Version 2.0 July 1992 by Kevin Brain. Released to the Public Domain.

InspectorManager

INHERITS FROM Object

CLASS DESCRIPTION

The InspectorManager class was designed to manage all of an application's inspector panels, especially for large applications with multiple nib files. Rather than requiring all inspector panels for an application to appear in a single nib file, this class allows inspector panel views created in various nib files to be managed by a single inspectorManager object. This has the advantage that connections between inspector panel controls and other objects can be made in any of the application's nib files, and the panel subsequently given to the inspectorManager object to be managed. This class is designed around the premise that selecting a certain type (or 'group') of inspector with the inspector's popUpList cannot uniquely determine the inspector that needs to be shown . The inspector that gets shown may depend on the currently selected object, if any. (For example, selecting 'Contents' in Workspace Manager may display the 'sound', 'tiff', or another inspector, depending on the currently selected file.) Therefore, the InspectorManager class lets its delegate determine which inspector should be shown when an inspector group is selected via the popUpList. This class uses compositing to and from an offscreen window to swap between inspector

views quickly (code and technique to do this was borrowed from the ToolInspector example by Sharon Biocca Zakhour, NeXT Developer Support Team).

The inspector panel managed by the InspectorManager class consists of three basic components: a popUpList at the top of the panel, an area where the inspector views are displayed, and a matrix of two buttons labelled "Revert" and "OK". The popUpList is referred to in this documentation as the 'group popUpList', and it selects the group (or 'type') of inspector to be viewed (as it does in Workspace Manager and Interface Builder, for example). Use of the popUpList and the Revert/OK buttons is optional; either can be hidden or displayed (using the methods **showGroupPopUp**, **hideGroupPopUp**, **showRevertOK**, and **hideRevertOK**). Initially, both are hidden. The group popUpList is automatically shown when a second group is added.

The InspectorManager class maintains a list of inspector views, most of which will be given to it from other objects. (Here 'inspector views' does not refer to objects of a particular view class, but any view showing an inspector.) Inspector views are given to the InspectorManger using either the **addInspector:title:** or the **addInspector:title:atLocation::cached:cacheWindow:** method. The first parameter to these methods is the view that draws the inspector (for example, a box with several controls as subviews). The second parameter (*title*) is the string that will be shown in the title bar of the inspector panel whenever the inspector is displayed. Both of these methods return an unsigned int that is subsequently used to indicate the particular inspector view. The **showing:** method returns the boolean value YES if the given inspector view is currently showing (and NO otherwise). The *cached* and *cacheWindow* parameters allow you to specify if the window should be cached offscreen for quick redraws. (This is new to version 2.0. In version 1.0, all windows were cached offscreen.)

The InspectorManager class also maintains a list of groups for the inspector. Groups are added with the method **addGroup:**, which takes as a parameter the name of the group as it is to be displayed in the popUpList, and returns an unsigned int that is subsequently used to indicate the group. By default, the **addGroup:** method assigns sequential numbers (starting at one and incrementing to

nine) to the groups to act as command-key equivalents to select the group and display the inspector panel. (This can be disabled by sending **setUseKeyEquivalents:**NO before any groups are added.) The currently selected group is returned by the **group** method.

The class has a delegate, which it sends messages to when the inspector panel's controls are used. When a group is selected (via the group popUpList), the delegate receives a **groupChanged:to:** method (if it responds to it). The second parameter of the **groupChanged:to** method is the number returned by the **addGroup:** method when this group was added. If the 'Revert' or the 'OK' button is pressed, an **inspectRevert:** or **inspectOK:** message, respectively, is sent to the delegate (again, only if the delegate responds to them). The first parameter of all these delegate methods is a pointer to the inspectorManager object.

A particular inspector is displayed by sending the InspectorManager object either the **switchToInspector:** or **showInspector:** method. The **switchToInspector:** method will remove all currently displayed inspector views before displaying the given inspector, while **showInspector:** will just show the given inspector without removing any others. The **hideInspector:** method removes the single given inspector view from the inspector panel's subview list. The **showInspector:** and **hideInspector:** methods enable multiple inspector views to be displayed at once on the inspector panel. In the simplest case, however, all inspector views will be the same size, only one will be displayed at once, and **switchToInspector:** will be used to display the desired inspector view. Note that the **switchToInspector:** and **hideInspector:** methods do not erase the removed inspector views, but instead rely on other views to be drawn over the same area. This requires you to be careful when laying out your inspector views in Interface Builder, but it allows inspectors to be switched as quickly as possible.

The InspectorManager class uses a nib file, Inspector.nib, which contains the single inspector panel and a default "message inspector" view. When the InspectorManager initializes, it adds the message inspector view to its list of inspectors. The message inspector is used to display any desired

message, which is done using the **showMessage:** method. (This method invokes **switchToInspector:**.) Initially there are no groups in the popUpList. (Actually, the group 'None' is in the popUpList in Inspector.nib file because Interface Builder doesn't handle a popUpList without any elements, but 'None' is removed when the first group is added with **addGroup:**.)

USAGE

Step-by-step guide to using InspectorManager:

1) Add the class and nib file to your project.
2) Create an instance of Inspector Manager in one of the following ways:
    a) With Interface Builder: Drag InspectorManager.h over the class window and drop, then instantiate the class. Connect an outlet from your controlling object to the instance.
    b) Programmatically: use the following line. I find appDidInit: is a good place.
    inspectorManager = [[InspectorManager alloc] init];
3) Add groups. For example:
    attributeGroup = [inspectorManager addGroup:"Attributes"];
    miscellaneousGroup = [inspectorManager addGroup:"Miscellaneous"];
4) Create the necessary inspector views, possibly in various nib files. Ensure the sizes are such that the inspectors showing at any given time together cover old inspector views. In the simplest case, where only one view is displayed in a panel at once, make all views the same size.
5) Send the inspector views to the inspectorManager with messages such as:
    noteInspectorNum = [inspectorManager addInspector:(id)rectangleInspectorView
        title:(const char *)"Rectangle Inspector" atLocation:0 :0
        cached:YES cacheWindow:[rectangleInspectorView window]];
6) Set inspectorManager's delegate
        [inspectorManager setDelegate:self];
    It is expected that many applications will change the delegate constantly as the currently selected object or window changes.

7) Implement the delegate method **groupChanged:to:** for all delegates. The easiest case (where each popUpList item brings up a particular inspector panel) could be implemented as follows:

```
- groupChanged:sender to:(int)newGroup;
{
    [inspectorManager switchToInspector:(newGroup - FIRSTADDEDINSPECTOR)]
    return self;
}
```

(Note: FIRSTADDEDINSPECTOR is defined in InspectorManager.h as the number that will be returned by the first addGroup: message sent.)

Most implementations of the **switchToInspector:** method will be a lot more involved. Normally, they will involve switching to various inspector views based the currently selected object and the currently selected inspector group.

Managing the changing of the Inspector Views

Inspectors must always reflect the current selection. This is the biggest challenge in implementing inspectors correctly. Here are a couple of strategies that could be used:

1) Use an object to serve as the InspectorManager's delegate for each window. When an object is selected, inform the delegate. When the key window changes, the InspectorManager delegate object for the new window updates the inspector.

2) Use the first responder mechanism. Override "becomeFirstResponder" for objects and have it trigger the update of the inspector panel.

3) Use the automatic updating mechanism. Send **setAutoupdate:**YES to your application object and then define a **windowDidUpdate:** method in the windows' delegates. According to the docs, if automatic updating is on, **update** is sent to each of the Application's Windows after each event has been processed, and Window's default version of the **update** method sends the delegate a **windowDidUpdate:** message, if the delegate can respond. I am not sure if this notification would

occur before the mouseUp event.

The first mechanism is the one I have used in the programs where I have used InspectorManager. I will outline this method below.

Access to the InspectorManager instance:
It is convenient to make a pointer to the InspectorManager instance readily available to other objects. One way to do this is to create the instance in the Application object (**inspectorManager = [[InspectorManager alloc] init]**), and then define a method (**- inspectorManager { return inspectorManager; }**) to return the pointer to it. Then you can always send a message to the InspectorManager instance by sending to **[NXApp inspectorManager]**.

Setting the delegate when the window changes
Have an object for each window that knows (or can determine) what is (are) the currently selected object(s) within the window. This object will act as the InspectorManager's delegate while that window is the key window. By making it the InspectorManager's delegate, it will then be notified when the group is changed (via the popUpList). To cause the object to become the delegate, we can use code like the following in the window delegate's **windowDidBecomeMain:** (or delegate's **windowDidBecomeKey:**) method.

```
- windowDidBecomeMain:sender
{
        // while this window is main, receive messages from inspector
        [[NXApp inspectorManager] setDelegate:self];
        [self groupChanged:self to:[[NXApp inspectorManager] group]];
        return self;
}
```

(In this example, the same object acts as both delegate to the window and to InspectorManager, hence the message to "self").

Implementing groupChanged:to:
Within this method, you simply send **switchToInspector:** and/or **showInspector:** messages to display the inspector(s) appropriate to the currently selected object. For example:

```
- groupChanged:sender to:(int)newGroup
{
    id    im;
    im=[NXApp inspectorManager];
    if (newGroup == ATTRIBUTESGROUP) {
        [[im panel] disableFlushWindow];
        switch (selectedObjectType) {
            case BOX:
                [im switchToInspector:BOXATTRIBUTESINSPECTOR];
                break;
            case CIRCLE:
                [im switchToInspector:CIRCLEATTRIBUTESINSPECTOR];
                break;
            }
        [selectedObject reflectAttributes];
        [[[im panel] reenableFlushWindow] flushWindowIfNeeded];
        }
    else {
        [im showMessage:"Not\nApplicable"];
        }
}
```

Reflecting new selections
When an object is selected, you can have the inspector updated by sending a message to the

InspectorManager delegate, such as:
[[[NXApp inspectorManager] delegate] groupChanged:self to:[[NXApp inspectorManager] group]]

VERSION / HISTORY

This is Version 2.0 of InspectorManager, distributed July, 1992.

The big enhancement in version 2.0 is an improved implementation of the offscreen caching. Rather than caching all inspector views in a single large offscreen window, I now use multiple windows. Caching can be disabled for each inspector view, but if caching is used for a view, you can supply the cache window rather than have InspectorManager create it. In most cases, it will be convenient to use the window holding the inspector view laid out in InterfaceBuilder for the cache. (In 1.0, when an inspector view was given to InspectorManager, it was removed from the window it was in, time was wasted allocating window space to cache it, and the old window just hung around taking up space. Now the original window can be used). The other enhancement to 2.0 is the **showMessage:** method.

Version 1.0 of InspectorManager, the initial "release", was distributed November, 1991.

AUTHOR / SUPPORT

ksbrain@zeus.waterloo.edu
University of Waterloo / Department of Systems Design / Waterloo, Ontario/N2L 3G1

Compositing techniques and functions from the ToolInspector example by Sharon Biocca Zakhour, NeXT Developer Support Team.

I am interested in hearing suggestions from people who use this class. Time permitting, I will make further versions available as significant improvements or bug-fixes are made.

THIS OBJECT CLASS IS DISTRIBUTED AS IS, WITH NO WARRANTY OR GUARANTEE EXPRESSED OR IMPLIED IN ANY RESPECT. THE AUTHOR IS NOT LIABLE FOR ANY DAMAGES WHATSOEVER DIRECTLY OR INDIRECTLY RELATED TO THE USAGE OF THIS WORK.

INSTANCE VARIABLES

Declared in InspectorManager id inspectorPanel;
id messagePanel;
id inspectorStrings;
id messageTextField;
id messageBox;
id delegate;
id revertOKOut;
id popupOut;
id inspectorList;
id groupList;
id visibleInspectors;
NXRect lastRect;
BOOL useKeyEquivalents;

inspectorPanel	outlet to inspector Panel
messagePanel	outlet to offscreen window that holds message inspector
inspectorStrings	outlet, stringTable object for inspector

messageTextField	outlet, textField that displays message
messageBox	outlet, view that displays "message" inspector
delegate	the inspectorManager's delegate
revertOKOut	outlet, matrix containing 'Revert' and 'OK' button cells
popupOut	outlet, button that covers popUpList
inspectorList	Storage object containing list of inspectors
groupList	Storage object containing list of groups
visibleInspectors	List object containing list of visible inspector views
lastRect	rect in layout window for last inspector view added
useKeyEquivalents	determines whether or not key equivalents are used
inspectorListEntry	structure of inspectorList and groupList entries
inspectorListEntry.view	the view object of this inspector
inspectorListEntry.cacheWindow	offscreen window that caches the view
inspectorListEntry.title	pointer to inspector title
inspectorListEntry.showing	YES if showing in the panel
inspectorListEntry.cached	YES if showing in the view is cached
inspectorListEntry.offscreenRect	rect in layout view for compositing

METHOD TYPES

Initializing a new instance - init

Adding inspectors and groups - (unsigned int)addInspector: (id)theView title:(const char *)theTitle;
- (unsigned int)addInspector:(id)theView title:(const char *)theTitle atLocation:(NXCoord)xLoc :(NXCoord)yLoc cached:(BOOL)isCached cacheWindow:

	(id)theCacheWindow; - (unsigned int)addGroup:(const char *)theTitle; - setUseKeyEquivalents:(BOOL)use;
	Displaying/hiding inspectors views - switchToInspector:(unsigned int)newInspector; - showInspector:(unsigned int)newInspector; - hideInspector:(unsigned int)newInspector;
Checking inspector visibility	- (BOOL)showing:(unsigned int)inspectorNum;
	Retrieving current group - (int)group;
	Displaying a message - showMessage:(const char *)theMessage; - messageTextField;
Retrieving inspector objects	- panel; - popUpListButton; - revertOKMatrix;
Setting/returning delegate	- setDelegate:(id)anObject; - delegate;
Targets of inspector panel controls	- selectGroup:sender; - revertPressed:sender; - okPressed:sender;

Showing/Hiding panel controls
- showRevertOK;
- hideRevertOK;
- showGroupPopUp;
- hideGroupPopUp;

INSTANCE METHODS

addGroup:

- (unsigned int)**addGroup:**(const char *)*theTitle*

Adds the group named *theTitle* to the list of groups maintained by the inspector. The group is also added as an item in the group popUpList. Sequential numbers (starting at one) are assigned to the groups as command-key equivalents to select the group. If this is the first group added, the inspectorManager sends itself a showGroupPopUp message to show the group popUp button (it is hidden until the first group is added). The number returned is subsequently used to compare with numbers returned by the **group** method to determine the currently selected group. Numbers assigned to groups will start with 0 and increment sequentially, so it is possible to create manifests for the groups in an application and then add the groups in the appropriate order so that the manifests represent the correct number.

See also: - **group:**

addInspector: title:

- (unsigned int)**addInspector:**(id)*theView* **title:**(const char *)*theTitle*

Invokes **addInspector: title:atLocation::cached:cacheWindow:** with the x and y location of the view set to the value of the manifests LOWERLEFTX and LOWERLEFTY (defined in InspectorManager.h), caching enabled, and a nil cacheWindow. This is a point just above the Revert and OK buttons in the provided Inspector.nib file. Returns a number which is subsequently used to

get the inspectorManager object to display *theView* as an inspector view.

See also: - **switchToInspector:**, - **showInspector:**, - **hideInspector:**

addInspector: title:atLocation::cached:cacheWindow:

- (unsigned int)**addInspector:**(id)*theView* **title:**(const char *)*theTitle*
  atLocation:(NXCoord)*xLoc* **:**(NXCoord)*yLoc*
  cached:(BOOL)*isCached* **cacheWindow:**(id)*theCacheWindow*

Adds *theView* to the InspectorManager's lists. *theTitle* is the title as it should appear in the inspector panel's title bar when the inspector is displayed. *xLoc* and *yLoc* give the location of the origin of the view within the inspector panel. If *isCached* == NO, the view is not cached in an offscreen buffer. (When the view is switched in, it is redrawn by sending it a **display** message.) When the view is cached, (*isCached* == YES), *theCacheWindow* specifies the window into which it will be cached. Often this will be the window in which the view was laid out in IB, and will be specified as [*theView* window]. If *isCached* == YES and *theCacheWindow* == nil, a new window will be created for the cache. This method returns a number which is subsequently used to get the inspectorManager object to display *theView* as an inspector view.

See also: - **switchToInspector:**, - **showInspector:**, - **hideInspector:**

delegate

- **delegate**

Returns the InspectorManager's delegate, or **NULL** if it doesn't have one.

See also: - **setDelegate:**

group

- (int)**group**

Returns the currenly selected group. The number returned is the number that was returned when the group was added with the **addGroup:title:** method.

See also: - **addGroup: title:**

hideInspector:

- **hideInspector:**(unsigned int)*inspectorNum*

Removes inspector number *inspectorNum* from the subview list of the inspector panel's content view. For efficiency, this method does not explicitly erase the old inspector view, but instead relies on subsequently shown inspectors to draw over top. If *inspectorNum* is not a valid inspector number the method does nothing. Returns **self**.

See also: - **switchToInspector:**, - **showInspector:**, - **showing:**

hideGroupPopUp

- **hideGroupPopUp**

Removes the group popUpButton from its superview (the inspector panel's content view). Note that this method does not redisplay the inspector panel's content view, so if you are not going to add an inspector view over the old group popUpButton position, you will have to erase the button's image yourself. Returns **self**.

hideRevertOK

- **hideRevertOK**

Removes the matrix containing the Revert and OK button cells from its superview (the inspector panel's content view). Note that this method assumes that a subsequent inspector view will be drawn over the location where the Revert/OK buttons were, so it does not and sends the inspector

panel's content view a display message. Returns **self**.

init

- **init**

Overrides the default init method. After sending the init to super to do the regular initialization, this method reads in the Inspector.nib file, sets the inspector panel to be a floating panel that becomes the key window only if needed, initializes the InspectorManager's lists, adds the default inspector views, and hides the inspector panel controls. Note that it is possible to create more than one instance of the InspectorManager class. This is allowed because the inspectorManager class, despite its name, would actually be suitable whenever a single panel is used to display one of several panels of information, such as in a preferences panel. Returns **self**.

okPressed:

- **okPressed:***sender*

An internal method which is the target of the inspector panel's OK button. This method sends the **inspectOK** message to the delegate if the delegate responds to it.

panel

- **panel**

Returns the inspector panel. Messages can be sent directly to the panel to move it, resize it, display it, make it non-floating, etc.

popUpListButton

- **popUpListButton**

Returns the id of the button that acts as the 'cover' for the group popUpList object. You may retrieve the actual popUpList object itself by sending '[[inspectorManagerInstance popUpListButton] target]'.

You could use the popUpList object to disable and enable individual items in the popUpList. (However, in keeping with the "user-is-in-control" philosophy, I believe it is better to always allow any group to be selected. Simply display the '**NOTAPPLICABLE**' inspector if the selected group does not apply to the currently selected object.) Do not add or remove objects from the popUpList, or change their names, by sending messages directly to the group popUpList object (always add groups with the **addGroup:** method).

revertPressed:

- **revertPressed:***sender*

An internal method which is the target of the inspector panel's Revert button. This method sends the **inspectRevert** message to the delegate if the delegate responds to it.

revertOKMatrix

- **revertOKMatrix**

Returns the id of the Matrix containing the 'Revert' and 'OK' buttons. You can get at the buttons themselves through this method, if you wish to change their titles or disable/enable them.

selectGroup:

- **selectGroup:***sender*

An internal method which is the target of the inspector panel's group popUpList object. This method sends the **groupChanged:to:** message to the delegate if the delegate responds to it.

setDelegate:

- **setDelegate:***anObject*

Makes *anObject* the InspectorManager's delegate, and returns **self**. See ªMETHODS IMPLEMENTED BY THE DELEGATEº at the end of this class specification.

See also: - **delegate**

setUseKeyEquivalents:

- **setUseKeyEquivalents:**(BOOL)*use*

If use == YES, sequential numbers (starting at one and incrementing to nine) are assigned to subsequently groups as command-key equivalents to select the group and display the inspector panel. By default, key equivalents are used. If they are not desired, send a **setUseKeyEquivalents:**NO message before adding groups. Returns **self**.

showing:

- (BOOL)**showing:**(unsigned int)*inspectorNum*

Returns YES if the inspector is currently displayed in the panel (if it exists in the subview list of the inspector panel's content view) or NO if it does not.

showInspector:

- **showInspector:**(unsigned int)*newInspector*

Adds inspector number *newInspector* to the subview list, without removing any views from the subview list of the inspector panel. Sends a display message to the inspector view. If *newInspector* is not a valid inspector number, or if its inspector is already visible, the method does nothing. Returns **self**.

See also: - **switchToInspector:**, - **hideInspector:**, - **showing:**

showGroupPopUp

- **showGroupPopUp**

Adds the group popUp button to the subview list of the inspector panel's content view, and sends the button a display message. Returns **self**.

showRevertOK

- **showRevertOK**

Adds the matrix containing the Revert and OK button cells to the subview list of the inspector panel's content view, and sends the matrix a display message. Returns **self**.

switchToInspector:

- **switchToInspector:**(unsigned int)*newInspector*

Removes all views from the subview list of the inspector panel, then adds inspector number *newInspector* to the subview list. For efficiency, this method does not explicitly erase the removed inspector views, but instead relies on subsequently shown inspectors to draw over top. If *newInspector* is not a valid inspector number, or if it is already visible, the method does nothing. Returns **self**.

See also: - **showInspector:**, - **hideInspector:**, - **showing:**

METHODS IMPLEMENTED BY THE DELEGATE

groupChanged:to:

- **groupChanged:***sender* **to:**(int)*newGroup*

Sent to the delegate (if it responds to it) when a new group is selected via the group popUpList. *sender* is the inspectorManager object and *newGroup* is the number of the group (the number returned by the **addGroup:** method when this group was added).

inspectRevert:

- **inspectRevert:***sender*

Sent to the delegate (if it responds to it) when the 'Revert' button is pressed. *sender* is the inspectorManager object.

inspectOK:

- **inspectOK:***sender*

Sent to the delegate (if it responds to it) when the 'OK' button is pressed. *sender* is the inspectorManager object.