EOFault

Inherits From: none *(EOFault is a root class)*

Declared In: EOControl/EOFault.h

Class Description

EOFault and EOFaultHandler form a general mechanism for substituting placeholder objects that convert themselves into regular objects. An EOFault is most commonly used by the Access Layer to represent an object not yet fetched from the database, but that must nonetheless exist as an instance in the application—typically because it's the destination of a relationship. EOFault is a completely general class; there's no need to create subclasses to customize fault handling. Instead, you create subclasses of EOFaultHandler to accommodate different means of converting faults into regular objects.

The faulting mechanism provides for continuity of an object's **id** even when that object's state isn't yet available. An EOFault simply holds the place for an ultimate "real" object, handling all methods that it can without causing the state to be loaded. When an EOFault receives a message that it can't handle, it calls upon its EOFaultHandler to *fire* it, converting it into a "real" object. This often involves accessing the external, persistent state of the object.

Creating an EOFault

Rather than allocating and initializing an EOFault, you turn an existing object into one using EOFault's **makeObjectIntoFault:withHandler:** class method. When you do so, you must provide an EOFaultHandler that will later help the fault to fire. **makeObjectIntoFault:withHandler:** preserves the **id** of the original object, overlaying its **isa** pointer with that of the EOFault class and slipping the EOFaultHandler among its instance variables. Once this is done, the original object is an EOFault that will fire when accessed.

The EOFaultHandler should be considered completely private property of the EOFault once you've created it. You should neither retain the EOFaultHandler or send it any other messages, instead dealing exclusively with the newly created EOFault or the EOFault class itself.

EOFault Behavior

EOFault implements many basic object methods in a manner that doesn't cause the receiver to fire. The following methods all behave as though normal for the original object:

- retain
- release
- autorelease
- retainCount
- class
- superclass
- isKindOfClass:
- isMemberOfClass:
- conformsToProtocol:
- isProxy
- methodSignatureForSelector:
- respondsToSelector:
- zone
- doesNotRecognizeSelector:

doesNotRecognizeSelector: is a special case here, in that it's only invoked if the selector in question isn't found for the original class. Normally, methods not implemented by EOFault, but implemented by the original class, cause the receiver to fire as described below.

These methods don't cause the receiver to fire, but also don't hide the presence of the EOFault class:

- description
- descriptionWithIndent:
- eoDescription
- descriptionWithLocale:
- descriptionWithLocale:indent:
- eoShallowDescription

The following common methods, along with any others not explicitly mentioned in this section, do cause the receiving EOFault to fire.

- dealloc
- self
- forwardInvocation:

When an EOFault receives one of these messages, it fires in one of a few different ways. **dealloc** invokes the **clearFault:** class method to revert the receiver back to its original state, then reinvokes **dealloc** to clean up instance variables and deallocate the object. The other methods all send a special message, **completeInitializationOfObject:**, to the EOFaultHandler to transform the EOFault into a regular object, possibly different from its original state. In addition, **forwardInvocation:** sends a **shouldPerformInvocation:** to the EOFaultHandler first, which allows it to perform the method itself without causing the EOFault to be transformed. If the EOFaultHandler returns YES, though, the EOFault then sends it a **completeInitializationOfObject:** message.

Examining an EOFault

Three additional EOFault methods allow you to explicitly check whether an object is an EOFault without causing it to fire, and to get its original class and EOFaultHandler if it is an EOFault. These methods are:

+ isFault:
+ targetClassForFault:
+ handlerForFault:

You can use these methods to base some decisions on whether an object is an EOFault, though you should rarely need to do so.

Method Types

Creating and examining faults	+ makeObjectIntoFault:withHandler: + isFault: + clearFault: + handlerForFault: + targetClassForFault: + respondsToSelector:
Checking class information	– class – isKindOfClass: – isMemberOfClass: – respondsToSelector: – conformsToProtocol: – methodSignatureForSelector:
Run-time support	– forwardInvocation: – doesNotRecognizeSelector:
Getting a fault’s description	– description – descriptionWithIndent: – descriptionWithLocale: – descriptionWithLocale:indent: – eoDescription – eoShallowDescription
Reference-counting	– retain – release – retainCount – autorelease – dealloc
Miscellaneous object methods	– self – isProxy – superclass – zone

Class Methods

clearFault:

+ (void)**clearFault:**(id)*aFault*

Restores *aFault* to its status prior to the **makeObjectInfoFault:withHandler:** message that created it. Raises an NSInvalidArgumentException if *aFault* isn't an EOFault.

You rarely use this method. Faults typically fire automatically when accessed, using EOFaultHandler's **completeInitializationOfObject:** method. See the EOFaultHandler class specification for more information.

handlerForFault:

+ (EOFaultHandler *)**handlerForFault:**(id)*aFault*

Returns the EOFaultHandler that will help *aFault* to fire. Returns **nil** if *aFault* isn't an EOFault.

isFault:

+ (BOOL)**isFault:**(id)*anObject*

Returns YES if *anObject* is an EOFault, NO otherwise.

makeObjectIntoFault:withHandler:

+ (void)**makeObjectIntoFault:**(id)*anObject* **withHandler:**(EOFaultHandler *)*aFaultHandler*

Converts *anObject* into an EOFault, assigning *aFaultHandler* as the object that stores its original state and later converts the EOFault back into a normal object (typically by fetching data from an external repository). The new EOFault becomes the owner of *aFaultHandler*; you shouldn't assign it to another object.

respondsToSelector:

+ (BOOL)**respondsToSelector:**(SEL)*aSelector*

Returns YES if the receiving class responds to *aSelector*, NO otherwise.

targetClassForFault:

+ (Class)**targetClassForFault:**(id)*aFault*

Returns the original class of the object that was turned into *aFault*, or **nil** if *aFault* isn't an EOFault. When the EOFault fires, it's guaranteed to be an instance of this class or possibly of a subclass. To get the actual class, you must send a **class** message to the EOFault, which may fire to determine its actual class membership.

Instance Methods

autorelease

– (id)**autorelease**

Performs as NSObject's **autorelease** method.

class

– (Class)**class**

Returns the class of the object that the receiving EOFault will become. This may cause the EOFault to fire in order to determine its actual class membership.

See also: – **classForFault:** (EOFaultHandler), + **targetClassForFault:**

conformsToProtocol:

– (BOOL)**conformsToProtocol:**(Protocol *)*aProtocol*

Returns YES if the object that the receiving EOFault will become conforms to *aProtocol*, NO if it doesn't. This may cause the EOFault to fire in order to determine its actual class membership.

See also: – **conformsToProtocol:forFault:** (EOFaultHandler)

dealloc

– (void)**dealloc**

Invokes the **clearFault:** class method to revert the receiving EOFault to its original class membership and state, then reinvokes **dealloc**.

description

– (NSString *)**description**

Sends **descriptionForObject:** to the receiver's EOFaultHandler and returns the result.

descriptionWithIndent:

– (NSString *)**descriptionWithIndent:**(unsigned int)*indentLevel*

Invokes **description** and returns the result.

descriptionWithLocale:

– (NSString *)**descriptionWithLocale:**(NSDictionary *)*locale*

Invokes **description** and returns the result.

descriptionWithLocale:indent:

– (NSString *)**descriptionWithLocale:**(NSDictionary *)*locale* **indent:**(unsigned)*indentLevel*

Invokes **description** and returns the result.

doesNotRecognizeSelector:

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

Raises an NSInvalidArgumentException.

eoDescription

– (NSString *)**eoDescription**

Invokes **description** and returns the result.

See also: – **eoDescription** (NSObject Additions)

eoShallowDescription

– (NSString *)**eoShallowDescription**

Invokes **description** and returns the result.

See also: – **eoShallowDescription** (NSObject Additions)

forwardInvocation:

– (void)**forwardInvocation:**(NSInvocation *)*anInvocation*

Causes the receiving EOFault to fire, if allowed by its EOFaultHandler, and forward *anInvocation* to its new incarnation. Sends a **shouldPerformInvocation:** to the receiver's EOFaultHandler first, giving it a chance to bypass the conversion. If the EOFaultHandler returns NO, returns immediately. If it returns YES, sends a **completeInitializationOfObject:** message to the EOFaultHandler with **self** as the argument. Once the receiver has fired it invokes *anInvocation*.

isKindOfClass:

– (BOOL)**isKindOfClass:**(Class)*aClass*

Returns YES if *aClass* is the class, or a superclass, of the object that the receiving EOFault will become, NO otherwise. This may cause the EOFault to fire in order to determine its actual class membership.

See also: – **isMemberOfClass:**, – **isKindOfClass:forFault:** (EOFaultHandler)

isMemberOfClass:

– (BOOL)**isMemberOfClass:**(Class)*aClass*

Returns YES if *aClass* is the class of the object that the receiving EOFault will become, NO otherwise. This may cause the EOFault to fire in order to determine its actual class membership.

See also: – **isKindOfClass:**, – **isMemberOfClass:forFault:** (EOFaultHandler)

isProxy

– (BOOL)**isProxy**

Returns NO.

methodSignatureForSelector:

– (NSMethodSignature *)**methodSignatureForSelector:**(SEL)*aSelector*

Returns a method signature for *aSelector* for the object that the receiving EOFault will become, or **nil** if one can't be found. This may cause the EOFault to fire in order to determine its actual class membership.

See also: – **methodSignatureForSelector:forFault:** (EOFaultHandler)

release

– (void)**release**

Performs as NSObject's **release** method.

respondsToSelector:

– (BOOL)**respondsToSelector:**(SEL)*aSelector*

Returns YES if the object that the receiving EOFault will become responds to *aSelector*, NO otherwise. This may cause the EOFault to fire in order to determine its actual class membership.

See also: – **respondsToSelector:forFault:** (EOFaultHandler)

retain

– (id)**retain**

Performs as NSObject's **retain** method.

retainCount

– (unsigned int)**retainCount**

Performs as NSObject's **retainCount** method.

self

– (id)**self**

Fires the receiver and returns **self**. This is the recommended way to simply fire an EOFault.

superclass

– (Class)**superclass**

Returns the superclass of the object that the receiving EOFault will become. This may cause the EOFault to fire in order to determine its actual class membership.

See also: – **classForFault:** (EOFaultHandler)

zone

– (NSZone *)**zone**

Performs as NSObject's **zone** method.