IBDocument Class Reference

Tools & Languages: IDEs

2009-03-24

Contents

IBDocument Class Reference

Inherits from	NSDocument : NSObject
Conforms to	`NSObject` (NSObject)
Framework	/Developer/Library/Frameworks/InterfaceBuilderKit.framework
Declared in	InterfaceBuilderKit/IBDocument.h
Companion guide	Interface Builder Plug-In Programming Guide

Overview

An `IBDocument` object provides the in-memory representation of a nib file and is one of the core classes in Interface Builder. It manages all of the objects in a nib file, observes them for undo support, and facilitates the general exchange of information with other objects. You use this object as is and do not subclass it.

A document object owns all of the objects in a nib file, including the windows, views, controls, controller objects, and custom objects in that nib file. You can use the methods of `IBDocument` to access these objects, add and remove objects, or change the inter-object relationships.

Most Interface Builder plug-ins should never need to access the current `IBDocument` object. The only times you might use this object are when you need to make additional changes to the object hierarchy beyond those provided automatically by Interface Builder. For example, you could use the document object to associate additional objects with a view as part of its design-time configuration.

Top-Level Objects

Each document contains a distinct set of top-level objects (also known as the "root objects"). These are the objects that appear at the top level of the document window and often consist of windows, menus, and custom controller objects. Placeholder objects such as File's Owner, First Responder, and Application are also top-level objects. (Placeholder objects are programmatic entities provided by Interface Builder.)

The user can add other top-level objects to a nib file by dragging items from the library window to the document window. You can add top-level objects programmatically using the methods of this class and specifying a parent object of `nil`.

First Class Objects

Interface Builder documents distinguish between objects that can be edited and selected and those that cannot. If an object can be edited and selected, it is considered to be a "first-class" object and is exposed to the user through Interface Builder's outline view. All other objects are "second-class", are controlled by an owning first-class object, and cannot be manipulated directly by the user in Interface Builder. Your plug-in can modify the second-class objects for which it is responsible. You should not attempt to modify second-class objects managed by other plug-ins, however.

All objects in a document, regardless of whether they are first class or second class, are stored in the same nib file. The distinction between first and second-class objects only affects access to the object in the Interface Builder windows and user interface.

Scroll views provide a perfect example of how first and second-class objects may be related in a nib file. A scroll view exposes its document view and scroller objects as first-class objects but does not expose its clip view. The clip view is considered to be an integral part of the scroll view and is therefore a second-class object to be hidden from the user. When the nib file is loaded, however, the clip view is instantiated along with the scroll view, document view, and scrollers.

Tasks

Getting the Document Object

`+ documentForObject:` (page 7)
Returns the document object that currently owns the specified object.

Getting the Objects in a Document

`– objects` (page 11)
Returns all of the first-class objects in the receiver.

`– topLevelObjects` (page 13)
Returns the top-level objects in the receiver.

`– childrenOfObject:` (page 8)
Returns an array containing the first-class children of the specified object.

`– parentOfObject:` (page 11)
Returns the first-class parent of the specified object.

`– removeObject:` (page 12)
Removes the specified object and its children from the document.

`– addObject:toParent:` (page 7)
Adds the specified object to the document as a first-class object.

`– moveObject:toParent:` (page 10)
Changes the parent of the specified object.

Getting the Object's Attributes

`- documentImageNamed:` (page 10)
: Returns the image resource with the specified name.

`- nameForDocumentImage:` (page 11)
: Returns the name of a specified image resource.

`- setMetadata:forKey:ofObject:` (page 12)
: Associates a custom value with the specified key of an object.

`- metadataForKey:ofObject:` (page 10)
: Returns the value associated with the specified key and object.

Creating Connections

`- connectOutlet:ofSourceObject:toDestinationObject:` (page 9)
: Creates an outlet connection between two objects.

`- connectAction:ofSourceObject:toDestinationObject:` (page 8)
: Creates an action connection between two objects.

`- connectBinding:ofSourceObject:toDestinationObject:keyPath:options:` (page 9)
: Creates a binding between two objects.

Class Methods

documentForObject:

Returns the document object that currently owns the specified object.

```
+ (id)documentForObject:(id)object
```

Parameters

object
: The object whose document you want.

Return Value
The document object that owns the specified object, or `nil` if the object has no owner. Objects that are in transition (such as on the pasteboard) are not owned by a document.

Instance Methods

addObject:toParent:

Adds the specified object to the document as a first-class object.

```
- (void)addObject:(id)object toParent:(id)parent
```

Parameters

`object`
: The object to add to the document.

`parent`
: The parent of *`object`*, or `nil` if *`object`* should be added to the document as a top-level object. The parent object must be a first-class object of the document.

Discussion

This method adds the object to the document's internal data structures as a child of *`parent`*. This method does not create any object-specific associations between *`object`* and *`parent`*. For example, if both *`parent`* and *`object`* are views, this method does not configure *`parent`* as the superview of *`object`*. You must create any supplemental connections yourself.

See Also

`– removeObject:` (page 12)

childrenOfObject:

Returns an array containing the first-class children of the specified object.

```
- (NSArray *)childrenOfObject:(id)object
```

Parameters

`object`
: The parent object whose children you want.

Return Value

An array of objects, each of which is a first-class object of *`object`*. If *`object`* has no children, this method returns an empty array object.

See Also

`– parentOfObject:` (page 11)

connectAction:ofSourceObject:toDestinationObject:

Creates an action connection between two objects.

```
- (void)connectAction:(NSString
    *)actionofSourceObject:(id)sourcetoDestinationObject:(id)destination
```

Parameters

`action`
: The name of the action message. This string must correspond to a selector in the destination object and the selector name must end with a trailing colon character.

`source`
: The object that receives the specified action message, otherwise known as the target of the action. This object should contain a selector whose signature matches the string in *`action`*.

`destination`
: The object that triggers the action and sends the action message. This is typically a control, such as a button.

Discussion
Although the behavior of the *source* and *destination* parameters might seem reversed, they are not. The parameter names correspond to the order in which the action connection is established inside the plug-in, rather than the order in which action messages flow. In Interface Builder, action connections can be created starting at either end of the connection. For plug-ins, it is often more convenient to start at the object containing the action method and connect that method to the object that sends the action message.

connectBinding:ofSourceObject:toDestinationObject:keyPath:options:

Creates a binding between two objects.

```
- (void)connectBinding:(NSString
   *)bindingNameofSourceObject:(id)sourcetoDestinationObject:(id)destinationkeyPath:(NSString
    *)keyPathoptions:(NSDictionary *)options
```

Parameters

bindingName
: The key path that identifies a property of the source object. The specified property must be bindable.

source
: The source object that owns the property identified by the *bindingName* parameter.

destination
: The object that provides data for the specified binding.

keyPath
: The key path in the destination object that points to the data being bound.

options
: A dictionary containing options for the binding, such as placeholder objects or an `NSValueTransformer` identifier. This value is optional—pass `nil` to specify no options. For information about the keys you can place in the dictionary, see the binding options constants in *NSKeyValueBindingCreation Protocol Reference*.

Discussion
For information about how to make your own custom objects bindable, see *Cocoa Bindings Programming Topics*.

connectOutlet:ofSourceObject:toDestinationObject:

Creates an outlet connection between two objects.

```
- (void)connectOutlet:(NSString
   *)outletofSourceObject:(id)sourcetoDestinationObject:(id)destination
```

Parameters

outlet
: The name of the outlet.

source
: The source object that contains the specified outlet. The specified outlet must exist and have the specified name.

destination
: The object to assign to the specified outlet.

documentImageNamed:

Returns the image resource with the specified name.

```
- (NSImage *)documentImageNamed:(NSString *)name
```

Parameters

name

The name of the desired image. The image must be a resource in the receiving document. This parameter must not be `nil`.

Return Value

The image resource with the specified name, or a default "missing image" resource if an image with the specified name could not be found. If *name* contains an empty string, this method returns `nil`.

See Also

- `nameForDocumentImage:` (page 11)

metadataForKey:ofObject:

Returns the value associated with the specified key and object.

```
- (id)metadataForKey:(NSString *)keyofObject:(id)object
```

Parameters

key

The key used to identify the metadata value.

object

The first-class object containing the metadata value.

Return Value

The current value associated with the specified key.

See Also

- `setMetadata:forKey:ofObject:` (page 12)

moveObject:toParent:

Changes the parent of the specified object.

```
- (void)moveObject:(id)objecttoParent:(id)parent
```

Parameters

object

The object whose parent you want to change. This object must be a first-class object.

parent

The new parent for *object*. This object must be a first-class object. Specify `nil` to make *object* a top-level object of the document.

Discussion
This method changes the parentage of the object in the document's internal data structures. This method does not make assumptions about the type of *`object`* and therefore does change any other associations between it and its new or former parent. For example, if *`object`* is a view, this method does not change the superview of *`object`* to *`parent`*. You must make such a change yourself.

nameForDocumentImage:

Returns the name of a specified image resource.

```
- (NSString *)nameForDocumentImage:(NSImage *)image
```

Parameters

`image`
The image resource whose name you want to obtain. This parameter must not be `nil`.

Return Value
The name of the specified image, or `nil` if the image is unnamed or is not a resource in the receiving document.

Discussion
This method returns the name of an image resource obtained through Interface Builder's synchronization mechanism. The name is suitable for exposing to the user in an `IBInspector` subclass.

See Also
`- documentImageNamed:` (page 10)

objects

Returns all of the first-class objects in the receiver.

```
- (NSArray *)objects
```

Return Value
An array containing all of the nib file's first-class objects.

Discussion
For information about what comprises a first-class object, see "First Class Objects" (page 6).

See Also
`- topLevelObjects` (page 13)

parentOfObject:

Returns the first-class parent of the specified object.

```
- (id)parentOfObject:(id)object
```

Parameters

`object`
The object whose parent you want. This object must be a first-class object of the document.

Return Value
The closest parent to *`object`* that is also a first-class object, or `nil` if *`object`* is a root object.

See Also
`- childrenOfObject:` (page 8)

removeObject:

Removes the specified object and its children from the document.

```
- (void)removeObject:(id)object
```

Parameters

`object`
The object to remove.

Discussion
During the removal process, this method makes no assumptions about the type of each removed object. As a result, this method removes the object only from the document's own internal data structures. This includes breaking any outlet, action, or binding connections between *`object`* and any other objects in the document. It does not include breaking relationships that are part of the object's own behavior. For example, this method does not disassociate a view from its superview. You must break any object-specific relationships yourself either before or after removing the object from the document.

See Also
`- addObject:toParent:` (page 7)

setMetadata:forKey:ofObject:

Associates a custom value with the specified key of an object.

```
- (void)setMetadata:(id)valueforKey:(NSString *)keyofObject:(id)object
```

Parameters

`value`
The value you want to set, or `nil` if you want to clear the current value. The value should be of a type supported by property lists, such as `NSString`, `NSNumber`, `NSArray`, `NSDictionary`, `NSData`, and so on. For more information, see *Property List Programming Guide*.

`key`
The key used to identify the value.

`object`
The first-class object on which to set the value.

Discussion
You can associate custom metadata values with the first-class objects of a document. The values you add are saved with the object in the nib file and persist across undo and pasteboard operations. Setting a metadata value on an object is not an undoable action itself, however.

If a metadata entry with the same key already exists in *`object`*, this method replaces the old value with the value in the *`property`* parameter.

See Also
`- metadataForKey:ofObject:` (page 10)

topLevelObjects

Returns the top-level objects in the receiver.

```
- (NSArray *)topLevelObjects
```

Return Value
An array of the nib file's top-level objects.

Discussion
This method returns all of the top-level objects in a document, including user-created objects (such as windows and menus) that have no parent object of their own, and placeholder objects such as File's Owner, First Responder, and Application. For more information, see "Top-Level Objects" (page 5).

See Also
`- objects` (page 11)

Document Revision History

This table describes the changes to *IBDocument Class Reference*.

Date	Notes
2009-03-24	Updated for Xcode 3.2.
2009-01-06	Made minor technical corrections.
2007-04-02	New document describing the methods for manipulating an Interface Builder document.