

ModuleProtocol

Adopted By: id *TheModuleController*, declared in "Module.h"

Declared In: "ModuleProtocol.h"

Protocol Description

The **ModuleProtocol** protocol defines the methods your modules may use to query Resound or ask Resound to perform specific functions. *TheModuleController*, declared as an instance variable in your subclass of Module, adheres to this protocol. *TheModuleController* is your access to Resound's API. **ModuleProtocol** is backward-compatible with previous versions to version 2.0a, including some methods which are now pretty much unneeded. One new method, **moduleWindowDidBecomeMain**, does not exist in earlier versions.

Ordinarily your module should not include Windows of its own (use Panels instead). But if you think it's really necessary, whenever one of your private Windows becomes main or key, call **moduleWindowDidBecomeMain** to give Resound a chance to update its internal information (basically that there's no key sound window any more). You shouldn't call this if a Panel became key, since Panels shouldn't be able to become main, and becoming main is the important item.

If you completely replace the Sound object associated with a SoundView, you need to tell Resound this so it can invalidate the pasteboard if necessary. You'd do this by calling **isOwner:** to determine if the SoundView is the owner of the pasteboard, and if so, call **invalidatePasteboard**. Use the following code to help guide you when replacing the entire Sound.

```

if ([TheModuleController isOwner: mySoundView ])
    // mySoundView is the SoundView in question

    [TheModuleController invalidatePasteboard];

    /* Set the new sound here */

```

If your module uses existing Pasteboard data, you can check to see if this data is actually valid (its associated sound hasn't been freed) by using **stillExists**. If you don't, you run the risk of bombing the program by using the pasteboard. Use the following code to help guide you when using the pasteboard.

```

id pb=[Pasteboard new];
const char* t[2]={NXSoundPboardType,NULL};

if ([pb findAvailableTypeFrom:t num:1]!=NXSoundPboardType)
{
    /* can't paste anyway--no sound on the pasteboard. Do what you like in this situation here */

}
else if ([TheModuleController stillExists])
{
    /* do our pasting or whatever stuff here */

}
else
{
    [TheModuleController invalidatePasteboard];

    /* do whatever you want that _doesn't_ paste here */
}

```

Before modifying sounds, you should **stop** any currently playing sound, lest freed information is accidentally sent

down the NXSoundStream.

Method Types

Accessing sounds

- currentSound
- currentSoundView
- currentWindow
- currentPlayingSound
- currentPlayingSoundView
- currentPlayingWindow

Creating A New Sound

- brandNewSound

Modifying sounds

- newSound: for:
- soundChanged
- selectionChanged
- zoomChanged

Playing and recording

- stop
- isPlaying
- isRecording

Dealing with windows

- moduleWindowDidBecomeMain

Pasteboard Bug Issues

- isOwner:
- stillExists
- invalidatePasteboard

Backward compatibility

- runLongTimePanel
- infoChanged
- soundTouched

Instance Methods

brandNewSound:

- **brandNewSound:** *thisSound*

Tells Resound to generate a new window and SoundView for *thisSound*, and register it with the application.

currentPlayingSound

- **currentPlayingSound**

Returns the currently playing Sound, NULL if no sound is playing. Note that this is the sound displayed in the playing SoundView, but not the *actual* sound that's playing—the *real* sound is a scratch sound internal to the SoundView, which can be accessed through **[[TheModuleController currentPlayingSoundView] soundBeingProcessed]**.

currentPlayingSoundView

- **currentPlayingSoundView**

Returns the current playing SoundView, NULL if no sound is playing.

currentPlayingWindow

- **currentPlayingWindow**

Returns the current playing window, NULL if no sound is playing.

currentSound

- **currentSound**

Returns the current Sound (the one whose window is the main window), NULL if none.

currentSoundView

- **currentSoundView**

Returns the current SoundView (the one whose window is the main window), NULL if none.

currentWindow

- **currentWindow**

Returns the current window (any Resound sound window which is main), NULL if none.

infoChanged

- **infoChanged**

Tells Resound any information of the type found in Resound's Attributes inspector has been changed for the current Sound. This updates the selection inspector if it's showing, but does *not* set the close box of the current window to the broken-X style. If you've already called **soundChanged** or **newSound:for:** for the sound in question, you don't need to call this method. This method is retained primarily for backward-compatibility, and shouldn't really be used.

invalidatePasteboard

- **invalidatePasteboard**

Invalidates the Pasteboard without notifying the owner of the Pasteboard (which *might* be a freed Sound or SoundView). Thereafter, using the Pasteboard will not bomb Resound.

isOwner:

- (BOOL) **isOwner:***this_sound_or_soundview*

Returns YES if *this_sound_or_soundview* is the owner of the Pasteboard, or (if *this_sound_or_soundview* is a SoundView) if *this_sound_or_soundview*'s embedded Sound is the owner of the Pasteboard.

isPlaying

- (BOOL) **isPlaying**

Returns YES if Resound's console is currently playing any sound.

isRecording

- (BOOL) **isRecording**

Returns YES if Resound's console is currently recording any sound.

moduleWindowDidBecomeMain

- **moduleWindowDidBecomeMain**

Tells Resound a Window (not Panel) private to your module has become main. This is necessary to allow Resound to update its inspectors and clear out its Current Sound, Current SoundView, and Current Window information.

newSound:for:

- **newSound:** *thisSound* **for:** *thisSoundView*

Tells Resound that *thisSound* has been attached to the currently-registered *thisSoundView*, typically the current SoundView, replacing its old sound. This method automatically calls **soundChanged:**. This method does *not* actually set *thisSound* as *thisSoundView*'s Sound—you're responsible for that. You're also responsible for freeing the old sound.

runLongTimePanel:

- (BOOL) **runLongTimePanel**

Displays an attention panel that tells the user that a process could take a long time to perform. Returns YES if the user wants to go ahead with a process, NO otherwise. This method is here primarily for backward-compatibility, and should not be currently used. Use **NXRunAlertPanel()** instead, and possibly **NXUserAborted()** to provide an escape from excessively long functions.

selectionChanged

- **selectionChanged**

Tells Resound that the selection information for the current SoundView has been changed. This updates the Selection inspector if it's showing, but does *not* set the close box of the current window to the broken-X style. If you've already called **soundChanged** or **newSound:for:** for the sound in question, you don't need to call this method.

soundChanged

- **soundChanged**

Tells Resound that the current Sound has been changed. This updates the inspector and sets the close box of the current window to the broken-X style.

soundTouched

- **soundTouched**

Tells Resound that the current Sound has been changed but not in a way that invalidates its zooming. This updates the inspector and sets the close box of the current window to the broken-X style. Use **soundChanged** instead: **soundTouched** is now just a compatibility method.

stillExists

- (BOOL) **stillExists**

Returns YES if the pasteboard owner is NULL (no owner) or it's a valid Resound Sound, SoundView, or Sound Window that hasn't been freed.

stop

- **stop**

Stops any currently playing or recording sound.

zoomChanged

- **zoomChanged**

Tells Resound that the reduction factor for the current SoundView has changed. This updates the selection inspector if it's showing, but does *not* set the close box of the current window to the broken-X style. If you've already called **soundChanged** or **newSound:for:** for the sound in question, you don't need to call this method.