NSObject Interface Builder Kit Additions Reference

Tools > Interface Builder

2007-05-10

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

NSObject Interface Builder Kit Additions Reference

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

Overview

The Interface Builder Kit framework extends the `NSObject` class by adding methods to discover design-time information about the object, such as its attributes and child objects. Additional methods provide Interface Builder with the classes used to inspect the object.

Subclassing Notes

You do not call the methods of this class directly. Instead, you override them to provide Interface Builder with information about your custom objects. Although all of the methods have a default implementation that you can use, many need to be overridden to provide relevant information about the receiver. Which methods you override is determined by the capabilities of your custom objects.

If your custom object is a subclass of `NSView`, you still probably need to override methods of this category to provide relevant object-level information. In addition, you may also need to override methods of the `NSView` category, which is described in *NSView Interface Builder Kit Additions Reference*.

Because these methods are relevant only in the context of Interface Builder, it is recommended that you implement these methods in a category on your class and include the category code only in your Interface Builder plug-in.

Tasks

Providing the Designable Attributes of an Object

`- ibPopulateKeyPaths:` (page 11)
Returns a set of key paths identifying the receiver's designable attributes.

Specifying the Inspectors of an Object

- `ibPopulateAttributeInspectorClasses:` (page 10)
  Returns the inspector classes used to edit the receiver's attributes.

Getting the Object's Name and Icon

- `ibDefaultLabel` (page 8)
  Returns the display name of the receiver.
- `ibDefaultImage` (page 7)
  Returns the icon of the receiver.

Identifying Child Objects in the View Hierarchy

- `ibObjectAtLocation:inWindowController:` (page 10)
  Returns the child object of the receiver that is located at the specified point in the window.

Specifying Child Object Information

- `ibDefaultChildren` (page 7)
  Returns the array of objects that should be exposed as children of the receiver.
- `ibRectForChild:inWindowController:` (page 11)
  Returns the visible frame of the receiver's child objects.
- `ibIsChildInitiallySelectable:` (page 9)
  Returns a Boolean value indicating whether the specified child object should be selected before its parent.
- `ibIsChildViewUserMovable:` (page 9)
  Returns a Boolean value indicating whether the location of the specified child view can be changed by the user.
- `ibIsChildViewUserSizable:` (page 9)
  Returns a Boolean value indicating whether the size of the specified child view can be changed by the user.

Document-related Notifications

- `ibDidAddToDesignableDocument:` (page 8)
  Notifies the receiver that it was added to the specified document.
- `ibDidRemoveFromDesignableDocument:` (page 8)
  Notifies the receiver that it was removed from the specified document.
- `ibAwakeInDesignableDocument:` (page 7)
  Notifies the receiver that it is about to be loaded into Interface Builder for the first time.

Instance Methods

ibAwakeInDesignableDocument:

Notifies the receiver that it is about to be loaded into Interface Builder for the first time.

```
- (void)ibAwakeInDesignableDocument:(IBDocument *)document
```

Parameters

document

The document into which the receiver was added.

Discussion

This method is called once when the object is first instantiated and added to a document in Interface Builder. You can use this method to perform any one-time configuration of the object.

See Also

- ibDidAddToDesignableDocument: (page 8)

ibDefaultChildren

Returns the array of objects that should be exposed as children of the receiver.

```
- (NSArray *)ibDefaultChildren
```

Return Value

The array of objects you want to expose.

Discussion

Interface Builder assumes that only the receiver is a first-class object—that is, only the receiver can be edited and manipulated by the user. However, if the receiver's associated objects also have editable attributes, you can expose those objects using this method. To do so, you simply override this method and return an array containing the objects you want to expose. Objects exposed using this method are also considered first-class objects. (For a detailed description of what constitutes a first-class object, see *IBDocument Class Reference*.) This method is typically used to expose subviews in a view hierarchy but may be used to expose non-view objects.

In your implementation of this method, you should return only the immediate child objects of the receiver you want to expose. Interface Builder calls this method recursively on the objects you return to allow them to expose their own children. If you do not want to expose a child view, but do want to expose a grandchild view, you may return the grandchild view as if it belonged to the receiver. Essentially, a view should be exposed only by one ancestor.

ibDefaultImage

Returns the icon of the receiver.

```
- (NSImage *)ibDefaultImage
```

Return Value

An image object containing the receiver's icon.

ibDefaultLabel

Returns the display name of the receiver.

```
- (NSString *)ibDefaultLabel
```

Return Value
A string containing the receiver's user-presentable name.

Discussion
You can override this method to return a custom name for the receiver. The default implementation of this method uses the following rules to return a name based on the receiver's class name:

1. Leading capital letters are stripped from the class name, except for the last one.
2. Spaces are inserted whenever the case changes.
3. The leading letter of each secondary word is then changed to lower case.

For example, the class name `NSTabViewItem` becomes the string "Tab view item".

ibDidAddToDesignableDocument:

Notifies the receiver that it was added to the specified document.

```
- (void)ibDidAddToDesignableDocument:(IBDocument *)document
```

Parameters
document
The document to which the receiver was added.

Discussion
You can use this method to perform any per-document configuration of the receiver. If you only need to configure an object once when it is first instantiated, override the `ibAwakeInDesignableDocument:` method instead.

See Also
- `ibDidRemoveFromDesignableDocument:` (page 8)
- `ibAwakeInDesignableDocument:` (page 7)

ibDidRemoveFromDesignableDocument:

Notifies the receiver that it was removed from the specified document.

```
- (void)ibDidRemoveFromDesignableDocument:(IBDocument *)document
```

Parameters
document
The document from which the receiver was removed.

Discussion
You can use this method to undo any per-document configuration you performed in the `ibDidAddToDesignableDocument:` method.

See Also
- `ibDidAddToDesignableDocument:` (page 8)

ibIsChildInitiallySelectable:

Returns a Boolean value indicating whether the specified child object should be selected before its parent.

```
- (BOOL)ibIsChildInitiallySelectable:(id)child
```

Parameters

child

A child object of the receiver.

Return Value

`YES` if an initial click on the specified child object results in the selection of the child object instead of the parent; otherwise, `NO` if the parent should be selected before its child. This method returns `YES` by default.

ibIsChildViewUserMovable:

Returns a Boolean value indicating whether the location of the specified child view can be changed by the user.

```
- (BOOL)ibIsChildViewUserMovable:(NSView *)view
```

Parameters

view

A child view of the receiver.

Return Value

`YES` if the location of the child view is user modifiable ; otherwise, `NO`.

Discussion

In most cases, you should not need to override this method. The default implementation returns an appropriate value based on the type of the receiver and the corresponding child view. The only time you might ever need to override this method is if the receiver is a container view that limits the movement of its child views. For example, a scroll view pins its scrollers at particular locations within its frame and therefore returns `NO`.

ibIsChildViewUserSizable:

Returns a Boolean value indicating whether the size of the specified child view can be changed by the user.

```
- (BOOL)ibIsChildViewUserSizable:(NSView *)child
```

Parameters

child

A child view of the receiver.

Return Value

`YES` if the size of the child is user modifiable; otherwise, `NO`.

Discussion
In most cases, you should not need to override this method. The default implementation returns an appropriate value based on the type of the receiver and the corresponding child view. The only time you might ever need to override this method is if the receiver is a container view that limits the sizing behavior of its child views. For example, a scroll view does not allow its scrollers to be resized and therefore returns `NO`.

ibObjectAtLocation:inWindowController:

Returns the child object of the receiver that is located at the specified point in the window.

```
- (id)ibObjectAtLocation:(NSPoint)windowPoint inWindowController:(NSWindowController
    *)controller
```

Parameters

point

The point, in the window's coordinate space, to test for a child object.

controller

The window controller object that manages the window of the receiver.

Return Value
The object at the specified point, or `nil` if the point is not in the receiver or in any of its child objects.

Discussion
Interface Builder calls this method recursively to locate objects in a window. The default implementation of this method walks the view hierarchy to find objects and is sufficient for most custom objects. You should override this method only if your object has visible child objects that lie outside of the receiver's own frame.

ibPopulateAttributeInspectorClasses:

Returns the inspector classes used to edit the receiver's attributes.

```
- (void)ibPopulateAttributeInspectorClasses:(NSMutableArray *)classes
```

Parameters

classes

On input, a mutable array. On output, this array should contain the stack of inspector classes to use to inspect the receiver.

Discussion
You typically override this method for each custom object or view provided by your plug-in. Your implementation of this method should call `super` first to retrieve the inherited inspectors for your class. You should then add any custom inspector classes to the array in the order that you want the corresponding slices to appear in the inspector window. The last inspector added to the *classes* array appears at the top of the inspector window, with the other inspectors situated below it.

Each inspector class you add must be a subclass of the `IBInspector` class. The inspectors you add appear under the attributes inspector mode of the inspector window.

A fairly typical implementation of this method would look like the following:

```
- (void)ibPopulateAttributeInspectorClasses:(NSMutableArray *)classes
{
    [super ibPopulateAttributeInspectorClasses:classes];
```

```
    [classes addObject:[MyInspectorClass class]];
}
```

ibPopulateKeyPaths:

Returns a set of key paths identifying the receiver's designable attributes.

```
- (void)ibPopulateKeyPaths:(NSMutableDictionary *)keyPaths
```

Parameters

keyPaths

A mutable dictionary into which you can place the receiver's key-value observable key paths.

Discussion

This method lets you return a set of key paths that identify the attributes of your objects. Interface Builder monitors these key paths and uses them to provide automatic support for undo operations and for the application's "lift-and-stamp" feature, which copies attributes between different instances of your object. The objects whose attributes you are exposing must be key-value observing (KVO) compliant. For information on how to support key-value observing, see *Key-Value Observing Programming Guide*.

The `keyPaths` dictionary contains each of the keys described in "`Key paths dictionary keys`" (page 12). Each key is associated with an `NSMutableSet` object to which you can add your own key path strings. The following is a list of the types of key paths that should go into each set:

- Add the key paths to your object's scalar attribute values (such as strings, booleans, and integers) to the `IBAttributeKeyPaths` set. Scalar attribute values include values represented by classes such as `NSString`, `NSNumber`, `NSColor`, `NSDate`, `NSValue`, and so on.
- Add the key paths to your object's to-one relation attributes to the `IBToOneRelationshipKeyPaths` set. To-one relations represent connections to other objects that have their own attributes. For example, controls typically have a to-one relation to their cell object.
- Add the key paths to your object's to-many relation attributes to the `IBToManyRelationshipKeyPaths` set. An example of an attribute with a to-many relation is one that contains a collection object.
- For any scalar attributes that contain localizable strings, add the corresponding key paths to the `IBLocalizableStringKeyPaths` set. This group is typically a subset of your object's scalar attributes.
- For any scalar attributes that contain geometry-based information, add the corresponding key paths to the `IBLocalizableGeometryKeyPaths` set. Geometry based attributes might include integers or floating-point values that represent the size or position of the object or its contents. Geometry information can change during the localization process and Interface Builder uses this information to record all relevant changes.
- For attributes that are localizable but are not string or geometry-based values, add the corresponding key paths to the `IBAdditionalLocalizableKeyPaths` set. For example, you could use this set to store the key paths to attributes containing images whose content might require localization.

Before adding the key paths associated with the receiver's class, call the `super` implementation to add the attributes of any superclasses.

ibRectForChild:inWindowController:

Returns the visible frame of the receiver's child objects.

```
- (NSRect)ibRectForChild:(id)child inWindowController:(NSWindowController
    *)controller
```

Parameters

child

The child object of the receiver whose visible frame should be returned. This object may not necessarily be a view. For example, if the receiver is a control, this parameter could contain the control's cell.

controller

The window controller object that manages the window containing the receiver.

Return Value

The visible frame of the child object in the specified window, or `NSZeroRect` if the child does not belong to the receiver or is not in the window managed by *controller*. The returned rectangle must be in the window's coordinate space.

Discussion

You should override this method if the receiver embeds child objects that are not views but which occupy a portion of the receiver's frame or if the receiver embeds child views that lie outside of the receiver's own frame but are still visible. If all of the receiver's children are views and are enclosed completely within the receiver's frame, you do not need to override this method. Controls that embed cells might use this method to return the frame for each cell.

Constants

Key paths dictionary keys

These variables identify the dictionary keys used to group and identify your code's available key paths.

```
extern NSString * const IBAttributeKeyPaths;
extern NSString * const IBToOneRelationshipKeyPaths;
extern NSString * const IBToManyRelationshipKeyPaths;
extern NSString * const IBLocalizableStringKeyPaths;
extern NSString * const IBLocalizableGeometryKeyPaths;
extern NSString * const IBAdditionalLocalizableKeyPaths;
```

Constants

`IBAttributeKeyPaths`

Identifies the scalar attributes of an object. Scalar attributes include `NSString`, `NSNumber`, `NSDate`, `NSValue`, and other objects that contain or represent scalar values.

`IBToOneRelationshipKeyPaths`

Identifies the attributes of an object that contain other related objects.

`IBToManyRelationshipKeyPaths`

Identifies the attributes of an object that contain collections of other related objects.

`IBLocalizableStringKeyPaths`

Identifies the scalar attributes of an object that contain localizable strings.

`IBLocalizableGeometryKeyPaths`

Identifies the scalar attributes of an object that contain geometric information that can change during localization.

`IBAdditionalLocalizableKeyPaths`

Identifies any localizable attributes that are not stored as strings or geometry-related scalar types.

Declared In
`InterfaceBuilderKit/IBObjectIntegration.h`

Document Revision History

This table describes the changes to *NSObject Interface Builder Kit Additions Reference*.

Date	Notes
2007-05-10	New document describing methods added to the NSObject class to support their manipulation in Interface Builder.

Document Revision History

Index

I

K