViewAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOColumnAssociation.h

Class Description

An EOTableViewAssociation object manages the individual EOColumnAssociations between an NSTableView (Application Kit) and an EODisplayGroup. An EOTableViewAssociation can sort the objects in the display group by the left-to-right order of the table columns. The first EOColumnAssociation to be bound to a table view automatically creates the EOTableViewAssociation; you should rarely need to do so yourself.

An EOTableViewAssociation receives data source and delegate messages from the table view, some of which it handles itself, and some of which it forwards to the appropriate EOColumnAssociations. For more information, see the EOColumnAssociation class specification.

Usable With

NSTableView

Aspects

source	Bound to the EODisplayGroup providing objects. This aspect doesn't use a key.
enabled	A boolean attribute of the objects, which determines whether each object's row is editable. Note that because EOColumnAssociation also uses this aspect, you can use it with different keys to limit editability to the whole row or to an individual cell (column) in that row.
textColor	An NSColor attribute of the objects, which determines the color of text for each object's row in the NSTableView.
bold	A boolean attribute of the objects, which determines whether each objects row is displayed in bold or regular weight text.

Aspects

italic	A boolean attribute of the objects, which determines whether each objects row is displayed in italic or normal angle text.

Object Keys Taken

dataSource	An EOTableViewAssociation responds to some data source messages and forwards others to the appropriate EOColumnAssociation.
delegate	An EOTableViewAssociation forwards delegate messages to the appropriate EOColumnAssociations.
target	Reserved, but not used.

Example

For an example of using an EOTableViewAssociation, see the EOColumnAssociation class specification.

Method Types

Setting up a table view association	
	+ bindToTableView:displayGroup:
Sorting	
	– setSortsByColumnOrder:
	– sortsByColumnOrder
Accessing the active EOColumnAssociation	
	– editingAssociation
Table view data source methods	
	– numberOfRowsInTableView:
	– tableView:setObjectValue:forTableColumn:row:
	– tableViewObjectValueForLocationtableView: objectValueForTableColumn:row:
Table view delegate methods	
	– tableView:shouldEditTableColumn:row:
	– tableView:willDisplayCell:forTableColumn:row:
Table view notification methods	
	– tableViewSelectionDidChange:

Control delegate methods
- – controlDidFailToFormatStringErrorDescriptioncontrol:
  didFailToFormatString:errorDescription:
- – control:isValidObject:
- – controlTextShouldBeginEditingcontrol:textShouldBeginEditing:

Class Methods

bindToTableView:displayGroup:

+ (void)**bindToTableView:**(NSTableView *)*aTableView*
  displayGroup:(EODisplayGroup *)*aDisplayGroup*

Creates an EOTableViewAssociation, binding *aTableView* to *aDisplayGroup*, if there isn't already a table view association for *aTableView.*

Instance Methods

editingAssociation

– (EOColumnAssociation *)**editingAssociation**

Returns the EOColumnAssociation for the NSTableView cell being edited, or **nil** if no cell is being edited.

setSortsByColumnOrder:

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

Controls whether the receiver applies a sort ordering to its EODisplayGroup. If *flag* is YES, it builds EOSortOrderings (EOControl) for each of the EOColumnAssociations, collects them into an NSArray based on the left-to-right order of the columns, and assigns them to the display group with **setSortOrderings:**. If *flag* is NO, it doesn't alter the sort ordering of the display group.

An EOTableViewAssociation assigns sort orderings based on the left to right order of the table columns, and reassigns them whenever the user moves a column.

See also: – **sortingSelector** (EOColumnAssociation)

sortsByColumnOrder

– (BOOL)**sortsByColumnOrder**

Returns YES if the receiver assigns EOSortOrderings (EOControl) to its EODisplayGroup based on the sorting selectors of its EOColumnAssociations, NO if it doesn't alter the display group's sort ordering.

Data Source, Delegate, and Notification Methods

control:didFailToFormatString:errorDescription:

– (BOOL)**control:**(NSControl *)*aTableView*
  didFailToFormatString:(NSString *)*aString*
  errorDescription:(NSString *)*errorDescription*

Forwards the message to the receiver's editing association.

See also: – **editingAssociation**

control:isValidObject:

– (BOOL)**control:**(NSControl *)*aTableView*
  isValidObject:(id)*anObject*

Forwards the message to the receiver's editing association.

See also: – **editingAssociation**

control:textShouldBeginEditing:

– (BOOL)**control:**(NSControl *)*aTableView*
  textShouldBeginEditing:(NSText *)*fieldEditor*

Forwards the message to the receiver's editing association.

See also: – **editingAssociation**

numberOfRowsInTableView:

– (int)**numberOfRowsInTableView:**(NSTableView *)*aTableView*

Returns the number of displayed objects in the receiver's EODisplayGroup.

See also: – **displayedObjects** (EODisplayGroup)

tableView:objectValueForTableColumn:row:

– (id)**tableView:**(NSTableView *)*aTableView*
objectValueForTableColumn:(NSTableColumn *)*aTableColumn*
row:(int)*rowIndex*

Forwards the message to *aTableColumn*’s identifier—assumed to be the EOColumnAssociation bound to that column—so that it can provide the value.

tableViewSelectionDidChange:

– (void)**tableViewSelectionDidChange:**(NSNotification *)*aNotification*

Updates the receiver’s EODisplayGroup based on the new selection in the table view.

See also: – **setSelectionIndexes:** (EODisplayGroup)

tableView:setObjectValue:forTableColumn:row:

– (void)**tableView:**(NSTableView *)*aTableView*
setObjectValue:(id)*value*
forTableColumn:(NSTableColumn *)*aTableColumn*
row:(int)*rowIndex*

Forwards the message to *aTableColumn*’s identifier—assumed to be the EOColumnAssociation bound to that column—so that it can set the value.

tableView:shouldEditTableColumn:row:

– (BOOL)**tableView:**(NSTableView *)*aTableView*
shouldEditTableColumn:(NSTableColumn *)*aTableColumn*
row:(int)*rowIndex*

Returns NO if the “enabled” aspect is bound and its value for the object at *rowIndex* is NO. Otherwise forwards the message to *aTableColumn*’s identifier—assumed to be the EOColumnAssociation bound to that column—and returns its response. Note that because the two associations’ **enabled** aspects can be bound to different keys, you can limit editability to the whole row or to an individual cell (column) in that row.

tableView:willDisplayCell:forTableColumn:row:

– (void)**tableView:**(NSTableView *)*aTableView*
    willDisplayCell:(id)*aCell*
    forTableColumn:(NSTableColumn *)*aTableColumn*
    row:(int)*rowIndex*

Alters the display characteristics for *aCell* according to the values for the **enabled**, **textColor**, **bold**, and **italic** aspects of the object at *rowIndex*. Then forwards the message to *aTableColumn*'s identifier—assumed to be the EOColumnAssociation bound to that column—allowing it to adjust *aCell* based on its own **enabled** aspect.

EOTextAssociation

Inherits From:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Conforms To:	NSCoding (EOAssociation) EOObserving (EODelayedObserver) NSObject (NSObject)
Declared In:	EOInterface/EOTextAssociation.h

Class Description

In a Yellow Box application, an EOTextAssociation object displays a plain or rich text attribute in an NSText object (Application Kit) by binding the text object to a string or NSData attribute. It determines the kind of text received from an object by examining the beginning for signature codes specific to RTF and RTFD. When writing text back to the object, the association examines the configuration of the NSText object to determine the type to use according to the following table:

Multiple Fonts	Allows Graphics	Type Written to Object
NO	NO	NSString text
YES	NO	NSData containing RTF
YES	YES	NSData containing RTFD

In a Java Client application, an EOTextAssociation object displays a plain text attribute in an EOTextField, EOTextArea, or EOFormCell by binding the text object to a string. Text is written back to the object as an NSString.

The following tables describe the display objects an EOTextAssociation can be used with, the aspects of an EOTextAssociation, and the object keys it takes.

Usable With
NSText, NSTextView, NSCStringText (Application Kit)
EOTextField, EOTextArea, EOFormCell (Java Client)

Aspects

value	A text attribute of the selected object.
editable (Yellow Box only)	A boolean attribute of the selected object, which determines whether the text object is editable.
enabled (Java Client only)	A boolean attribute of the selected object, which determines whether the text object is enabled.

Object Keys Taken

delegate (Yellow Box only)	An EOTextAssociation accepts delegate messages related to the editing and validation of text; see the NSText, NSTextView, and NSCStringText class specifications for more information.

EOViewLayout

Inherits From:	Object
Implements:	java.awt.LayoutManager2 java.io.Serializable
Package:	com.apple.client.eointerface

Class Description

EOViewLayout is an AWT LayoutManager that implements the geometry options available in InterfaceBuilder's Size inspector. The size of a Component embedded in a Container using this layout will be a function of both its autosizing mask and its initial size (see **setAutosizingMask** for details).

EOViewLayout is for use in Java Client applications only; there isn't an equivalent class for Yellow Box.

Constructors

public **EOViewLayout**()

Any consumers of EOViewLayout should use the **defaultInstance**.

Static Methods

defaultInstance

public static EOViewLayout **defaultInstance**()

Returns that single instance of the receiver used to lay out all InterfaceBuilder-generated Containers.

Instance Methods

setAutosizingMask

public void **setAutosizingMask**(java.awt.Component *component*, int *mask*)

Sets the autosizing mask of *component* to *mask*. This information is subsequently used by the receiver to calculate the new location and dimensions of *component* whenever its parent is resized. The *mask* should be some bitwise combination of the following:

Fixed	neither *component*'s location nor its dimensions may be adjusted
MaxXMargin	the distance between *component*'s right edge and that of its parent may be adjusted
MinXMargin	*component*'s left edge distance may be adjusted
MaxYMargin	the distance between *component*'s bottom edge and that of its parent may be adjusted
MinYMargin	*component*'s top edge distance may be adjusted
WidthSizable	*component*'s width may be adjusted
HeightSizable	*component*'s height may be adjusted
BothSizable	both width and height may be adjusted

Note that unless *mask* is Fixed (the default),*component*'s adjusted size will be some factor of its size at the moment **setAutosizingMask** was invoked.

NSImage Additions

Inherits From:	NSObject
Declared In:	EOInterface/EOControlAssociation.h

Class Description

Enterprise Objects Framework adds one method to NSImage to aid in conversion of image data from databases. This method is used as a factory method for custom value archiving, as described in the EOCustomClassArchiving informal protocol specification. See the NSImage class specification in the Application Kit documentation for a list supported image file formats.

Class Methods

imageWithData:

+ **imageWithData:**(NSData *)*imageData*

Creates an NSImage from *imageData* and returns it.

See also: **– initWithData:** (NSImage class of the Application Kit), **– TIFFRepresentation** (NSImage class of the Application Kit)