EOInterface Framework

API Reference

The EOInterface Framework

Package:	com.apple.client.eocontrol (com.apple.client.eointerface) com.apple.yellow.eointerface (com.apple.yellow.eointerface)

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	com.apple.yellow.eointerface	com.apple.client.eointerface	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
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.

Association	com.apple.yellow.eointerface	com.apple.client.eointerface	Description
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.

Association	com.apple.yellow.eointerface	com.apple.client.eointerface	Description
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.
EOTextAssociation	Yes	Yes	Displays a plain or rich text attribute in an NSText object (Application Kit) or an EOTextField, EOTextArea, or EOFormCell (com.apple.client.eointerface) by binding the text object to a string or NSData attribute.

EOActionAssociation

Inherits from:	(com.apple.client.eointerface) EOAssociation : EODelayedObserver (EOControl) : Object (com.apple.yellow.eointerface) EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver) (com.apple.client.eointerface only) java.awt.event.ActionListener (com.apple.client.eointerface only) NSDisposable (EOAssociation)
Package:	com.apple.client.eointerface com.apple.yellow.eointerface

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

(com.apple.client.eointerface) Any object that implements the method `addActionListener` (javax.swing.JButton and javax.swing.JMenuItem, for example)

(com.apple.yellow.eointerface) NSControl, NSActionCell, and their subclasses

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.

Constructors

EOActionAssociation

```
public EOActionAssociation(Object aDisplayObject)
```

Creates a new EOActionAssociation to monitor and update the value in *`aDisplayObject`*, typically a button or menu item.

You normally set up associations in Interface Builder, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` (EOAssociation), `establishConnection` **(EOAssociation)**

Instance Methods

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent anActionEvent)
```

(com.apple.client.eointerface) 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.

breakConnection

```
public void breakConnection()
```

See the `breakConnection` method description in the superclass EOAssociation.

establishConnection

```
public void establishConnection()
```

See the `establishConnection` method description in the superclass EOAssociation.

isUsableWithObject

```
public boolean isUsableWithObject(Object aDisplayObject)
```

(com.apple.client.eointerface) Returns `true` if *`aDisplayObject`* implements the method `addActionListener`, `false` otherwise.

See Also: `isUsableWithObject` (EOAssociation)

primaryAspect

```
public String primaryAspect()
```

(com.apple.client.eointerface) Returns `EOAssociation.ActionAspect`.

See Also: `primaryAspect` (EOAssociation)

subjectChanged

```
public void subjectChanged()
```

See the `subjectChanged` method description in the superclass EOAssociation.

EOActionCellAssociation

Inherits from:	EOGenericControlAssociation : EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

Class Description

EOActionCellAssociation is the default association class for use with NSActionCells (Application Kit).

Note: This class doesn't exist in the com.apple.client.eointerface package.

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 `controlIsValidObject`, 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.

Constructors

EOActionCellAssociation

`public EOActionCellAssociation(Object aDisplayObject)`

Creates a new EOActionCellAssociation to monitor and update the value in *aDisplayObject*, which is typically an Application Kit NSActionCell.

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` **(EOAssociation),** `establishConnection` **(EOAssociation)**

EOActionInsertionAssociation

Inherits from:	(com.apple.client.eointerface) EOAssociation : EODelayedObserver (EOControl) : Object (com.apple.yellow.eointerface) EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver) (com.apple.client.eointerface only) java.awt.event.ActionListener (com.apple.client.eointerface only) NSDisposable (EOAssociation)
Package:	com.apple.client.eointerface com.apple.yellow.eointerface

Class Description

An EOActionInsertionAssociation object inserts objects from one display group into another.

Usable With

(com.apple.client.eointerface) Any object that implements the method `addActionListener` (javax.swing.JButton and javax.swing.JMenuItem, for example)

(com.apple.yellow.eointerface) Any object that responds to `setAction`, 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.

Constructors

EOActionInsertionAssociation

`public EOActionInsertionAssociation(Object` *`aDisplayObject`*`)`

Creates a new EOActionInsertionAssociation to monitor and update the value in *`aDisplayObject`*.

You normally set up associations in Interface Builder, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` (EOAssociation), `establishConnection` **(EOAssociation)**

Instance Methods

actionPerformed

`public void actionPerformed(java.awt.event.ActionEvent` *`event`*`)`

(com.apple.client.eointerface) 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.

breakConnection

```
public void breakConnection()
```

(com.apple.client.eointerface) See the `breakConnection` method description in the superclass (EOAssociation).

establishConnection

```
public void establishConnection()
```

(com.apple.client.eointerface) See the `establishConnection` method description in the superclass (EOAssociation).

isUsableWithObject

```
public boolean isUsableWithObject(Object aDisplayObject)
```

(com.apple.client.eointerface) Returns `true` if *`aDisplayObject`* implements the method `addActionListener`, `false` otherwise.

See Also: `isUsableWithObject` (EOAssociation)

primaryAspect

```
public String primaryAspect()
```

(com.apple.client.eointerface) Returns `EOAssociation.SourceAspect`.

See Also: `primaryAspect` (EOAssociation)

subjectChanged

```
public void subjectChanged()
```

(com.apple.client.eointerface) See the `subjectChanged` method description in the superclass (EOAssociation).

EOAssociation

Inherits from:	(com.apple.client.eointerface) EODelayedObserver (EOControl) : Object (com.apple.yellow.eointerface) EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver) (com.apple.client.eointerface only) NSDisposable
Package:	com.apple.client.eointerface com.apple.yellow.eointerface

Class at a Glance

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.

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)

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. For information on the different EOAssociation subclasses you can use, see the following subclass specifications:

com.apple.client.eointerface Associations

EOActionAssociation	EOActionInsertionAssociation
EOComboBoxAssociation	EOMasterDetailAssociation
EOTableAssociation	EOTableColumnAssociation
EOTableViewAssociation	EOTextAssociation

com.apple.yellow.eointerface Associations

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

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" (page 39)
- "Setting up an EOAssociation Programmatically" (page 41)
- "Creating a Subclass of EOAssociation" (page 42)

Constants

(com.apple.client.eointerface only) EOAssociation defines the following String constants to identify the names of association aspects:

`ActionAspect`	`EnabledAspect`	`SourceAspect`
`ArgumentAspect`	`ParentAspect`	`TitlesAspect`

BoldAspect	SelectedObjectAspect	ValueAspect
DestinationAspect	SelectedTitleAspect	URLAspect
ItalicAspect		

(com.apple.client.eointerface only) The class defines additional String constants to identify association signatures (see the method description `aspectSignatures` for more information):

AttributeAspectSignature	NullAspectSignature
AttributeToOneAspectSignature	ToOneAspectSignature
AttributeToOneToManyAspectSignature	ToOneToManyAspectSignature
AttributeToManyAspectSignature	ToManyAttributeSignature

Interfaces Implemented

NSDisposable

`dispose`

Method Types

Declaring capabilities

`aspects` (com.apple.yellow.eointerface static method)

`aspects` (com.apple.client.eointerface instance method)

`aspectSignatures` (com.apple.yellow.eointerface static method)

`aspectSignatures` (com.apple.client.eointerface instance method)

`objectKeysTaken` (com.apple.yellow.eointerface)

`isUsableWithObject` (com.apple.yellow.eointerface static method)

`isUsableWithObject` (com.apple.client.eointerface instance method) `associationClassesSuperseded` (com.apple.yellow.eointerface static method)

`associationClassesSuperseded` (com.apple.client.eointerface instance method)

`displayName` (com.apple.yellow.eointerface static method)

`displayName` (com.apple.client.eointerface instance method)

`primaryAspect` (com.apple.yellow.eointerface static method)

`primaryAspect` (com.apple.client.eointerface instance method)

`canBindAspect`

Getting all possible EOAssociations for a display object

`associationClassesForObject:`

Getting the display object

`object`

Examining bindings

`displayGroupForAspect`

`displayGroupKeyForAspect`

Updating values

`subjectChanged`

`endEditing`

Accessing enterprise object values

`setValueForAspect`

`setValueForAspectAtIndex`

`valueForAspect`

`valueForAspectAtIndex`

Handling validation errors

`shouldEndEditing`

`shouldEndEditingAtIndex`

Constructors

EOAssociation

```
public EOAssociation(Object aDisplayObject)
```

Never use this method to create an EOAssociation. EOAssociation is conceptually an abstract class, and you'd never use instances of it. Instead, use subclasses of EOAssociation. Instances of the subclasses can be created programmatically with a constructor of this same form. For more information, see the constructor description for the subclass you want to use. For a list of the subclasses, see the "Class Description" (page 22).

Static Methods

aspects

```
public static NSArray aspects()
```

(com.apple.yellow.eointerface only) 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.

See Also: `aspects` **instance method**

aspectSignatures

```
public static NSArray aspectSignatures()
```

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

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

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

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

See Also: `aspectSignatures` **instance method**

associationClassesForObject:

```
public static NSArray associationClassesForObject(Object aDisplayObject)
```

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

associationClassesSuperseded

```
public static NSArray associationClassesSuperseded()
```

(com.apple.yellow.eointerface only) 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 EOPopUpAssociation is always more appropriate to use with pop-up buttons.

See Also: `associationClassesSuperseded` **instance method**

displayName

```
public static String displayName()
```

(com.apple.yellow.eointerface only) 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.

See Also: `displayName` **instance method**

isUsableWithObject

```
public static boolean isUsableWithObject(Object aDisplayObject)
```

(com.apple.yellow.eointerface only) Overridden by subclasses to return `true` if instances of the receiving class are usable with *`aDisplayObject`*, `false` 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.

See Also: `isUsableWithObject` **instance method**

objectKeysTaken

```
public static NSArray objectKeysTaken()
```

(com.apple.yellow.eointerface only) 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.

See Also: `objectKeysTaken` **instance method**

primaryAspect

```
public static String primaryAspect()
```

(com.apple.yellow.eointerface only) Overridden by subclasses to return the default aspect, usually one denoting the displayed value, which by convention is named "value". EOAssociation's implementation returns `null`.

See Also: `primaryAspect` **instance method**

Instance Methods

aspects

```
public NSArray aspects()
```

(com.apple.client.eointerface only) Overridden by subclasses to return the names of the receiving class's aspects, as string objects. Subclasses should include their superclass's aspects and add their own when overriding this method.

See Also: `aspects` **static method**

aspectSignatures

```
public NSArray aspectSignatures()
```

(com.apple.client.eointerface only) Overridden by subclasses to return the signatures of the receiver's aspects, an array of string objects matching its aspects array index for index. The signature strings can be any of:

Constant	The Aspect Can Be Bound to
`AttributeAspectSignature`	Attributes
`AttributeToOneAspectSignature`	Attributes and to-one relationships
`AttributeToManyAspectSignature`	Attributes and to-many relationships
`AttributeToOneToManyAspectSignature`	Attributes, to-one relationships, and to-many relationships

Constant	The Aspect Can Be Bound to
`ToOneAspectSignature`	To-one relationships
`ToOneToManyAspectSignature`	To-one and to-many relationships
`ToManyAttributeSignature`	To-many relationships
`NullAspectSignature`	An EODisplayGroup without a key (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 `AttributeToOneToManyAspectSignature` strings.

See Also: `aspectSignatures` **static method**

associationClassesSuperseded

```
public NSArray associationClassesSuperseded()
```

(com.apple.client.eointerface only) 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 EOPopUpAssociation is always more appropriate to use with pop-up buttons.

See Also: `associationClassesSuperseded` **static method**

bindAspect

```
public void bindAspect(
   String aspectName,
   EODisplayGroup aDisplayGroup,
   String 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" (page 41) in the class description for more information.

See Also: `establishConnection`

breakConnection

```
public void breakConnection()
```

Removes the receiver from its EODisplayGroup and display object. Subclasses should override this method to remove the receiver from any outlets of the display object and invoke `super`'s implementation at the end.

See Also: `establishConnection`

canBindAspect

```
public boolean canBindAspect(
   String aspectName,
   EODisplayGroup aDisplayGroup,
   String key)
```

Overridden by subclasses to return `true` if the receiver can tie an aspect named *aspectName* from its display object to the property identified by *key* in *aDisplayGroup*, `false` 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 `true`.

See Also: `localKeys` (EODisplayGroup), `attributeKeys` (EOClassDescription), `toOneRelationshipKeys` (EOClassDescription), `toManyRelationshipKeys` **(EOClassDescription)**

copyMatchingBindingsFromAssociation

```
public void copyMatchingBindingsFromAssociation(EOAssociation anAssociation)
```

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

displayGroupForAspect

```
public EODisplayGroup displayGroupForAspect(String aspectName)
```

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

See Also: `displayGroupKeyForAspect`

displayGroupKeyForAspect

```
public String displayGroupKeyForAspect(String aspectName)
```

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

See Also: `displayGroupForAspect`

displayName

```
public String displayName()
```

(com.apple.client.eointerface only) Returns the name used for display purposes. EOAssociation's implementation simply returns the name of the receiver's class.

See Also: `displayName` **static method**

endEditing

`public boolean endEditing()`

Overridden by subclasses to pass the value of the receiver's display object to the EODisplayGroup, by invoking `setValueForAspect` with the display object's value and the appropriate aspect (typically "value"). Returns `true` if successful, `false` if not—specifically if `setValueForAspect` returns `false`. 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 `true`.

establishConnection

`public 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. Subclasses should invoke `super`'s implementation after establishing their own connections.

See "Setting up an EOAssociation Programmatically" (page 41) in the class description for more information.

See Also: `breakConnection`

"Setting up an EOAssociation Programmatically" **(page 41)**

isEnabled

`protected boolean isEnabled()`

(com.apple.client.eointerface only) Returns `false` if the receiver has explicitly disabled its display object or if the receiver's `EnabledAspect` (if bound) resolves to `false`; `true` otherwise.

isEnabledAtIndex

```
protected boolean isEnabledAtIndex(int index)
```

(com.apple.client.eointerface only) Returns `false` if the receiver has explicitly disabled its display object or if the receiver's `EnabledAspect` (if bound) resolves to `false` for index; `true` otherwise.

isExplicitlyDisabled

```
public boolean isExplicitlyDisabled()
```

(com.apple.client.eointerface only) Returns `true` if the receiver has explicitly disabled its display object, `false` otherwise.

isUsableWithObject

```
public boolean isUsableWithObject(Object aDisplayObject)
```

(com.apple.client.eointerface only) Overridden by subclasses to return `true` if instances of the receiving class are usable with *`aDisplayObject`*, `false` if they aren't. The receiving class can examine any relevant characteristic of *`aDisplayObject`*—its class, configuration, and so on. EOAssociation's implementation returns `false`.

See Also: `isUsableWithObject` **static method**

object

```
public Object object()
```

Returns the receiver's display object.

See Also:

objectKeysTaken

`public NSArray objectKeysTaken()`

(com.apple.client.eointerface only) Overridden by subclasses to return the names of display object outlets that instances assume control of. Interface Builder uses this information to disable connections from these outlets in its Connections Inspector.

See Also: `objectKeysTaken` **static method**

primaryAspect

`public String primaryAspect()`

(com.apple.client.eointerface only) Overridden by subclasses to return the default aspect, usually one denoting the displayed value, which by convention is named "value". EOAssociation's implementation returns `null`.

See Also: `primaryAspect` **static method**

priority

`public int priority()`

Returns the receiver's change notification priority. For more information, see the EODelayedObserver class specification (EOControl).

setAutoCreated

`public void setAutoCreated(boolean aBoolean)`

(com.apple.client.eointerface only) This method is provided for internal use and is intentionally undocumented. You should never need to invoke or customize this method.

setExplicitlyDisabled

```
public void setExplicitlyDisabled(boolean flag)
```

(com.apple.client.eointerface only) Sets according to *flag* whether or not the association is explicitly disabled. This method and its counterpart `isExplicitlyDisabled` are used by objects in the com.apple.client.eoapplication and com.apple.client.eogeneration packages for Direct to Java Client applications. An association is "explicitly disabled" when the display object shouldn't be editable, such as in the case where the display object simply displays the results of a search.

setValueForAspect

```
public boolean setValueForAspect(
   Object value,
   String 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` message with *value* and the key as arguments. Returns `true` if successful, or if there's no display group bound to *aspectName*. Returns `false` if there's an display group and it doesn't accept the new value.

See Also: `valueForAspect`

setValueForAspectAtIndex

```
public boolean setValueForAspectAtIndex(
   Object value,
   String aspectName,
   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 `setValueForObjectAtIndex` message with *value*, *index*, and the key as arguments. Returns `true` if successful, or if there's no display group bound to *aspectName*. Returns `false` if there's a display group and it doesn't accept the new value.

See Also: `valueForAspectAtIndex`

shouldEndEditing

```
public boolean shouldEndEditing(
   String aspectName,
   String inputString,
   String errorDescription)
```

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

For example, an association bound to an NSControl object (Application Kit) receives a controlDidFailToFormatStringErrorDescription delegate message when the control's formatter fails to format the input string. Its implementation of that method invokes shouldEndEditing.

See Also: shouldEndEditingAtIndex

shouldEndEditingAtIndex

```
public boolean shouldEndEditingAtIndex(
   String aspectName,
   String inputString,
   String errorDescription,
   int index)
```

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

subjectChanged

```
public 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

```
public Object valueForAspect(String 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 `null` if there's no display group or key bound to *aspectName*.

See Also: `setValueForAspect`

valueForAspectAtIndex

```
public Object valueForAspectAtIndex(
   String aspectName,
   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` message with *index* and the key. Returns `null` if there's no display group or key bound to *aspectName*.

See Also: `setValueForAspectAtIndex`

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 com.apple.yellow.eointerface package, 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 com.apple.yellow.eointerface package, 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 `setValueForAspect`. This process and the querying of display groups are described under "Monitoring Changes from the Display Object" (page 45).

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.

```
JTextComponent salaryText;
EODisplayGroup employeeGroup;
EOTextAssociation association;

association = new EOTextAssociation(salaryText);
association.bindAspect(EOTextAssociation.ValueAspect, employeeGroup, "salary");
association.bindAspect(
    EOTextAssociation.EnabledAspect, employeeGroup, "eligibleForRaise");
association.establishConnection();
```

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` 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 when using the com.apple.yellow.eointerface APIs 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 com.apple.yellow.eointerface.EOAssociation 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:

Aspects (required)

Your EOAssociation subclass must define an `aspects` class method that returns an NSArray of aspect names, as Strings. Some standard aspects are:*value*, the value of an attribute or relationship; *enabled*, whether the control should be enabled; *titles*, all existing values for an attribute; and *selectedTitle*, the value of the selected attribute (bound to the same key as "titles").

What the subclass works with (required)

Interface Builder asks each EOAssociation subclass if it can work with a given object when it displays its Connections Inspector. Your subclass should implement the `isUsableWithObject` class method to examine the object provided and return `true` 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 Strings 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 String or Number), "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.

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` 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 constructor is

```
public EOAssociation(Object object)
```

but you rarely need to write custom initialization code in this method. Instead, you override `establishConnection`, which is where the real initialization takes place, as described above in "Setting up an EOAssociation Programmatically" (page 41).

Your subclass's implementation of `establishConnection` 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 `setValueForAspect` with the display object's new contents. Complex associations might set enterprise object values more directly via EODisplayGroup's `setSelectedObjectValue`, `setValueForObject`, or `setValueForObjectAtIndex` 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 `false` 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 `controlIsValidObject` and `controlDidFailToFormatStringErrorDescription` 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 `controlIsValidObject` should simply try to save the new value, using EOAssociation's setValueForAspect or setValueForAspectAtIndex, returning true or false as that message does. For controlDidFailToFormatStringErrorDescription, the typical response should be to invoke shouldEndEditing or shouldEndEditingAtIndex.

EOColumnAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

Class Description

An EOColumnAssociation object cooperates with an EOTableViewAssociation to display values in a column of an NSTableView (Application Kit).

Note: This class doesn't exist in the com.apple.client.eointerface package.

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.

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

EOColumnAssociations provide values using NSTableView's DataSource methods `tableViewSetObjectValueForLocation` and `tableViewObjectValueForLocation`. 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 identifier 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 corresponding row.

Method Types

Sorting rows

`setSortingSelector`

`sortingSelector`

Table view data source methods

`tableViewSetObjectValueForLocation`

`tableViewObjectValueForLocation`

Table view delegate methods

`tableViewShouldEditLocation`

`tableViewWillDisplayCell`

Control delegate methods

`controlDidFailToFormatStringErrorDescription`

`controlIsValidObject`

`controlTextShouldBeginEditing`

Constructors

EOColumnAssociation

```
public EOColumnAssociation(Object aDisplayObject)
```

Creates a new EOColumnAssociation to monitor and update the row values in *aDisplayObject*, an NSTableColumn (Application Kit).

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` (EOAssociation), `establishConnection` **(EOAssociation)**

Instance Methods

controlDidFailToFormatStringErrorDescription

```
public boolean controlDidFailToFormatStringErrorDescription(
   com.apple.yellow.application.NSControl aTableView,
   String aString,
   String errorDescription)
```

Invokes `shouldEndEditing` (defined by EOAssociation) and returns the result.

controlIsValidObject

```
public boolean controlIsValidObject(
   com.apple.yellow.application.NSControl aControl,
   Object anObject)
```

Saves the value of any cell being edited using `setValueForAspect`, and if successful sends an `associationDidEndEditing` message to the receiver's EODisplayGroup. Returns `true` if successful (or if no changes need be saved), `false` if unsuccessful.

controlTextShouldBeginEditing

```
public boolean controlTextShouldBeginEditing(
   com.apple.yellow.application.NSControl aControl,
   com.apple.yellow.application.NSText text)
```

Sends an `associationDidBeginEditing` message to the receiver's EODisplayGroup and returns `true`.

establishConnection

`public void establishConnection()`

Attaches the receiver to the outlets of its display object and otherwise configures the display object (such as by setting its action method). See "Setting up an EOAssociation Programmatically" (page 41) in the class description for more information.

See Also: `breakConnection` (EOAssociation)

setSortingSelector

`public void setSortingSelector(NSSelector aSelector)`

Sets the method selector used to sort rows to *aSelector*, one of (defined in EOControl):

- EOSortOrdering.CompareAscending
- EOSortOrdering.CompareDescending
- EOSortOrdering.CompareCaseInsensitiveAscending
- EOSortOrdering.CompareCaseInsensitiveDescending
- `null` (to tell the receiver not to sort)

For more information on these selectors, see the section "Comparison Methods" in the EOSortOrdering class specification (EOControl).

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 CompareAscending.

sortingSelector

`public NSSelector sortingSelector()`

Returns the method selector used to sort rows, or `null` if the column isn't sorted.

tableViewObjectValueForLocation

```
public Object tableViewObjectValueForLocation(
   com.apple.yellow.application.NSTableView tableView,
   com.apple.yellow.application.NSTableColumn tableColumn,
   int rowIndex)
```

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

tableViewSetObjectValueForLocation

```
public void tableViewSetObjectValueForLocation(
   com.apple.yellow.application.NSTableView tableView,
   Object value,
   com.apple.yellow.application.NSTableColumn tableColumn,
   int rowIndex)
```

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

tableViewShouldEditLocation

```
public boolean tableViewShouldEditLocation(
   com.apple.yellow.application.NSTableView tableView,
   com.apple.yellow.application.NSTableColumn tableColumn,
   int rowIndex)
```

Returns `false` if the `enabled` aspect is bound and its value for the object at *rowIndex* is `false`. Otherwise returns `true`. 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.

tableViewWillDisplayCell

```
public void tableViewWillDisplayCell(
   com.apple.yellow.application.NSTableView tableView,
   Object aCell,
   com.apple.yellow.application.NSTableColumn tableColumn,
   int rowIndex)
```

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

EOColumnEditor

Inherits from:	Object
Implements:	javax.swing.table.TableCellEditor javax.swing.CellEditor (javax.swing.table.TableCellEditor)
Package:	com.apple.client.eointerface

Class Description

EOColumnEditor is an abstract class that implements generalized cell editing management for javax.swing.JTables. Swing specifies that JTable cell editing is performed by an object implementing the javax.swing.table.TableCellEditor interface. EOColumnEditor implements this interface in a generalized way, and concrete subclasses such as EOTextColumnEditor perform component-specific instantiation and event communication.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

The most important function of an EOColumnEditor instance is mediating between its Component and the EOTableColumnAssociation that's bound to the edited column. This mediation enables the validation of edited values that associations are required to perform.

Create a subclass of EOColumnEditor if you want to use a Component for JTable editing for which no EOColumnEditor is implemented.

Interfaces Implemented

javax.swing.table.TableCellEditor

`addCellEditorListener` (javax.swing.CellEditor)

`cancelCellEditing` (javax.swing.CellEditor)

`getCellEditorValue`

`getTableCellEditorComponent`

`isCellEditable`

`removeCellEditorListener` (javax.swing.CellEditor)

`shouldSelectCell`

`stopCellEditing`

Method Types

Instantiation

`createEditorComponent`

`editingTableColumnAssociation`

`editorComponent`

`setCellEditorValue`

`setEditorComponent`

Event handling

`beginEditing`

`endEditing`

Instance Methods

beginEditing

```
protected void beginEditing()
```

Invoked from `shouldSelectCell` and `getTableCellEditorComponent` to inform the receiver that editing has been requested and should begin (`shouldSelectCell` is invoked only by mouse clicks). EOColumnEditor's implementation sends `associationDidBeginEditing` to the EODisplayGroup of the EOTableColumnAssociation that's bound to the receiver's TableColumn; so subclasses should invoke `super`'s implementation before activating their Component.

createEditorComponent

```
protected abstract java.awt.Component createEditorComponent()
```

Creates and returns a Component to perform the editing—a JTextField or JComboBox, for example. Invoked in EOColumnEditor's constructor, this method must be overridden by every subclass in order to create and return the Component it manages.

editingTableColumnAssociation

```
protected com.apple.client.eointerface.EOTableColumnAssociation
   editingTableColumnAssociation()
```

Returns the EOTableColumnAssociation that's bound to the column being edited, which is cached in EOColumnEditor's implementation of `shouldSelectCell` and `getTableCellEditorComponent`.

editorComponent

```
public java.awt.Component editorComponent()
```

Returns the receiver's Component—a user interface control that implements the editing mechanism. EOColumnEditor caches the Component in the constructor (in the method `createEditorComponent`, which is invoked from the constructor).

endEditing

```
protected void endEditing()
```

Invoked from `cancelCellEditing` and `stopCellEditing` to inform the receiver that it should end editing. EOColumnEditor's implementation sends `associationDidEndEditing` to the EODisplayGroup of the EOTableColumnAssociation that's bound to the receiver's TableColumn. Subclasses should invoke `super`'s implementation after deactivating their Component.

getCellEditorValue

```
public Object getCellEditorValue()
```

Returns the receiver's `editorComponent`. EOColumnEditor's implementation simply returns `null`, so subclasses must override this method.

isCellEditable

```
public boolean isCellEditable(java.util.EventObject event)
```

Returns `true` if *`event`* is an event that should trigger editing, `false` otherwise. EOColumnEditor's implementation simply returns `false`. Subclasses must override this method.

setCellEditorValue

```
protected abstract void setCellEditorValue(Object initialValue)
```

Invoked from `getTableCellEditorComponent` to assign *`initialValue`* as the receiver's `editorComponent`. Subclasses must override this method.

setEditorComponent

```
public void setEditorComponent(java.awt.Component editorComponent)
```

Sets the receiver's editor component to *`editorComponent`*. Invoked by the constructor, where *`editorComponent`* is the Component returned from `createEditorComponent`.

shouldSelectCell

```
public boolean shouldSelectCell(java.util.EventObject event)
```

Returns `true` if event represents a legitimate selection trigger, or `false` otherwise. EOColumnEditor's implementation invokes `beginEditing` and returns `true`.

stopCellEditing

```
public boolean stopCellEditing()
```

Informs the receiver that it should stop editing. EOColumnEditor's implementation invokes `endEditing` and returns `true`.

Note: Validation failures aren't handled with this method. The boolean return value is ignored.

EOComboBoxAssociation

Inherits from:	(com.apple.client.eointerface) EOAssociation : EODelayedObserver (EOControl) : Object (com.apple.yellow.eointerface) EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver) (com.apple.client.eointerface only) java.awt.event.ActionListener (com.apple.client.eointerface only) NSDisposable (EOAssociation)
Package:	com.apple.client.eointerface com.apple.yellow.eointerface

Class Description

An EOComboBoxAssociation object displays an attribute or to-one relationship value in an NSComboBox (Application Kit), or javax.swing.JComboBox. The items in the combo box can be entered manually, or for a relationship, constructed dynamically from values supplied by an EODisplayGroup. EOComboBoxAssociation is very similar to the EOPopUpAssociation (com.apple.yellow.eointerface only).

Usable With

(com.apple.client.eointerface) javax.swing.JComboBox

(com.apple.yellow.eointerface) NSComboBox (Application Kit)

Aspects

titles	Property of the enterprise objects in an EODisplayGroup that supplies the titles for the items in the combo box list.
selectedTitle	String property of the enterprise object supplying the title to display in the combo 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	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	A boolean attribute of the selected object that determines whether the combo box is enabled.

Object Keys Taken

(com.apple.yellow.eointerface only) 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.
(com.apple.yellow.eointerface only) dataSource	When the NSComboBox requests values for its list, the EOComboBoxAssociation provides them by querying the appropriate EODisplayGroup or groups.
(com.apple.yellow.eointerface only) 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.

Constructors

EOComboBoxAssociation

`public EOComboBoxAssociation(Object` ***`aDisplayObject`***`)`

Creates a new EOComboBoxAssociation to monitor and update the values in *`aDisplayObject`*, a combo box (using the com.apple.yellow.eointerface APIs, it's a com.apple.yellow.application.NSComboBox; using com.apple.client.eointerface APIs, it's a javax.swing.JComboBox).

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` (EOAssociation), `establishConnection` **(EOAssociation)**

Instance Methods

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent anActionEvent)
```

(com.apple.client.eointerface only) 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.

breakConnection

```
public void breakConnection()
```

(com.apple.client.eointerface) Causes the association to remove itself from the list of listeners of the JComboBox, and then calls `super`.

(com.apple.yellow.eointerface) See the `breakConnection` method description in the superclass (EOAssociation).

endEditing

```
public boolean endEditing()
```

See the `endEditing` method description in the superclass (EOAssociation).

establishConnection

```
public void establishConnection()
```

(com.apple.client.eointerface) Causes the association to add itself as a listener of the JComboBox.

(com.apple.yellow.eointerface) See the `establishConnection` method description in the superclass (EOAssociation).

isUsableWithObject

```
public boolean isUsableWithObject(Object aDisplayObject)
```

See the `isUsableWithObject` method description in the superclass (EOAssociation).

primaryAspect

```
public String primaryAspect()
```

See the `primaryAspect` method description in the superclass (EOAssociation).

subjectChanged

```
public void subjectChanged()
```

See the `subjectChanged` method description in the superclass (EOAssociation).

EOControlActionAdapter

Inherits from:	Object
Implements:	java.awt.event.ActionListener NSDisposable
Package:	com.apple.client.eointerface

Class Description

The EOControlActionAdapter class is used to connect user interface controls to the objects that respond to actions performed on those controls. They are usually generated automatically to represent connections made in Interface Builder. For example, suppose you control-drag a connection from a "Fetch" button to a display group and that you connect the button to the display group's `fetch` method. At runtime, an EOControlActionAdapter object is used to invoke the display group's `fetch` method when a user clicks the Fetch button. In this example, the display group is the EOControlActionAdapter's **target**, "fetch" is the name of the **action** (method) to perform on the target, and the button is the **listenee**. An EOControlActionAdapter listens for the listenee (the button) to be acted upon (to be pushed). When the listenee is acted upon, the EOControlActionAdapter performs the action on its target (invokes the display group's `fetch` method).

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

Interfaces Implemented

NSDisposable

```
   dispose
```

Constructors

EOControlActionAdapter

```
public EOControlActionAdapter(
   Object target,
   String actionName,
   Object listenee)

public EOControlActionAdapter(
   String actionName,
   Object listenee)
```

Creates and returns a new EOControlActionAdapter object that performs the method identified by `actionName` on `target` when `listenee` is acted upon. Raises and `llegalStateException` if `listenee` is `null`.

Instance Methods

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent event)
```

Performs the receiver's action on its target. If target is `null`, this method simply returns. If the target doesn't implement the action method, this method prints an error message and returns.

setTarget

```
public void setTarget(Object target)
```

Sets the receiver's target to *target*.

EOControlAssociation

Inherits from:	EOGenericControlAssociation : EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

Class Description

EOControlAssociation is the default EOAssociation subclass for use with NSControl objects (Application Kit).

Note: This class doesn't exist in the com.apple.client.eointerface package.

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 `controlTextShouldBeginEditing` and `controlDidFailToFormatStringErrorDescription`.

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.

Constructors

EOControlAssociation

`public EOControlAssociation(Object aDisplayObject)`

Creates a new EOControlAssociation to monitor and update the row values in *aDisplayObject*, an NSControl object (Application Kit).

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` **(EOAssociation),** `establishConnection` **(EOAssociation)**

EODetailSelectionAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

Class Description

An EODetailSelectionAssociation binds two EODisplayGroups together through a relationship, so that the destination display group acts as an editor for that relationship.

Note: This class doesn't exist in the com.apple.client.eointerface package.

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.

Constructors

EODetailSelectionAssociation

```
public EODetailSelectionAssociation(Object aDisplayObject)
```

Creates a new EODetailSelectionAssociation to monitor and update the value in *aDisplayObject*, an EODisplayGroup.

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` **(EOAssociation),** `establishConnection` **(EOAssociation)**

EODisplayGroup

Inherits from:	(com.apple.client.eointerface) Object (com.apple.yellow.eointerface) NSObject
Implements:	(com.apple.client.eointerface only) NSDisposable (com.apple.client.eointerface only) NSInlineObservable
Package:	com.apple.client.eointerface com.apple.yellow.eointerface

Class at a Glance

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

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.
`insertNewObjectAtIndex`	Creates a new object and inserts it into the EODataSource.

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 EOEditingContext.Editor and EOEditingContext.MessageHandler interface specifications (both interfaces are defined in EOControl). EODisplayGroup implements all the methods of these informal protocols; see their specifications 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:

```
EODistributedDataSource dataSource;    /* Assume this exists. */
EODisplayGroup displayGroup;

displayGroup = new EODisplayGroup();
displayGroup.setDataSource(dataSource);
```

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 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 `displayGroupDisplayArrayForObjects`, 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 EOQualifier's `filteredArrayUsingQualifier` and EOSortOrdering's `sortedArrayUsingKeyOrderArray` methods, 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 `displayGroupDisplayArrayForObjects`, 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 `true` to permit the action or `false` to deny it. For more information, see each method's description in the EODisplayGroup.Delegate interface 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` returns the value of a named property for the object at a given index, and `setValueForObjectAtIndex` sets it. Similarly, `valueForObject` and `setValueForObject`access the objects by object identifier. EOAssociations can also get and set values for the first object in the selection using `selectedObjectValueForKey` and `setSelectedObjectValue`.

Constants

(com.apple.client.eointerface only) EODisplayGroup defines String constants for the names of the notifications it posts. For more information, see "Notifications" (page 110).

Interfaces Implemented

NSInlineObservable (com.apple.client.eointerface only)

`observerData`

`setObserverData`

NSDisposable (com.apple.client.eointerface only)

`dispose`

Method Types

Configuring behavior

`defaultStringMatchFormat`

`defaultStringMatchOperator`

`fetchesOnLoad`

queryBindingValues

queryOperatorValues

selectsFirstObjectAfterFetch

setDefaultStringMatchFormat

setDefaultStringMatchOperator

setFetchesOnLoad

setQueryBindingValues

setQueryOperatorValues

setSelectedObject

setSelectedObjects

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`

`qualifyDataSource`

`enterQueryMode`

`inQueryMode`

`setInQueryMode`

`enabledToSetSelectedObjectValueForKey`

Fetching objects from the data source

`fetch`

Getting the objects

`allObjects`

`displayedObjects`

Updating display of values

`redisplay`

`updateDisplayedObjects`

Setting the objects

`setObjectArray`

Changing the selection

`setSelectionIndexes`

`selectObjectsIdenticalTo`

`selectObject`

`clearSelection`

`selectNext`

`selectPrevious`

Examining the selection

`selectionIndexes`

`selectedObject`

`selectedObjects`

Adding keys

`setLocalKeys`

`localKeys`

Getting the associations

`observingAssociations`

Setting the delegate

`setDelegate`

`delegate`

Changing values from associations

`setSelectedObjectValue`

`selectedObjectValueForKey`

`setValueForObject`

`valueForObject`

`setValueForObjectAtIndex`

`valueForObjectAtIndex`

Editing by associations

`associationDidBeginEditing`

`associationFailedToValidateValue`

`associationDidEndEditing`

`editingAssociation`

`endEditing`

Querying changes for associations

`contentsChanged`

`selectionChanged`

`updatedObjectIndex`

Interacting with the EOEditingContext

`editorHasChangesForEditingContext`

`editingContextWillSaveChanges`

`editingContextPresentErrorMessage`

Constructors

EODisplayGroup

```
public EODisplayGroup()
```

Creates a new EODisplayGroup. The new display group needs to have an EODataSource set with `setDataSource`.

See Also: `bindAspect` **(EOAssociation)**

Static Methods

globalDefaultForValidatesChangesImmediately

```
public static boolean globalDefaultForValidatesChangesImmediately()
```

Returns `true` if the default behavior for new display group instances is to immediately handle validation errors, or `false` if the default behavior leaves errors for the EOEditingContext to handle when saving changes.

See Also: `validatesChangesImmediately`

globalDefaultStringMatchFormat

```
public static String globalDefaultStringMatchFormat()
```

Returns the default string match format string used by display group instances.

See Also: `defaultStringMatchFormat`

globalDefaultStringMatchOperator

```
public static String globalDefaultStringMatchOperator()
```

Returns the default string match operator used by display group instances.

See Also: `defaultStringMatchOperator`

setGlobalDefaultForValidatesChangesImmediately

```
public static void setGlobalDefaultForValidatesChangesImmediately(boolean flag)
```

Sets the default behavior display group instances use when they encounter a validation error. If *`flag`* is `true`, the default behavior is for display groups to immediately present an attention panel indicating a validation error. If *`flag`* is `false`, the default behavior if for display groups to leave validation errors to be handled when changes are saved. By default, display groups don't validate changes immediately.

See Also: – `saveChanges` (EOEditingContext), `setValidatesChangesImmediately`

setGlobalDefaultStringMatchFormat

```
public static void setGlobalDefaultStringMatchFormat(String format)
```

Sets the default string match format to be used by display group instances. The default format string for pattern matching is "`%@*`".

See Also: `setDefaultStringMatchFormat`

setGlobalDefaultStringMatchOperator

`public static void setGlobalDefaultStringMatchOperator(String op)`

Sets the default string match operator to be used by display group instances. The default operator is case insensitive like.

See Also: `setDefaultStringMatchOperator`

Instance Methods

allObjects

`public NSArray allObjects()`

Returns all of the objects collected by the receiver.

See Also: `displayedObjects`, `fetch`

associationDidBeginEditing

`public 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`, `associationFailedToValidateValue`

associationDidEndEditing

`public 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`, `associationFailedToValidateValue`

associationFailedToValidateValue

```
public boolean associationFailedToValidateValue(
   EOAssociation anAssociation,
   String value,
   String key,
   Object anObject,
   String errorDescription)
```

Invoked by *anAssociation* from its `shouldEndEditingAtIndex` method to let the receiver handle a validation error. This method opens an attention panel with *errorDescription* as the message and returns `false`.

See Also: `displayGroupShouldDisplayAlert` (EODisplayGroup.Delegate)

awakeFromNib

```
public void awakeFromNib()
```

(com.apple.client.eointerface only) Invoked when the receiver is unarchived from a nib file to prepare it for use in an application. You should never invoke this method directly. Finishes initializing the receiver and updates the display.

See Also: `finishInitialization`, `redisplay`

clearSelection

```
public boolean clearSelection()
```

Invokes `setSelectionIndexes` to clear the selection, returning `true` on success and `false` on failure.

contentsChanged

```
public boolean contentsChanged()
```

Returns `true` if the receiver's array of objects has changed and not all observers have been notified, `false` otherwise. EOAssociations use this in their `subjectChanged` methods to determine what they need to update.

See Also: `selectionChanged`, `updatedObjectIndex`

dataSource

```
public EODataSource dataSource()
```

Returns the receiver's EODataSource.

See Also: `setDataSource`

defaultStringMatchFormat

```
public String defaultStringMatchFormat()
```

Returns the format string that specifies how pattern matching will be performed on string values in the query dictionaries (`equalToQueryValues`, `greaterThanQueryValues`, and `lessThanQueryValues`). 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

```
public String defaultStringMatchOperator()
```

Returns the operator used to perform pattern matching for string values in the query dictionaries (`equalToQueryValues`, `greaterThanQueryValues`, and `lessThanQueryValues`). If a key in one of the query dictionaries 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

```
public Object delegate()
```

Returns the receiver's delegate.

See Also: `setDelegate`

delete

`public void delete(Object anObject)`

(com.apple.client.eointerface only) This action method invokes `deleteSelection`.

deleteObjectAtIndex

`public boolean deleteObjectAtIndex(int index)`

Attempts to delete the object at *`index`*, returning `true` if successful and `false` if not. Checks with the delegate using `displayGroupShouldDeleteObject`. If the delegate returns `false`, this method fails and returns `false`. If successful, sends the delegate a `displayGroupDidDeleteObject` message.

This method performs the delete by sending `deleteObject` to the EODataSource. If that message throws an exception, this method fails and returns `false`.

deleteSelection

`public boolean deleteSelection()`

Attempts to delete the selected objects, returning `true` if successful and `false` if not.

displayedObjects

`public 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`, `displayGroupDisplayArrayForObjects` (EODisplayGroup.Delegate), `qualifier`, `sortOrderings`

editingAssociation

`public EOAssociation editingAssociation()`

Returns the EOAssociation editing a value if there is one, `false` if there isn't.

See Also: `associationDidBeginEditing`, `associationDidEndEditing`

editingContextPresentErrorMessage

```
public void editingContextPresentErrorMessage(
   EOEditingContext anEditingContext,
   String errorMessage)
```

Invoked by *anEditingContext* as part of the EOEditingContext.MessageHandlers interface, this method presents an attention panel with *errorMessage* as the message to display.

editingContextShouldContinueFetching

```
public boolean editingContextShouldContinueFetching(
   com.apple.client.eocontrol.EOEditingContext anEditingContext,
```

(com.apple.client.eointerface only) Invoked by *anEditingContext* as part of the EOEditingContext.MessageHandlers interface, this method presents an attention panel prompting the user about whether or not to continue fetching the current result set.

editingContextWillSaveChanges

```
public 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 throws an exception if it returns `false`. Thus, if there's an association that refuses to end editing, anEditingContext doesn't save changes.

editorHasChangesForEditingContext

```
public boolean editorHasChangesForEditingContext(EOEditingContext anEditingContext)
```

Invoked by *anEditingContext* as part of the EOEditors interface, this method returns `false` if any association is editing, `true` otherwise.

See Also: `editingAssociation`, `associationDidBeginEditing`, `associationDidEndEditing`

enabledToSetSelectedObjectValueForKey

`public boolean enabledToSetSelectedObjectValueForKey(String key)`

Returns `true` to indicate that a single value association (such as an EOControlAssociation for a NSTextField) should be enabled for setting *key*, `false` 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

`public boolean endEditing()`

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

See Also: `editingAssociation`

enterQueryMode

`public void enterQueryMode(Object sender)`

This action method invokes `setInQueryMode` with an argument of `true`.

equalToQueryValues

`public 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

`public boolean fetch()`

Attempts to fetch objects from the EODataSource, returning `true` on success and `false` on failure.

Before fetching, invokes `endEditing` and sends `displayGroupShouldFetch` to the delegate, returning `false` if either of these methods does. If both return `true`, sends a `fetchObjects` message to the receiver's EODataSource to replace the object array, and if successful sends the delegate a `displayGroupDidFetchObjects` message.

fetch

```
public void fetch(Object anObject)
```

(com.apple.client.eointerface only) This action method invokes `fetch`.

fetchesOnLoad

```
public boolean fetchesOnLoad()
```

Returns `true` if the receiver fetches automatically after being loaded from a nib file, `false` if it must be told explicitly to fetch. The default is `false`. You can set this behavior in Interface Builder using the Inspector panel.

See Also: `fetch`, `setFetchesOnLoad`

finishInitialization

```
public void finishInitialization()
```

(com.apple.client.eointerface only) Invoked from the EODisplayGroup constructor and from `awakeFromNib` to finish initializing a newly created display group. You should never invoke this method directly. Sets the receiver's editing context to it's data source's editing context (if available), registers the receiver for `ObjectsChangedInEditingContextNotification`s and `InvalidatedAllObjectsInStoreNotification`s, establishes the receiver as an editor for the editing context, and establishes the receiver as the editing context's message handler (unless the editing context already has a message handler).

greaterThanQueryValues

```
public 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`

inQueryMode

```
public boolean inQueryMode()
```

Returns `true` to indicate that the receiver is in query mode, `false` 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

```
public void insert(Object sender)
```

(com.apple.client.eointerface only) This action method invokes `insertNewObjectAtIndex` with an index just past the first index in the selection, or 0 if there's no selection.

insertedObjectDefaultValues

```
public 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.

insertNewObjectAtIndex

```
public Object insertNewObjectAtIndex(int anIndex)
```

(com.apple.yellow.eointerface only) Asks the receiver's EODataSource to create a new object by sending it a `createObject` message, then inserts the new object using `insertObjectAtIndex`. 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 `displayGroupCreateObjectFailed` message or, if the delegate doesn't respond, opens an attention panel to inform the user of the error.

See Also: `insert`

insertObjectAtIndex

```
public void insertObjectAtIndex(
   Object anObject,
   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 `displayGroupShouldInsertObject`. If the delegate refuses, *`anObject`* isn't inserted. After successfully inserting the object, this method informs the delegate with a `displayGroupDidInsertObject` message, and selects the newly inserted object. Throws an exception if *`index`* is out of bounds.

Unlike the `insertNewObjectAtIndex` 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.

```
public Object insertNewObjectAtIndex(int anIndex)
```

(com.apple.client.eointerface only) Asks the receiver's EODataSource to create a new object by sending it a `createObject` message, then inserts the new object using `insertObjectAtIndex`. 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 `displayGroupCreateObjectFailed` message or, if the delegate doesn't respond, opens an attention panel to inform the user of the error.

lessThanQueryValues

`public 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

`public 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

`public NSArray observingAssociations()`

Returns all EOAssociations that observe the receiver's objects.

qualifier

`public 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

```
public 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

```
public 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.

```
public void qualifyDataSource(Object sender)
```

(com.apple.client.eointerface only) This action method simply invokes the no argument version.

qualifyDisplayGroup

```
public 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.

```
public void qualifyDisplayGroup(Object sender)
```

(com.apple.client.eointerface only) This action method simply invokes the no argument version.

queryBindingValues

```
public 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

```
public NSDictionary queryOperatorValues()
```

Returns a dictionary of operators to use on items in the query dictionaries (`equalToQueryValues`, `greaterThanQueryValues`, and `lessThanQueryValues`). If a key in a query dictionary also exists in `queryOperatorValues`, that operator for that key is used.

See Also: `qualifierFromQueryValues`

redisplay

```
public void redisplay()
```

Notifies all observing associations to redisplay their values.

See Also: `observingAssociations`

selectedObject

```
public Object selectedObject()
```

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

See Also: `displayedObjects`, `selectionIndexes`

selectedObjects

`public NSArray selectedObjects()`

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

See Also: `displayedObjects`, `selectionIndexes`

selectedObjectValueForKey

`public Object selectedObjectValueForKey(String *key*)`

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

See Also: `valueForObjectAtIndex`

selectionChanged

`public boolean selectionChanged()`

Returns `true` if the selection has changed and not all observers have been notified, `false` otherwise. EOAssociations use this in their `subjectChanged` methods to determine what they need to update.

See Also: `contentsChanged`

selectionIndexes

`public NSArray selectionIndexes()`

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

See Also: `displayedObjects`, `selectedObjects`, `selectedObject`, `setSelectionIndexes`

selectNext

`public boolean selectNext()`

Attempts to select the object just after the currently selected one, returning `true` if successful and `false` if not. The selection is altered in this way:

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

`public void selectNext(Object` *`sender`*`)`

(com.apple.client.eointerface only) This action method simply invokes the no argument version.

selectObject

`public boolean selectObject(Object` *`anObject`*`)`

Returns `true` to indicate that the receiver has found and selected anObject, `false` 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

`public boolean selectObjectsIdenticalTo(NSArray` *`objects`*`)`

Attempts to select the objects in the receiver's displayed objects array which are equal to those of *`objects`*, returning `true` if successful and `false` otherwise.

selectObjectsIdenticalToSelectFirstOnNoMatch

```
public boolean selectObjectsIdenticalTo(
   NSArray objects,
   boolean flag)
```

(com.apple.client.eointerface only) Selects the objects in the receiver's displayed objects array that are equal to those of *objects*, returning `true` if successful and `false` otherwise. If no objects in the displayed objects array match *objects* and *flag* is `true`, attempts to select the first object in the displayed objects array.

See Also: `setSelectionIndexes`

selectPrevious

```
public boolean selectPrevious()
```

Attempts to select the object just before the presently selected one, returning `true` if successful and `false` if not. The selection is altered in this way:

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

```
public void selectPrevious(Object anObject)
```

(com.apple.client.eointerface only) This action method simply invokes the no argument version.

selectsFirstObjectAfterFetch

```
public boolean selectsFirstObjectAfterFetch()
```

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

See Also: `displayedObjects`, `fetch`, `setSelectsFirstObjectAfterFetch`

setDataSource

`public 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 `ObjectsChangedInEditingContextNotification` and `InvalidatedAllObjectsInStoreNotification` 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

`public void setDefaultStringMatchFormat(String format)`

Sets how pattern matching will be performed on String values in the query dictionaries (`equalToQueryValues`, `greaterThanQueryValues`, and `lessThanQueryValues`). This format is used for query dictionary properties that have String 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 query 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

`public void setDefaultStringMatchOperator(String matchOperator)`

Sets the operator used to perform pattern matching for String values in the `queryMatch` dictionary. This operator is used for properties listed in the query dictionaries (`equalToQueryValues`, `greaterThanQueryValues`, and `lessThanQueryValues`) that have String values and that do not have an associated entry in the `queryOperatorValues` dictionary. In these cases, the operator *`matchOperator`* 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

`public void setDelegate(Object anObject)`

Sets the receiver's delegate to *`anObject`*.

See Also: `delegate`

setEqualToQueryValues

`public 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

`public void setFetchesOnLoad(boolean flag)`

Controls whether the receiver automatically fetches its objects after being loaded from a nib file. If *flag* is `true` it does; if *flag* is `false` the receiver must be told explicitly to fetch. The default is `false`. You can also set this behavior in Interface Builder using the Inspector panel.

See Also: `fetch`, `fetchesOnLoad`

setGreaterThanQueryValues

`public 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

`public void setInQueryMode(boolean flag)`

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

See Also: `inQueryMode`, `enterQueryMode`

setInsertedObjectDefaultValues

`public 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.

See Also: `insertedObjectDefaultValues`

setLessThanQueryValues

`public 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

`public 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

`public 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

`public void setQualifier(EOQualifier aQualifier)`

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

Note: To set the qualifier used to fetch objects from the database, set the qualifier of the display group's `dataSource` (assuming that the data source is an EODatabaseDataSource).

If the receiver's delegate responds to `displayGroupDisplayArrayForObjects`, that method is used instead of the qualifier to filter the objects.

See Also: `displayedObjects`, `qualifier`, `qualifierFromQueryValues`, **setAuxiliaryQualifier (EODatabaseDataSource in EOAccess)**

setQueryBindingValues

`public void setQueryBindingValues(NSDictionary values)`

Sets the dictionary of values that a user wants to query on. You use this method to perform a query stored in the model file. Bind keys in the `queryBindingValues` dictionary to elements of your component that specify query values.

setQueryOperatorValues

`public void setQueryOperatorValues(NSDictionary values)`

Sets the dictionary of operators to use on items in the query dictionaries (`equalToQueryValues`, `greaterThanQueryValues`, and `lessThanQueryValues`). If a key in a query dictionary also exists in `queryOperatorValues`, that operator for that key is used.

setSelectedObject

`public void setSelectedObject(Object anObject)`

Sets the selected objects to *`anObject`*.

setSelectedObjects

```
public void setSelectedObjects(NSArray objects)
```

Sets the selected objects to *objects*.

setSelectedObjectValue

```
public boolean setSelectedObjectValue(
   Object value,
   String key)
```

Invokes `setValueForObject` with the first selected object, returning `true` if successful and `false` otherwise. This method should be invoked only by EOAssociation objects to propagate changes from display objects.

See Also: `setValueForObjectAtIndex`, `valueForObject`

setSelectionIndexes

```
public boolean setSelectionIndexes(NSArray indexes)
```

Selects the objects at *indexes* in the receiver's array if possible, returning `true` if successful and `false` if not (in which case the selection remains unaltered). *indexes* is an array of Numbers. 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 `false`, this method fails and returns `false`. This method then checks the delegate with a `displayGroupShouldChangeSelection` message. If the delegate returns `false`, this method also fails and returns `false`. If the receiver successfully changes the selection, its observers (typically EOAssociations) each receive a `subjectChanged` message.

setSelectsFirstObjectAfterFetch

```
public void setSelectsFirstObjectAfterFetch(boolean 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 `true` it does; if `flag` is `false` 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

```
public 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 `displayGroupDisplayArrayForObjects`, that method is used instead of the sort orderings to order the objects.

See Also: `displayedObjects`, `sortOrderings`

setUsesOptimisticRefresh

```
public void setUsesOptimisticRefresh(boolean flag)
```

Controls how the receiver redisplays on changes to objects. If `flag` is `true` it redisplays only when elements of its displayed objects array change; if `flag` is `false` 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 `false`. You can also change this setting in Interface Builder's Inspector panel using the Refresh All check box.

See Also: `usesOptimisticRefresh`

setValidatesChangesImmediately

```
public void setValidatesChangesImmediately(boolean 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 `validateValueForKey` 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 `true`, the receiver immediately presents an attention panel indicating the validation error. If this method is invoked with a *`flag`* of `false`, 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`

setValueForObject

```
public boolean setValueForObject(
   Object value,
   Object anObject,
   String key)
```

Sets a property of *`anObject`*, identified by *`key`*, to *`value`*. Returns `true` if successful and `false` otherwise. If a new value is set, sends the delegate a `displayGroupDidSetValueForObject` 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 `validateValueForKey` message, returning `false` if the object refuses to validate *`value`*. Otherwise, validation errors are checked by the EOEditingContext when it attempts to save changes.

See Also: `setValueForObjectAtIndex`, `setSelectedObjectValue`, `valueForObject`, `validatesChangesImmediately`

setValueForObjectAtIndex

```
public boolean setValueForObjectAtIndex(
   Object value,
   int index,
   String key)
```

Invokes `setValueForObject` with the object at *`index`*, returning `true` if successful and `false` otherwise. This method should be invoked only by EOAssociation objects to propagate changes from display objects.

See Also: `setSelectedObjectValue`,`valueForObjectAtIndex`

sortOrderings

```
public 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`

undoManager

```
public NSUndoManager undoManager()
```

(com.apple.client.eointerface only) Returns the receiver's undo manager.

updateDisplayedObjects

```
public void updateDisplayedObjects()
```

Recalculates the receiver's displayed objects array and redisplays. If the receiver's delegate responds to `displayGroupDisplayArrayForObjects`, 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

```
public 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 `true`. EOAssociations can use this method to optimize redisplay of their user interface objects.

usesOptimisticRefresh

```
public boolean usesOptimisticRefresh()
```

Returns `true` if the receiver redisplays only when its displayed objects change, `false` if it redisplays on any change in its EOEditingContext.

See Also: `setUsesOptimisticRefresh`

validatesChangesImmediately

```
public boolean validatesChangesImmediately()
```

Returns `true` if the receiver immediately handles validation errors, or `false` if it leaves errors for the EOEditingContext to handle when saving changes.

See Also: `setValidatesChangesImmediately`

valueForKeyObject

```
public Object valueForKeyObject(
   String key,
   com.apple.client.eocontrol.EOKeyValueCodingAdditions anObject)
```

(com.apple.client.eointerface only) Do not use this method. Use `valueForObjectKey` instead.

valueForObject

```
public Object valueForObject(
   Object anObject,
   String key)
```

(com.apple.yellow.eointerface only) Returns *`anObject`*'s value for the property identified by *`key`*.

valueForObjectKey

```
public Object valueForObjectKey(
   com.apple.yellow.eocontrol.EOKeyValueAdditions anObject,
   String key)
```

(com.apple.client.eointerface only) Returns `anObject`'s value for the property identified by `key`.

valueForObjectAtIndex

```
public Object valueForObjectAtIndex(
   int index,
   String key)
```

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

willChange

```
public void willChange()
```

Notifies observers that the receiver will change.

Notifications

DisplayGroupWillFetchNotification

```
public static final String DisplayGroupWillFetchNotification
```

Posted whenever an EODisplayGroup receives a `fetch` message. The notification contains:

Notification Object	The EODisplayGroup that received the `fetch` message.
Userinfo	None

EOForm

Inherits from:	EOMatrix : EOView : javax.swing.JPanel : javax.swing.JComponent : java.awt.Container : java.awt.Component Object
Implements:	java.awt.LayoutManager NSDisposable (EOView)
Package:	com.apple.client.eointerface

Class Description

The EOForm class is a subclass of EOMatrix that manages a collection of titled text fields laid out on a grid. Each title/text field pair is an EOFormCell.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

Interfaces Implemented

java.awt.LayoutManager

`addLayoutComponent`

`layoutContainer`

`minimumLayoutSize`

`preferredLayoutSize`

`removeLayoutComponent`

Method Types

Constructor

`EOForm`

Adding form cells

`add`

Constructors

EOForm

```
public EOForm(
   String debuggingHint,
   int rows,
    int cols,
    int rowSpacing,
    int colSpacing)
```

Returns a new EOForm object. The `debuggingHint` argument is a string you can use to uniquely identify the view. When the form is instantiated from a nib file, the `debuggingHint` is a string generated by Interface Builder.

Instance Methods

add

```
public java.awt.Component add(java.awt.Component formCell)
```

Adds `formCell`, an EOFormCell, to the receiver's collection of form cells.

addLayoutComponent

```
public void addLayoutComponent(String name, java.awt.Component component)
```

Simply returns.

layoutContainer

```
public void layoutContainer(java.awt.Container formCell)
```

Lays out the title and text field of `formCell`.

minimumLayoutSize

```
public java.awt.Dimension minimumLayoutSize(java.awt.Container aContainer)
```

Returns the value returned from `aContainer`'s `getMinimumSize`.

preferredLayoutSize

```
public java.awt.Dimension preferredLayoutSize(java.awt.Container aContainer)
```

Returns the value returned from *aContainer*'s `getPreferredSize`.

removeLayoutComponent

```
public void removeLayoutComponent(java.awt.Component aComponent)
```

Simply returns.

EOFormCell

Inherits from:	javax.swing.JComponent : java.awt.Container : java.awt.Component Object
Implements:	EOTextAssociation.JTextComponentAccess NSDisposable
Package:	com.apple.client.eointerface

Class Description

EOFormCell objects implement entries in EOForms.An EOFormCell has a **field component**, an editable EOTextField into which users enter data; and a **title component**, an uneditable EOTextField that identifies the purpose of the form cell's field component.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

For more information on forms and form cells, see the EOForm class specification.

Interfaces Implemented

EOTextAssociation.JTextComponentAccess

`jTextComponent`

NSDisposable

`dispose`

Method Types

Accessing the field component

`fieldComponent`

`jTextComponent`

Accessing the title and title component

`titleComponent`

`title`

`setTitle`

`setTitleWidth`

`titleWidth`

Instance Methods

fieldComponent

```
public EOTextField fieldComponent()
```

Returns the receiver's field component, the editable text field into which users enter data.

jTextComponent

```
public javax.swing.text.JTextComponent jTextComponent()
```

Returns the receiver's field component, the editable text field into which users enter data.

setTitle

`public void setTitle(String` *`aString`*`)`

Sets the receiver's title to *`aString`*. This is a convenience method for setting the text value of the receiver's `titleComponent`.

setTitleWidth

`public void setTitleWidth(int` *`width`*`)`

Sets the width of the receiver's `titleComponent`. Typically the width of the title component is handled automatically. You should never need to invoke this method.

title

```
public String title()
```

Returns the receiver's title. This is a convenience method for setting the text value of the receiver's `titleComponent`.

titleComponent

```
public EOTextField titleComponent()
```

Returns the receiver's title component, the uneditable text field that identifies the purpose of the `fieldComponent`.

titleWidth

```
public int titleWidth()
```

Returns the width of the receiver's `titleComponent`.

EOFrame

Inherits from:	javax.swing.JFrame : java.awt.Frame : java.awt.Window : java.awt.Container : java.awt.Component : Object
Implements:	NSDisposable
Package:	com.apple.client.eointerface

Class Description

An EOFrame is a window that uses an EOViewLayout to manage layout geometry.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

For more information on EOFrame's layout management, see the EOViewLayout class specification.

Interfaces Implemented

NSDisposable

```
dispose
```

Constructors

EOFrame

`public EOFrame(String` *`debuggingHint`*`)`

Creates a new EOFrame object. The *`debuggingHint`* argument is a string you can use to uniquely identify the view. When the form is instantiated from a nib file, the *`debuggingHint`* is a string generated by Interface Builder.

Instance Methods

setSize

`public void setSize(int` *`width`*`, int` *`height`*`)`

Sets the receiver's width and height.

EOGenericControlAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

Class Description

EOGenericControlAssociation is the abstract superclass of EOControlAssociation and EOActionCellAssociation. You never use instances of this class directly; its `isUsableWithObject` method always returns `false`. See the subclass specifications for more information.

Note: This class doesn't exist in the com.apple.client.eointerface package.

Usable With	Aspects	Object Keys Taken
Nothing	value	target
	enabled	delegate

Instance Methods

control

```
public com.apple.yellow.application.NSControl control()
```

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

editingAssociation

```
public 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.

EOImageAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : Object
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.client.eointerface

Class Description

EOImageAssociation associates the contents of its `ValueAspect`'s display group with an EOImageView.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

Usable With

EOImageView

Aspects

ValueAspect	An NSData containing the image data
URLAspect	A URL from which to retrieve the image

Constructors

EOImageAssociation

`public EOImageAssociation(Object *aDisplayObject*)`

Creates a new EOImageAssociation to monitor and update the value in *aDisplayObject*, an EOImageView.

You normally set up associations in Interface Builder, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` (EOAssociation), `establishConnection` **(EOAssociation)**

Instance Methods

establishConnection

`public void establishConnection()`

See the `establishConnection` method description in the superclass EOAssociation.

image

`public java.awt.Image image()`

Returns the receiver's EOImageView's `image`.

imageWithData

```
public java.awt.Image imageWithData(NSData data)
```

Creates an Image from the data in `data`.

isUsableWithObject

```
public boolean isUsableWithObject(Object aDisplayObject)
```

Returns `true` if `aDisplayObject` is an instance of EOImageView, `false` otherwise.

See Also: `isUsableWithObject` (EOAssociation)

primaryAspect

```
public String primaryAspect()
```

Returns `EOAssociation.ValueAspect`.

See Also: `primaryAspect` (EOAssociation)

subjectChanged

```
public void subjectChanged()
```

See the `subjectChanged` method description in the superclass EOAssociation.

EOImageView

Inherits from:	javax.swing.JComponent java.awt.Container : java.awt.Component Object
Package:	com.apple.client.eointerface

Class Description

The EOImageView class is used to display images (java.awt.Image objects) in Java Client applications.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

Constants

EOImageView defines the following `int` constants to specify the scaling behavior of an EOImageView:

Constant	Scaling Behavior
`ScaleNone`	No scaling
`ScaleProportionally`	Scales in proportion to the image size
`ScaleToFit`	Scales to fit the portion of the user interface the image view occupies
`ScaleProportionallyIfTooLarge`	Scales in proportion to the image size, but only if the image is too large to fit its portion of the user interface (the image view never scales the image to be larger)

Method Types

Accessing the image

`image`

`setImage`

Configuring scaling behavior

`setImageScaling`

`imageScaling`

`setScalingHints`

`scalingHints`

Painting

`imageUpdate`

`paint`

`setBorder`

`setBounds`

Instance Methods

image

```
public java.awt.Image image()
```

Returns the receiver's image.

imageScaling

```
public int imageScaling()
```

Returns the type of scaling the receiver uses. The return value is one of:

- `ScaleNone`
- `ScaleProportionally`
- `ScaleToFit`
- `ScaleProportionallyIfTooLarge`

imageUpdate

```
public boolean imageUpdate(
   java.awt.Image image,
   int flags,
   int x,
   int y,
   int width,
   int height)
```

See the method description for `imageUpdate` in Sun's JComponent class documentation.

paint

```
public void paint(java.awt.Graphics g)
```

See the method description for `setBorder` in Sun's JComponent class documentation.

scalingHints

```
public int scalingHints()
```

Returns the receiver's scaling hints—a constant identifying the algorithm the receiver uses to scale its image.

setBorder

```
public void setBorder(javax.swing.border.Border border)
```

See the method description for `setBorder` in Sun's JComponent class documentation.

setBounds

```
public void setBounds(
   int x,
   int y,
   int width,
   int height)
```

See the method description for `setBounds` in Sun's Component class documentation.

setImage

```
public void setImage(java.awt.Image image)
```

Sets the receiver's image to `image` and repaints (only if `image` is different from the receiver's old image).

setImageScaling

```
public void setImageScaling(int imageScaling)
```

Sets the scaling behavior of the receiver; that is, identifies the circumstances under which the receiver scales. The `imageScaling` argument should be one of the following constants (defined in EOImageView):

- `ScaleNone`
- `ScaleProportionally`
- `ScaleToFit`
- `ScaleProportionallyIfTooLarge`

The default scaling behavior is `ScaleProportionallyIfTooLarge`. For more information on these constants, see "Constants" (page 128).

setScalingHints

```
public void setScalingHints(int scalingHints)
```

Sets the algorithm the receiver uses to scale it's image.The `scalingHints` argument should be one of the following constants (defined in java.awt.Image):

- `SCALE_DEFAULT`
- `SCALE_FAST`
- `SCALE_SMOOTH`
- `SCALE_REPLICATE`
- `SCALE_AREA_AVERAGING`

The default is `SCALE_SMOOTH`. For more information on the algorithms identified by these constants, see Sun's Image class documentation

EOMasterCopyAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

Class Description

An EOMasterCopyAssociation object synchronizes two EODisplayGroups that share the same data source but have different qualifiers.

Note: This class doesn't exist in the com.apple.client.eointerface package.

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.)

Constructors

EOMasterCopyAssociation

```
public EOMasterCopyAssociation(Object aDisplayObject)
```

Creates a new EOMasterCopyAssociation to monitor and update the value in *`aDisplayObject`*, an EODisplayGroup.

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` **(EOAssociation),** `establishConnection` **(EOAssociation)**

EOMasterDetailAssociation

Inherits from:	(com.apple.client.eointerface) EOAssociation : EODelayedObserver (EOControl) : Object (com.apple.yellow.eointerface) EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver) (com.apple.client.eointerface only) NSDisposable (EOAssociation)
Package:	com.apple.client.eointerface com.apple.yellow.eointerface

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 com.apple.yellow.eointerface, by contrast, with an EOMasterPeerAssociation, the two EODisplayGroups are independent of each other (EOMasterPeerAssociation is not a com.apple.client.eointerface 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	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.

Constructors

EOMasterDetailAssociation

```
public EOMasterDetailAssociation(Object aDisplayObject)
```

Creates a new EOMasterDetailAssociation to monitor and update the value in *aDisplayObject*, an EODisplayGroup.

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` (EOAssociation), `establishConnection` **(EOAssociation)**

Instance Methods

isUsableWithObject

```
public boolean isUsableWithObject(Object aDisplayObject)
```

(com.apple.client.eointerface) Returns `true` if *aDisplayObject* is an instance of EODisplayGroup and its `dataSource` is either `null` or an EODetailDataSource (EOControl).

See Also: `isUsableWithObject` (EOAssociation)

primaryAspect

```
public String primaryAspect()
```

(com.apple.client.eointerface) Returns `EOAssociation.ParentAspect`.

See Also: `primaryAspect` (EOAssociation)

priority

```
public int priority()
```

Returns EOObserverPrioritySecond (one notch above the default priority). This guarantees that changes in the master are propagated to the detail before any other updates are made.

subjectChanged

```
public void subjectChanged()
```

See the `subjectChanged` method description in the superclass EOAssociation.

EOMasterPeerAssociation

Inherits from:	EOMasterDetailAssociation : EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

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.

Note: This class doesn't exist in the com.apple.client.eointerface package.

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 relationship between salesman and sales based on city ID (the relationship is not a class property). You can then display that information using an EOMasterPeerAssociation.

Constructors

EOMasterPeerAssociation

```
public EOMasterPeerAssociation(Object aDisplayObject)
```

Creates a new EOMasterDetailAssociation to monitor and update the value in `aDisplayObject`, an EODisplayGroup.

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` **(EOAssociation),** `establishConnection` **(EOAssociation)**

EOMatrix

Inherits from:	EOView : javax.swing.JPanel : javax.swing.JComponent : java.awt.Container : java.awt.Component Object
Implements:	NSDisposable (EOView)
Package:	com.apple.client.eointerface

Class Description

EOMatrix is a class used to group collections of mutually exclusive JRadioButtons and to lay them out on a grid. It is a subclass of EOView that uses a java.awt.GridLayout.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

For more information on the way a matrix of JRadioButtons behaves, see the Sun class documentation for javax.swing.ButtonGroup.

Constructors

EOMatrix

```
public EOMatrix(String debuggingHint,
   int rows,
   int cols,
   int rowSpacing,
   int colSpacing)
```

Creates and returns a new EOMatrix object. The *debuggingHint* argument is a string you can use to uniquely identify the view. When the form is instantiated from a nib file, the *debuggingHint* is a string generated by Interface Builder.

Instance Methods

add

```
public java.awt.Component add(java.awt.Component radioButton)
```

Adds *radioButton* if it's an instance of javax.swing.JRadioButton, otherwise simply returns.

EOMatrixAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

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).

Note: This class doesn't exist in the com.apple.client.eointerface package.

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).

Constructors

EOMatrixAssociation

```
public EOMatrixAssociation(Object aDisplayObject)
```

Creates a new EOMatrixAssociation to monitor and update the value in *aDisplayObject*, an NSMatrix (Application Kit).

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` **(EOAssociation)**, `establishConnection` **(EOAssociation)**

EOPickTextAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

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.

Note: This class doesn't exist in the com.apple.client.eointerface package.

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.

Constructors

EOPickTextAssociation

```
public EOPickTextAssociation(Object aDisplayObject)
```

Creates a new EOPickTextAssociation to monitor and update the row values in *`aDisplayObject`*, an NSControl (Application Kit) which has a text as an attribute.

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` **(EOAssociation),** `establishConnection` **(EOAssociation)**

EOPopUpAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

Class Description

An EOPopUpAssociation object displays an attribute or to-one relationship value in an NSPopUpButton (Application Kit).

Note: This class doesn't exist in the com.apple.client.eointerface package.

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.

Constructors

EOPopUpAssociation

`public EOPopUpAssociation(Object aDisplayObject)`

Creates a new EOPopUpAssociation to monitor and update the values in *aDisplayObject*, an NSPopUpList (Application Kit).

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` (EOAssociation), `establishConnection` **(EOAssociation)**

Instance Methods

setTagValueForOther

`public 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

`public int tagValueForOther()`

Returns the "unknown" tag.

EOQuickTimeAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : Object
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.client.eointerface

Class Description

EOQuickTimeAssociation associates the contents of its `URLAspect`'s display group with an EOQuickTimeView.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

Usable With

EOQuickTimeView

Aspects

URLAspect	A URL for the location of the QuickTime movie.

Constructors

EOQuickTimeAssociation

`public EOQuickTimeAssociation(Object` *`aDisplayObject`*`)`

Creates a new EOQuickTimeAssociation to monitor and update the value in *`aDisplayObject`*, an EOQuickTimeView.

You normally set up associations in Interface Builder, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` (EOAssociation), `establishConnection` **(EOAssociation)**

Instance Methods

breakConnection

`public void breakConnection()`

See the `breakConnection` method description in the superclass EOAssociation.

isUsableWithObject

`public boolean isUsableWithObject(Object` *`aDisplayObject`*`)`

Returns `true` if *`aDisplayObject`* is an instance of EOQuickTimeView, `false` otherwise.

See Also: `isUsableWithObject` (EOAssociation)

primaryAspect

```
public String primaryAspect()
```

Returns `EOAssociation.URLAspect`.

See Also: `primaryAspect` (EOAssociation)

subjectChanged

```
public void subjectChanged()
```

See the `subjectChanged` method description in the superclass EOAssociation.

EOQuickTimeView

Inherits from:	javax.swing.JPanel javax.swing.JComponent java.awt.Container : java.awt.Component Object
Package:	com.apple.client.eointerface

Class Description

The EOQuickTimeView class is used to display QuickTime movies in Java Client applications.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

Constants

EOQuickTimeView defines the following `int` constants to identify resizing behavior:

- `QuickTimeCanvasNoResizing`
- `QuickTimeCanvasAspectResizing`
- `QuickTimeCanvasFreeResizing`

- `QuickTimeCanvasIntegralResizing`
- `QuickTimeCanvasPerformanceResizing`
- `QuickTimeCanvasHorizontalResizing`
- `QuickTimeCanvasVerticalResizing`

These same constants are also defined in quicktime.app.display.QTCanvas. They are duplicated in EOQuickTimeView for convenience. For information on the resizing behavior associated with these constants, see the QTCanvas documentation.

Method Types

Determining if the QuickTime system is available

`isQuickTimeAvailable`

Setting the QuickTime movie and player

`movie`

`setMovie`

`setMovieFromURL`

`player`

`setPlayer`

Configuring resizing behavior

`setCanvasResizing`

`canvasResizing`

Painting

`getPreferredSize`

`setBounds`

Static Methods

isQuickTimeAvailable

```
public static boolean isQuickTimeAvailable()
```

Returns `true` if the QuickTime for Java classes are in the class path and are loaded; `false` otherwise. If the classes are in the class path but aren't loaded, this method attempts to load them.

Instance Methods

canvasResizing

```
public int canvasResizing()
```

Returns an integer that identifies the receiver's resizing behavior. The return value is one of the following constants (defined in EOQuickTimeView):

- `QuickTimeCanvasNoResizing`
- `QuickTimeCanvasAspectResizing`
- `QuickTimeCanvasFreeResizing`
- `QuickTimeCanvasIntegralResizing`
- `QuickTimeCanvasPerformanceResizing`
- `QuickTimeCanvasHorizontalResizing`
- `QuickTimeCanvasVerticalResizing`

For more information on the resizing constants, see "Constants" (page 161).

getPreferredSize

```
public java.awt.Dimension getPreferredSize()
```

See the method description for `getPreferredSize` in Sun's JComponent class documentation.

movie

```
public Object movie()
```

Returns the receiver's QuickTime movie, a quicktime.std.movies.Movie.

player

```
public Object player()
```

Returns the receiver's QuickTime player, a quicktime.app.players.QTPlayer.

setBounds

```
public void setBounds(
   int x,
   int y,
   int width,
   int height)
```

See the method description for `setBounds` in Sun's Component class documentation.

setCanvasResizing

```
public void setCanvasResizing(int canvasResizing)
```

Sets the resizing behavior of the receiver. The *`canvasResizing`* argument should be one of the following constants (defined in EOQuickTimeView):

- `QuickTimeCanvasNoResizing`
- `QuickTimeCanvasAspectResizing`
- `QuickTimeCanvasFreeResizing`
- `QuickTimeCanvasIntegralResizing`

- `QuickTimeCanvasPerformanceResizing`
- `QuickTimeCanvasHorizontalResizing`
- `QuickTimeCanvasVerticalResizing`

The default resizing behavior is `QuickTimeCanvasAspectResizing`. For more information on these constants, see "Constants" (page 161).

setMovie

```
public void setMovie(Object movie)
```

Sets the receiver's QuickTime movie to *movie*, a quicktime.std.movies.Movie.

setMovieFromURL

```
public void setMovieFromURL(String url)
```

Sets the receiver's QuickTime movie to the movie at *url*.

setPlayer

```
public void setPlayer(Object player)
```

Sets the receiver's QuickTime player to *player*, a quicktime.app.players.QTPlayer.

EORadioMatrixAssociation

Inherits from:	EOAssociation : EODelayedObserver : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

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.

Note: This class doesn't exist in the com.apple.client.eointerface package.

Usable With
NSMatrix

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

```
public 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

```
public int tagValueForOther()
```

Returns the "unknown" tag.

EORecursiveBrowserAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

Class Description

An EORecursiveBrowserAssociation is the default association for use with a multi-column NSBrowser (Application Kit).

Note: This class doesn't exist in the com.apple.client.eointerface package.

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 (`true`) or a branch (`false`).
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 `false` 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 `false` 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.

Constructors

EORecursiveBrowserAssociation

```
public EORecursiveBrowserAssociation(Object aDisplayObject)
```

Creates a new EORecursiveBrowserAssociation to monitor and update the values in *`aDisplayObject`*, an NSBrowser (Application Kit).

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` **(EOAssociation),** `establishConnection` **(EOAssociation)**

EOTable

Inherits from:	javax.swing.JScrollPane : javax.swing.JComponent : java.awt.Container : java.awt.Component Object
Implements:	NSDisposable
Package:	com.apple.client.eointerface

Class Description

The EOTable class is used to represent tables of data. An EOTable object uses a JTable to do its work. As a subclass of JScrollPane, an EOTable wraps its JTable in a scroll view and adds the JTable's JTableHeader to the EOTable's column header. If you want to configure or message an EOTable's JTable, you can access the it with the method `jTable`.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

Interfaces Implemented

NSDisposable

```
dispose
```

Instance Methods

debuggingHint

```
public String debuggingHint()
```

Returns the receiver's debugging hint.

jTable

```
public javax.swing.JTable jTable()
```

Returns the receiver's JTable.

setDebuggingHint

```
public void setDebuggingHint(String hint)
```

Sets the receiver's debugging hint to *hint*.

EOTableAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : Object
Implements:	javax.swing.event.ListSelectionListener EOObserving (EODelayedObserver) NSDisposable (EOAssociation)
Package:	com.apple.client.eointerface

Class Description

EOTableAssociation associates the contents of its `SourceAspect`'s display group with an EOTable (an object that places a javax.swing.JTable in a scroll view). In general use, it should never be necessary to explicitly instantiate this class, as EOTableColumnAssociation's `setTable` assures that an instance exists for its *`table`*.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

Usable With
EOTable

Aspects
`EOAssociation.EnabledAspect`
`EOAssociation.SourceAspect`

Interfaces Implemented

javax.swing.event.ListSelectionListener

`valueChanged`

Constructors

EOTableAssociation

```
public EOTableAssociation(Object aDisplayObject)
```

Creates a new EOTableAssociation to monitor and update the value in *`aDisplayObject`*, an EOTable.

In general use, it should never be necessary to explicitly instantiate this class, as EOTableColumnAssociation's `setTable` assures that an instance exists for its *`table`*.

See Also: `bindAspect` (EOAssociation), `establishConnection` (EOAssociation)

Static Methods

instanceForTable

```
public static EOTableAssociation instanceForTable(Object table)
```

Invoked from EOTableColumnAssociation's `setTable` to ensure that an EOTableAssociation has been created for *`table`*.

Instance Methods

addColumnAssociation

```
public void addColumnAssociation(EOTableColumnAssociation aTableColumnAssociation)
```

Adds *aTableColumnAssociation* to the receiver's set of EOTableColumnAssociations. If the receiver's `SourceAspect` is unbound, this method binds it to *aTableColumnAssociation*'s display group and then invokes `establishConnection`.

breakConnection

```
public void breakConnection()
```

See the `breakConnection` method description in the superclass EOAssociation.

editingAssociation

```
public EOTableColumnAssociation editingAssociation()
```

Returns the EOTableColumnAssociation bound to the column being edited in the receiver's display object, if any.

establishConnection

```
public void establishConnection()
```

See the `establishConnection` method description in the superclass EOAssociation.

isUsableWithObject

`public boolean isUsableWithObject(Object candidate)`

Returns `true` if *`candidate`* is an instance of EOTable and its `jTable` is an instance of JTable, *`false`* otherwise.

See Also: `isUsableWithObject` (EOAssociation)

primaryAspect

`public String primaryAspect()`

Returns `SourceAspect`.

See Also: `primaryAspect` (EOAssociation)

removeColumnAssociation

`public void removeColumnAssociation(EOTableColumnAssociation aTableColumnAssociation)`

Removes *`aTableColumnAssociation`* from the receiver's set of EOTableColumnAssociations. If *`aTableColumnAssociation`* is the last of the receiver's column associations, it prepares itself for garbage collection.

subjectChanged

`public void subjectChanged()`

See the `subjectChanged` method description in the superclass EOAssociation.

valueChanged

`public void valueChanged(com.sun.java.swing.event.ListSelectionEvent event)`

EOTableAssociation listens to its display object's TableModel in order to synchronize the selection indices of its `SourceAspect`'s EODisplayGroup with those of the model. This method represents the association's implementation of the ListSelectionListener interface.

C L A S S

EOTableColumnAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : Object
Implements:	javax.swing.event.ListSelectionListener EOObserving (EODelayedObserver) NSDisposable (EOAssociation)
Package:	com.apple.client.eointerface

Class Description

An EOTableColumnAssociation associates a single attribute of all enterprise objects in its `ValueAspect`'s EODisplayGroup with a Swing JTable TableColumn. The value of each object's attribute is displayed in its corresponding row.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

By far the easiest way to configure EOTableColumnAssociations is in Interface Builder, but they may also be instantiated programmatically. Because Swing's TableColumn maintains no reference to its containing JTable, this relationship must be explicitly specified via `setTable` before `establishConnection` is invoked.

Usable With

javax.swing.table.TableColumn

Aspects	
`BoldAspect`	
`EnabledAspect`	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.
`ItalicAspect`	
`ValueAspect`	An attribute of the objects, displayed in each row of the TableColumn.

Constructors

EOTableColumnAssociation

`public EOTableColumnAssociation(Object` *`aDisplayObject`*`)`

Creates a new EOTableAssociation to monitor and update the value in *`aDisplayObject`*, a javax.swing.table.TableColumn.

You normally set up associations in Interface Builder, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections. Because Swing's TableColumn maintains no reference to its containing JTable, this relationship must be explicitly specified via `setTable` before `establishConnection` is invoked

See Also: `bindAspect` **(EOAssociation)**

Static Methods

setTableColumnCustomizer

```
public static void setTableColumnCustomizer(TableColumnCustomizer tableColumnCustomizer)
```

Sets `tableColumnCustomizer` as the object that determines associations' editors and renderers. By default, an EOTableColumnAssociation's editor is the corresponding TableColumn's editor; or, if the TableColumn doesn't have an editor, an EOTextColumnEditor is used. Similarly, an EOTableColumnAssociation's renderer is the corresponding TableColumn's renderer; or, if the TableColumn doesn't have an editor, a javax.swing.table.DefaultTableCellRenderer is used.

tableColumnCustomizer

```
public static TableColumnCustomizer tableColumnCustomizer()
```

Returns the object that specifies editors and renderers for associations.

Instance Methods

format

```
public java.text.Format format()
```

Returns the java.lang.text.Format used to format values bound to the receiver's `ValueAspect` for display and editing.

isEditableAtRow

```
public boolean isEditableAtRow(int row)
```

Returns whether or not the property bound to the receiver's `ValueAspect` is editable at *`row`*, as determined by the `EnabledAspect`. If this aspect is bound, a non-zero value at *`row`* indicates that the property may be edited. If the `EnabledAspect` is unbound all rows are considered editable.

primaryAspect

```
public String primaryAspect()
```

Returns `ValueAspect`.

setFormat

```
public void setFormat(java.text.Format aFormat)
```

Sets the java.lang.text.Format object to use in formatting values bound to the receiver's `ValueAspect` for display and editing.

setTable

```
public void breakConnection()
```

Because TableColumn maintains no reference to its containing JTable, the consumer must explicitly specify this relationship by invoking `setTable` *before* `establishConnection`. This method also assures that an instance of EOTableAssociation exists for *`table`*.

EOTableViewAssociation

Inherits from:	EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver)
Package:	com.apple.yellow.eointerface

Class Description

An EOTableViewAssociation object manages the individual EOColumnAssociations between an NSTableView (Application Kit) and an EODisplayGroup.

Note: This class doesn't exist in the com.apple.client.eointerface package.

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.
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`

Sorting

`setSortsByColumnOrder`

`sortsByColumnOrder`

Accessing the active EOColumnAssociation

`editingAssociation`

Table view data source methods

`numberOfRowsInTableView`

`tableViewSetObjectValueForLocation`

`tableViewObjectValueForLocation`

Table view delegate methods

`tableViewShouldEditLocation`

`tableViewWillDisplayCell`

Table view notification methods

`tableViewSelectionDidChange`

Control delegate methods

`controlDidFailToFormatStringErrorDescription`

`controlIsValidObject`

`controlTextShouldBeginEditing`

Constructors

EOTableViewAssociation

```
public EOTableViewAssociation(Object aDisplayObject)
```

Creates a new EOTableViewAssociation to manage EOColumnAssociations associated with *aDisplayObject*, an NSTableView (Application Kit).

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` (EOAssociation), `establishConnection` **(EOAssociation)**

Static Methods

bindToTableView

```
public static void bindToTableView(
   com.apple.yellow.application.NSTableView aTableView,
   EODisplayGroup aDisplayGroup)
```

Creates an EOTableViewAssociation, binding *aTableView* to *aDisplayGroup*, if there isn't already a table view association for *aTableView*. EOColumnAssociation's `establishConnection` invokes this method to guarantee the presence of a coordinating EOTableViewAssociation.

Instance Methods

controlDidFailToFormatStringErrorDescription

```
public boolean controlDidFailToFormatStringErrorDescription(
   com.apple.yellow.application.NSControl aTableView,
   String aString,
   String errorDescription)
```

Forwards the message to the receiver's editing association.

See Also: `editingAssociation`

controlIsValidObject

```
public boolean controlIsValidObject(
   com.apple.yellow.application.NSControl aTableView,
   Object anObject)
```

Forwards the message to the receiver's editing association.

See Also: `editingAssociation`

controlTextShouldBeginEditing

```
public boolean controlTextShouldBeginEditing(
   com.apple.yellow.application.NSControl aTableView,
   com.apple.yellow.application.NSText fieldEditor)
```

Forwards the message to the receiver's editing association.

See Also: `editingAssociation`

editingAssociation

```
public EOColumnAssociation editingAssociation()
```

Returns the EOColumnAssociation for the NSTableView cell being edited, or `null` if no cell is being edited.

numberOfRowsInTableView

```
public int numberOfRowsInTableView(com.apple.yellow.application.NSTableView aTableView)
```

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

See Also: `displayedObjects` **(EODisplayGroup)**

setSortsByColumnOrder

```
public void setSortsByColumnOrder(boolean flag)
```

Controls whether the receiver applies a sort ordering to its EODisplayGroup. If `flag` is `true`, 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 `false`, 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

```
public boolean sortsByColumnOrder()
```

Returns `true` if the receiver assigns EOSortOrderings (EOControl) to its EODisplayGroup based on the sorting selectors of its EOColumnAssociations, `false` if it doesn't alter the display group's sort ordering.

tableViewObjectValueForLocation

```
public Object tableViewObjectValueForLocation(
   com.apple.yellow.application.NSTableView aTableView,
   com.apple.yellow.application.NSTableColumn aTableColumn,
   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

```
public void tableViewSelectionDidChange(NSNotification aNotification)
```

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

See Also: `setSelectionIndexes` **(EODisplayGroup)**

tableViewSetObjectValueForLocation

```
public void tableViewSetObjectValueForLocation(
   com.apple.yellow.application.NSTableView aTableView,
   Object value,
   com.apple.yellow.application.NSTableColumn aTableColumn,
   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.

tableViewShouldEditLocation

```
public boolean tableViewShouldEditLocation(
   com.apple.yellow.application.NSTableView aTableView,
   com.apple.yellow.application.NSTableColumn aTableColumn,
   int rowIndex)
```

Returns `false` if the `enabled` aspect is bound and its value for the object at *rowIndex* is 0. 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.

tableViewWillDisplayCell

```
public void tableViewWillDisplayCell(
   com.apple.yellow.application.NSTableView aTableView,
   Object aCell,
   com.apple.yellow.application.NSTableColumn aTableColumn,
   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.

EOTextArea

Inherits from:	javax.swing.JScrollPane : javax.swing.JComponent : java.awt.Container : java.awt.Component Object
Implements:	EOTextAssociation.JTextComponentAccess
Package:	com.apple.client.eointerface

Class Description

EOTextArea, a subclass of javax.swing.JScrollPane, is used to represent scrolling text regions. An EOTextArea object uses a JTextArea to do its work. The main business of an EOTextArea is to configure the JTextArea's behavior and appearance. An EOTextArea's JTextArea has a vertical scroll bar but not a horizontal scroll bar and it wraps its lines of text. If you want to perform additional configuration on an EOTextArea's JTextArea, you can access the JTextArea with the method `jTextArea`.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

Interfaces Implemented

EOTextAssociation.JTextComponentAccess

`jTextComponent`

Method Types

Accessing the text area's JTextArea

`jTextArea`

Methods forwarded to the text area's JTextArea

`setEditable`

`setOpaque`

`setSize`

`setText`

Instance Methods

jTextArea

```
public javax.swing.JTextArea jTextArea()
```

Returns the receiver's JTextArea.

jTextComponent

```
public javax.swing.text.JTextComponent jTextComponent()
```

Returns the receiver's JTextArea.

setEditable

```
public void setEditable(boolean flag)
```

Sets the receiver's editability (by setting its JTextArea's editability).

setOpaque

```
public void setOpaque(boolean flag)
```

Sets whether or not the receiver is opaque (by setting its JTextArea to be opaque or not).

setSize

```
public void setSize(java.awt.Dimension aDimension)
```

```
public void setSize(int width, int height)
```

Sets the size of the receiver's JTextArea to `aDimension` or to `width` and `height`, and then resizes the text area to accommodate the vertical scroll bar.

setText

```
public void setText(String aString)
```

Sets the receiver's text value to `aString` by setting the receiver's JTextArea's text.

EOTextAssociation

Inherits from:	(com.apple.client.eointerface) EOAssociation : EODelayedObserver (EOControl) : Object (com.apple.yellow.eointerface) EOAssociation : EODelayedObserver (EOControl) : NSObject
Implements:	EOObserving (EODelayedObserver) (com.apple.client.eointerface only) java.awt.event.FocusListener (com.apple.client.eointerface only) NSDisposable (EOAssociation)
Package:	com.apple.client.eointerface com.apple.yellow.eointerface

Class Description

In a Java Client application (using com.apple.client.eointerface), 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 a String.

In a com.apple.yellow.eointerface 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

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
(com.apple.client.eointerface) EOTextField, EOTextArea, EOFormCell
(com.apple.yellow.eointerface) NSText, NSTextView

Aspects	
value	A text attribute of the selected object.
(com.apple.yellow.eointerface only) editable	A boolean attribute of the selected object, which determines whether the text object is editable.
(com.apple.client.eointerface only) enabled	A boolean attribute of the selected object, which determines whether the text object is enabled.

Object Keys Taken	
(com.apple.yellow.eointerface only) delegate	An EOTextAssociation accepts delegate messages related to the editing and validation of text; see the NSText and NSTextView class specifications for more information.

Constructors

EOTextAssociation

`public EOTextAssociation(Object aDisplayObject)`

Creates a new EOTextAssociation to monitor and update the value in *aDisplayObject*, which is typically an Application Kit NSActionCell or, in com.apple.client.eointerface applications, an EOFormCell.

You normally set up associations with the Interface Builder application, in which case you don't need to create them programmatically. However, if you do create them up programmatically, setting them up is a multi-step process. After creating an association, you must bind its aspects and establish its connections.

See Also: `bindAspect` (EOAssociation), `establishConnection` **(EOAssociation)**

Instance Methods

breakConnection

`public void breakConnection()`

See the `breakConnection` method description in the superclass EOAssociation.

endEditing

`public void endEditing()`

See the `endEditing` method description in the superclass EOAssociation.

establishConnection

```
public void establishConnection()
```

See the `establishConnection` method description in the superclass EOAssociation.

focusGained

```
public void focusGained(java.awt.event.FocusEvent aFocusEvent)
```

(com.apple.client.eointerface only) EOTextAssociation listens to its display object's focus state changes in order to notify the display group when the user starts editing in the display object. `focusGained` is invoked when the user selected the display object in order to edit its value.

focusLost

```
public void focusLost(java.awt.event.FocusEvent aFocusEvent)
```

(com.apple.client.eointerface only) Invoked when a user leaves the display object, having finished editing its value.

format

```
public java.text.Format format()
```

(com.apple.client.eointerface only) Returns the java.lang.text.Format used to format values bound to the receiver's `ValueAspect` for display and editing.

isUsableWithObject

```
public boolean isUsableWithObject(Object aDisplayObject)
```

(com.apple.client.eointerface only) Returns `true` if *`aDisplayObject`* implements the EOTextAssociation.JTextComponentAccess interface and if its `jTextComponent` is non `null`, `false` otherwise.

See Also: `isUsableWithObject` (EOAssociation)

primaryAspect

```
public String primaryAspect()
```

(com.apple.client.eointerface only) Returns `ValueAspect`.

setFormat

```
public void setFormat(java.text.Format aFormat)
```

(com.apple.client.eointerface only) Sets the java.lang.text.Format object to use in formatting values bound to the receiver's ValueAspect for display and editing.

subjectChanged

```
public void subjectChanged()
```

See the `subjectChanged` method description in the superclass EOAssociation.

EOTextColumnEditor

Inherits from:	EOColumnEditor
Implements:	java.awt.event.ActionListener java.awt.event.FocusListener javax.swing.table.TableCellEditor (EOColumnEditor) javax.swing.CellEditor (javax.swing.table.TableCellEditor)
Package:	com.apple.client.eointerface

Class Description

EOTextColumnEditor is a concrete subclass of EOColumnEditor whose instances mediate between EOTextColumnAssociations and EOTextFields (an EOTextColumnEditor's `editorComponent` is an EOTextField).

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

For more information on the purpose of EOTextColumnEditors, see the EOColumnEditor class specification.

Interfaces Implemented

java.awt.event.ActionListener

`actionPerformed`

java.awt.event.FocusListener

`focusGained`

`focusLost`

Method Types

Instantiation

`createEditorComponent`

Handling events

`actionPerformed`

`beginEditing`

`endEditing`

Accessing the text field

`getCellEditorValue`

`isCellEditable`

`setCellEditorValue`

Instance Methods

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent event)
```

Invokes `stopCellEditing`.

See Also: `stopCellEditing` (EOColumnEditor)

beginEditing

```
protected void beginEditing()
```

Adds the receiver to its editor component as a java.awt.event.FocusListener and as a java.awt.event.ActionListener, and invokes `super`'s implementation.

See Also: `beginEditing` (EOColumnEditor)

createEditorComponent

```
protected abstract java.awt.Component createEditorComponent()
```

Returns a newly instantiated javax.swing.JTextField with a black javax.swing.border.LineBorder.

See Also: `createEditorComponent` (EOColumnEditor)

endEditing

```
protected void endEditing()
```

Removes the receiver from its editor component's focus and action listener lists, and invokes `super`'s implementation.

See Also: `endEditing` (EOColumnEditor)

focusLost

`public void focusLost(java.awt.event.FocusEvent event)`

Invokes `stopCellEditing`.

See Also: `stopCellEditing` (EOColumnEditor)

getCellEditorValue

`public Object getCellEditorValue()`

Overrides `super`'s implementation to return the text value of the receiver's `editorComponent`, an EOTextField.

isCellEditable

`public boolean isCellEditable(java.util.EventObject event)`

Overrides `super`'s implementation to return `true` as long as *`event`* is not a java.awt.event.MouseEvent with a click count of less than two.

setCellEditorValue

`public void setCellEditorValue(Object initialValue)`

Sets the value of the receiver's editor component, an EOTextField by default, to *`initialValue`* using the method `setText`.

EOTextField

Inherits from:	javax.swing.JTextField : javax.swing.JTextComponent : javax.swing.JComponent : java.awt.Container : java.awt.Component Object
Package:	com.apple.client.eointerface

Class Description

EOTextField is a subclass of javax.swing.JTextField that adds the notion of *selectability*.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

When an EOTextField object is selectable, it behaves in every way as a JTextField. However, when an EOTextField is not selectable, its text can't be selected. An EOTextField is selectable by default. To set it so it's not selectable, invoke `setSelectable` with `false`.

Instance Methods

isFocusTraversable

```
public boolean isFocusTraversable()
```

Returns the result of the `super`'s implementation if the receiver is selectable, `false` otherwise.

setSelectable

```
public void setSelectable(boolean flag)
```

Sets the receiver as selectable if *`flag`* is true, or as unselectable otherwise.

EOView

Inherits from:	javax.swing.JPanel : javax.swing.JComponent : java.awt.Container : java.awt.Component Object
Implements:	NSDisposable
Package:	com.apple.client.eointerface

Class Description

EOView is a subclass of javax.swing.JPanel that uses an EOViewLayout object to provide its layout logic.

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

For more information on the layout behavior of EOView objects, see the EOViewLayout class specification.

Interfaces Implemented

NSDisposable

    dispose

Constructors

EOView

`public EOView(String debuggingHint)`

Creates and returns a new EOView object. The *debuggingHint* argument is a string you can use to uniquely identify the view. When the form is instantiated from a nib file, the *debuggingHint* is a string generated by Interface Builder.

See Also: `debuggingHint`

Instance Methods

add

`public java.awt.Component add(java.awt.Component aComponent)`

Adds *aComponent* to the receiver and returns it.

debuggingHint

```
public String debuggingHint()
```

Returns the receiver's debugging hint, a string that uniquely identifies the EOView.

See Also: `EOView` **constructor**

setBounds

```
public void setBounds(int x, int y, int width, int height)
```

See the method description for `setBounds` in Sun's JPanel class documentation.

toString

```
public String toString()
```

Returns the receiver's `debuggingHint`.

EOViewLayout

Inherits from:	Object
Implements:	java.awt.LayoutManager2 java.awt.LayoutManager (java.awt.LayoutManager2) java.io.Serializable
Package:	com.apple.client.eointerface

Class Description

EOViewLayout is an AWT LayoutManager for use in Java Client application (using com.apple.client.eointerface). It implements the geometry options available in Interface Builder'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).

Note: This class doesn't exist in the com.apple.yellow.eointerface package.

Constants

EOViewLayout defines the following `int` constants:

- `MaxXMargin`
- `MinXMargin`
- `MaxYMargin`
- `MinYMargin`
- `WidthSizable`
- `HeightSizable`
- `BothSizable`

For more information on what these constants are and how they're used, see the method description for `setAutosizingMask`.

Constructors

EOViewLayout

```
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:

Constant	Description
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 0 (zero), the default mask, *component*'s adjusted size is a factor of its size when `setAutosizingMask` was invoked.

EODisplayGroup.Delegate

(informal interface)

Package:	com.apple.client.eointerface com.apple.yellow.eointerface

Interface Description

The EODisplayGroup.Delegate interface defines methods that an EODisplayGroup can invoke in its delegate. Delegates are not required to provide implementations for all of the methods in the interface, and you don't have to use the implements keyword to specify that the object implements the Delegates interface. Instead, declare and implement any subset of the methods declared in the interface 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`

`displayGroupDidFetchObjects`

`displayGroupShouldRefetch`

Inserting, updating, and deleting objects

`displayGroupShouldInsertObject`

`displayGroupDidInsertObject`

`displayGroupCreateObjectFailed`

`displayGroupDidSetValueForObject`

`displayGroupShouldDeleteObject`

`displayGroupDidDeleteObject`

Managing the display

`displayGroupShouldDisplayAlert`

`displayGroupShouldRedisplay`

`displayGroupDisplayArrayForObjects`

Managing the selection

`displayGroupShouldChangeSelection`

`displayGroupDidChangeSelection`

`displayGroupDidChangeSelectedObjects`

Changing the data source

`displayGroupDidChangeDataSource`

Instance Methods

displayGroupCreateObjectFailed

```
public abstract void displayGroupCreateObjectFailed(
   EODisplayGroup aDisplayGroup,
   com.apple.yellow.eocontrol.EODataSource aDataSource)
```

Invoked from `insertNewObjectAtIndex` 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

```
public abstract void displayGroupDidChangeDataSource(EODisplayGroup aDisplayGroup)
```

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

displayGroupDidChangeSelectedObjects

```
public abstract 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

```
public abstract void displayGroupDidChangeSelection(EODisplayGroup aDisplayGroup)
```

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

displayGroupDidDeleteObject

```
public abstract void displayGroupDidDeleteObject(
   EODisplayGroup aDisplayGroup,
   Object anObject)
```

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

displayGroupDidFetchObjects

```
public abstract void displayGroupDidFetchObjects(
   EODisplayGroup aDisplayGroup,
   NSArray objects)
```

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

displayGroupDidInsertObject

```
public abstract void displayGroupDidInsertObject(
   EODisplayGroup aDisplayGroup,
   Object anObject)
```

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

displayGroupDidSetValueForObject

```
public abstract void displayGroupDidSetValueForObject(
   EODisplayGroup aDisplayGroup,
   Object value,
   Object anObject,
   String key)
```

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

displayGroupDisplayArrayForObjects

```
public abstract NSArray displayGroupDisplayArrayForObjects(
   EODisplayGroup aDisplayGroup,
   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 EOQualifier's `filteredArrayUsingQualifier` and EOSortOrdering's `sortedArrayUsingKeyOrderArray` methods in EOControl 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`

displayGroupShouldChangeSelection

```
public abstract boolean displayGroupShouldChangeSelection(
   EODisplayGroup aDisplayGroup,
   NSArray newIndexes)
```

Allows the delegate to prevent a change in selection by *`aDisplayGroup`*. *`newIndexes`* is the proposed new selection, an array of Numbers. If the delegate returns `true`, the selection changes; if the delegate returns `false`, the selection remains as it is.

displayGroupShouldDeleteObject

```
public abstract boolean displayGroupShouldDeleteObject(
   EODisplayGroup aDisplayGroup,
   Object anObject)
```

Allows the delegate to prevent *`aDisplayGroup`* from deleting *`anObject`*. If the delegate returns `true`, *`anObject`* is deleted; if the delegate returns `false`, the deletion is abandoned.

displayGroupShouldDisplayAlert

```
public abstract boolean displayGroupShouldDisplayAlert(
   EODisplayGroup aDisplayGroup,
   String title,
   String message)
```

Allows the delegate to prevent *aDisplayGroup* from displaying an attention panel with *title* and *message*. The delegate can return `true` to allow *aDisplayGroup* to display the panel, or `false` to prevent it from doing so (perhaps displaying a different attention panel).

displayGroupShouldFetch

```
public abstract boolean displayGroupShouldFetch(EODisplayGroup aDisplayGroup)
```

Allows the delegate to prevent *aDisplayGroup* from fetching. If the delegate returns `true`, *aDisplayGroup* performs the fetch; if the delegate returns `false`, *aDisplayGroup* abandons the fetch.

displayGroupShouldInsertObject

```
public abstract boolean displayGroupShouldInsertObject(
   EODisplayGroup aDisplayGroup,
   Object anObject,
   int anIndex)
```

Allows the delegate to prevent *aDisplayGroup* from inserting *anObject* at *anIndex*. If the delegate returns `true`, *anObject* is inserted; if the delegate returns `false`, the insertion is abandoned.

displayGroupShouldRedisplay

```
public abstract boolean displayGroupShouldRedisplay(
   EODisplayGroup aDisplayGroup,
   NSNotification aNotification)
```

Invoked whenever *aDisplayGroup* receives an `ObjectsChangedInEditingContextNotification`, this method allows the delegate to suppress redisplay based on the nature of the change that has occurred. If the delegate returns `true`, *aDisplayGroup* redisplays; if it returns `false`, *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 `ObjectsChangedInEditingContextNotification`.

See Also: `redisplay`

displayGroupShouldRefetch

```
public abstract boolean displayGroupShouldRefetch(
   EODisplayGroup aDisplayGroup,
   NSNotification aNotification)
```

Invoked whenever *aDisplayGroup* receives an `InvalidatedAllObjectsInStoreNotification`, this method allows the delegate to suppress refetching of the invalidated objects. If the delegate returns `true`, *aDisplayGroup* immediately refetches its objects. If the delegate returns `false`, *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.

EOTextAssociation.JTextComponentAccess

Package: com.apple.client.eointerface

Interface Description

EOTextAssociation.JTextComponentAccess is an interface that specifies the way an EOTextAssociation accesses its display object's underlying javax.swing.text.JTextComponent.

Note: This interface doesn't exist in the com.apple.yellow.eointerface package.

Instance Methods

jTextComponent

```
public abstract javax.swing.text.JTextComponent jTextComponent()
```

Returns the receiver's JTextComponent.

EOTableColumnAssociation.TableColumnCustomizer

Package: com.apple.client.eointerface

Interface Description

EOTableColumnAssociation.TableColumnCustomizer is an interface the API an object uses to specify custom editors and renderers for an EOTableColumnAssociation.

Note: This interface doesn't exist in the com.apple.yellow.eointerface package.

To use your own editor or renderer in the JTable of an EOTable, you define a class that implements EOTableColumnAssociation.TableColumnCustomizer's two methods: `editorForAssociation`, which should return an editor for the specified association, and `rendererForAssociation`, which should return a renderer for the specified association. Register an instance of your TableColumnCustomizer using EOTableColumnAssociation's static method `setTableColumnCustomizer`.

For more information on how TableColumnCustomizers are used, see the EOTableColumnAssociation class specification.

Instance Methods

editorForAssociation

```
public abstract EOColumnEditor
   editorForAssociation(EOTableColumnAssociation tableColumnAssociation)
```

Returns the EOColumnEditor to be used for *tableColumnAssociation*'s display object (a javax.swing.table.TableColumn).

rendererForAssociation

```
public abstract javax.swing.table.TableCellRenderer
   rendererForAssociation(EOTableColumnAssociation tableColumnAssociation)
```

Returns the TableCellRenderer to be used for *tableColumnAssociation*'s display object (a javax.swing.table.TableColumn).

Deprecated API

This file enumerates those EOInterface classes and methods that have been deprecated and should no longer be used. Wherever possible, notes have been included to indicate what API should be used in place of the deprecated class or method.

EOTableAssociation

isEditableAtRow

```
public boolean isEditableAtRow(int row)
```

Returns whether or not the display object bound to the receiver is editable at *row* as determined by the `EnabledAspect`. If this aspect is bound, a non-zero value at *row* indicates that the property can be edited. If the `EnabledAspect` is unbound all rows are considered editable.

T H E   A P P L E   P U B L I S H I N G   S Y S T E M

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software.

Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.