NXWorkspaceRequestProtocol

Adopted By: No NEXTSTEP classes

Declared In: appkit/workspaceRequest.h

Protocol Description

The NXWorkspaceRequestProtocol protocol is implemented by an object in the Workspace Manager that responds to application requests to do such things as open files, launch applications, and return file icons. This object is made available through the Application object's **workspace** method. For example, the following code uses the Workspace Manager object to request that a file be opened in the Edit application:

```
[[Application workspace] openFile:"/tmp/README"
   withApplication:"Edit"];
```

Before NEXTSTEP Release 3, some of the functionality of this protocol was found in the Speaker and Listener classes. New applications should be changed to send messages to the Workspace Manager object rather than to Workspace Manager's Listener.

Many of the methods in the NXWorkspaceRequestProtocol protocol depend on the existence of an Application object and its Speaker and/or Listener objects. This presents no problem for NEXTSTEP applications, but other applications may have to create an Application object and send it a **run** message in order to use these methods.

Method Types

Opening files	± openFile: ± openFile:withApplication: ± openFile:fromImage:at:inView: ± openFile:withApplication:andDeactivate: ± openTempFile: ± findString:inFile:
Manipulating applications	± launchApplication: ± launchApplication:showTile:autolaunch: ± hideOtherApplications
Manipulating files	± performFileOperation:source:destination:files:options: ± selectFile:inFileViewerRootedAt:
Requesting information about files	± getIconForFile: ± getInfoForFile:application:type: ± getFullPathForApplication: ± getInfoForFileSystemAt:isRemovable:isWritable: isUnmountable:description:type:
Requesting additional time before logout	± extendPowerOffBy:
Tracking changes to the file system	± fileSystemChanged ± didFileSystemChange

Updating registered services and file types
± findApplications

Tracking changes to the defaults database
± defaultsChanged
± didDefaultsChange

Tracking status changes for applications and devices
± beginListeningForApplicationStatusChanges
± endListeningForApplicationStatusChanges
± beginListeningForDeviceStatusChanges
± endListeningForDeviceStatusChanges
± getMountedRemovableMedia:
± mountNewRemovableMedia:
± checkForRemovableMedia

Animating an image ± slideImage:from:to:

Unmounting a device ± unmountAndEjectDeviceAt:

Instance Methods

beginListeningForApplicationStatusChanges

± (void)**beginListeningForApplicationStatusChanges**

Notifies Workspace Manager that the application wants to be notified of changes in the status of all applications. After sending this message, the Application object's delegate will receive the following messages when an application is launched or after one terminates:

app:applicationWillLaunch:
app:applicationDidLaunch:
app:applicationDidTerminate:

See also: ± **endListeningForApplicationStatusChanges**

steningForDeviceStatusChanges

± (void)**beginListeningForDeviceStatusChanges**

Notifies Workspace Manager that the application wants to be notified when various media (usually optical or floppy disks) are mounted or unmounted. After sending this message, the Application object's delegate will receive the following messages after a device is mounted or unmounted:

app:mounted:
app:unmounted:

These methods complement the **unmounting:ok:** method, which is sent just before a device is unmounted so that applications can end all their accesses to that device.

See also: ± **endListeningForDeviceStatusChanges**

rRemovableMedia

± (void)**checkForRemovableMedia**

Causes the Workspace Manager to poll the system's drives for any disks that have been inserted but not yet mounted. **checkForRemovableMedia** doesn't wait until such disks are mounted; instead, it asks the Workspace Manager to mount the disk asynchronously and returns immediately.

See also: ± **getMountedRemovableMedia:**, ± **mountNewRemovableMedia:**

Changed

± (void)**defaultsChanged**

Informs Workspace Manager that the defaults database has changed. Workspace Manager then reads all the defaults it is interested in and reconfigures itself appropriately. For example, this method is used by the Preferences application to notify Workspace Manager whether the user prefers to see hidden files.

See also: ± **didDefaultsChange**

ultsChange

± (BOOL)**didDefaultsChange**

Returns whether a change to the defaults database has been registered with a **defaultsChanged** message, and clears the internal flag (for the sending application) that indicates such a change.

ystemChange

± (BOOL)**didFileSystemChange**

Returns whether a change to the file system has been registered with a **fileSystemChanged** message, and clears the internal flag (for the sending application) that indicates such a change.

eningForApplicationStatusChanges

± (void)**endListeningForApplicationStatusChanges**

Notifies Workspace Manager that the application is no longer interested in notifications of application launches and terminations.

See also: ± **beginListeningForApplicationStatusChanges**

eningForDeviceStatusChanges

± (void)**endListeningForDeviceStatusChanges**

Notifies Workspace Manager that the application is no longer interested in notifications of the mounting and unmounting of devices (such as optical and floppy disks).

See also: ± **beginListeningForDeviceStatusChanges**

owerOffBy:

± (int)**extendPowerOffBy:**(int)*requestedMs*

Requests more time before the power goes off or the user logs out. An application can send this message in response to a **powerOffIn:andSave:** message that doesn't give the application enough time to prepare for the impending shutdown.

requestedMs is how many additional milliseconds are needed, beyond the number given in the **powerOffIn:andSave:** message. The actual granted number of additional milliseconds is returned.

See also: **- powerOffIn:andSave:** (Application), **- app:powerOffIn:andSave:** (Application delegate)

emChanged

± (void)**fileSystemChanged**

Informs Workspace Manager that the file system has changed. Workspace Manager then gets the status of all the files and directories it is interested in and updates itself appropriately. This method is used by many objects that write or delete files. Even if this method isn't invoked, Workspace Manager will note changes to the file system relatively quickly if it is the active application.

See also: ± **didFileSystemChange**

ications

± (void)**findApplications**

Instructs Workspace Manager to examine all applications in the normal places and update its records of registered services and file types. This can be useful in an project building application, for example.

g:inFile:

± (BOOL)**findString:**(const char *)*aString* **inFile:**(const char *)*filename*

Instructs Workspace Manager to open the file *filename* (specified with a complete path), with the string *aString* selected. The file is opened using the default application for its type. The application that opens the file must implement the **msgSetPosition:posType:andSelect:ok:** message in its Listener's delegate to find the selection.

athForApplication:

± (const char *)**getFullPathForApplication:**(const char *)*appName*

Returns the full path for the application *appName*, or NULL if *appName* isn't in one of Workspace Manager's application directories. The returned string is valid until the next message to a Speaker object; the application must copy it if it must be retained.

orFile:

± (NXImage *)**getIconForFile:**(const char *)*fullPath*

Returns a newly allocated NXImage with the icon for the single file specified by *fullPath*, or **nil** if there is an error.

See also: ± **getInfoForFile:application:type:**

orFile:application:type:

± (BOOL)**getInfoForFile:**(const char *)*fullPath*
application:(char **)*appName*
type:(NXAtom *)*type*

Retrieves information about the file specified by *fullPath*. After invoking this method, the string pointed to by *appName* is set to the application Workspace Manager would use to open *fullPath*. This string should be freed by the caller. The NXAtom pointed to by *type* will contain one of the following values or a file name extension such as ªrtfº indicating the file's type:

Value	*fullPath* is a:
NXPlainFileType	plain (untyped) file
NXDirectoryFileType	directory
NXApplicationFileType	NEXTSTEP application

NXFilesystemFileType	file system mount point
NXShellCommandFileType	executable shell command

Returns YES upon success, NO otherwise.

See also: ± **getIconForFile:**

orFileSystemAt:isRemovable:isWritable:isUnmountable: description:type:

± (BOOL)**getInfoForFileSystemAt:**(const char *)*fullPath*
isRemovable:(BOOL *)*removableFlag*
isWritable:(BOOL *)*writableFlag*
isUnmountable:(BOOL *)*unmountableFlag*
description:(char **)*description*
type:(char **)*fileSystemType*

Describes the file system at *fullPath*. Returns YES if *fullPath* is a file system mount point, or NO if it isn't. Upon success, *description* will describe the file system; this value can be used in strings, but it shouldn't be depended upon by program logic. Example values for *description* are ªhardº, ªnfsº, and ªforeignº. *fileSystemType* will indicate the file system type; values could be ªNeXTº, ªDOSº, or other values.

ntedRemovableMedia:

± (BOOL)**getMountedRemovableMedia:**(char **)*pathnames*

Asks the Workspace Manager for the pathnames of all currently mounted removable disks, and returns whether the query was successful. Returns YES if it was successful in filling in *pathnames* and NO otherwise. *pathnames* is a pointer to a null-terminated, tab-separated list of full pathnames.

If the computer provides an interrupt or other notification when the user inserts a disk into a drive, the Workspace Manager will mount the disk immediately. However, if no notification is given, the Workspace Manager won't be aware that a disk needs to be mounted. On such systems, an application should invoke either **mountNewRemovableMedia:** or **checkForRemovableMedia** before invoking **getMountedRemovableMedia:**. Either of these methods cause the Workspace Manager to poll the drives to see if a disk is present. If a disk has been inserted but not yet mounted, these methods will cause the Workspace Manager to mount it.

The Disk button in an Open or Save panel invokes **getMountedRemovableMedia:** and **mountNewRemovableMedia:** as part of its operation, so most application won't need to invoke these methods directly.

See also: ± **mountNewRemovableMedia:**, ± **checkForRemovableMedia**

erApplications

± (void)**hideOtherApplications**

Hides all applications other than the sender. (Since the user can cause the same effect by Command-double-clicking on an application's tile, a programmatic invocation of this method is usually unnecessary.)

pplication:

± (BOOL)**launchApplication:**(const char *)*appName*

Instructs Workspace Manager to launch the application *appName*. *appName* need not be specified with a full path and, in the case of an application wrapper, can be specified with or without the ª.appº

extension. Returns YES if the application is successfully launched or already running, and NO if it can't be launched.

See also: ± **launchApplication:showTile:autolaunch:**

pplication:showTile:autolaunch:

± (BOOL)**launchApplication:**(const char *)*appName*
showTile:(BOOL)*showTile*
autolaunch:(BOOL)*autolaunch*

Instructs Workspace Manager to launch the application *appName*. If *showTile* is NO, Workspace Manager won't display a tile for the application. (The tile will exist, but it won't be placed on the screen.) If *autolaunch* is YES, the NXAutoLaunch command line default will be set as though the application were autolaunched from the dock. This method is provided to enable daemon-like apps that lack a normal user interface and for use by alternative dock programs. Its use is not generally encouraged.

Returns YES if the application is successfully launched or already running, and NO if it can't be launched.

See also: ± **launchApplication:**

ewRemovableMedia:

± (BOOL)**mountNewRemovableMedia:**(char **)*newPaths*

Causes the Workspace Manager to poll the system's drives for any disks that have been inserted but not yet mounted. **mountNewRemovableMedia:** waits until the new disks have been mounted and then fills in *newPaths* with a zero-terminated, tab-separated list of full pathnames to all newly mounted disks. This method returns YES if it was successful in filling in *newPaths* and NO otherwise.

See also: ± **getMountedRemovableMedia:**, ± **checkForRemovableMedia**

e:

± (BOOL)**openFile:**(const char *)*fullPath*

Instructs Workspace Manager to open the file specified by *fullPath* using the default application for its type. The sending application is deactivated before the request is sent. Returns YES if the file is successfully opened, and NO otherwise.

See also: ± **openFile:withApplication:andDeactivate:**, ± **openTempFile:**

e:fromImage:at:inView:

± (BOOL)**openFile:**(const char *)*fullPath*
fromImage:(NXImage *)*anImage*
at:(const NXPoint *)*point*
inView:(View *)*aView*

Instructs Workspace Manager to open the file specified by *fullPath* using the default application for its type. Before opening the file, Workspace Manager will provide animation to give the user feedback that the file is to be opened. To provide this animation, *anImage* should contain an icon for the file, and its image should be displayed at *point*, specified in *aView*'s coordinates.

The sending application is deactivated before the request is sent. Returns YES if the file is successfully opened, and NO otherwise.

See also: ± **openFile:withApplication:andDeactivate:**

:withApplication:

± (BOOL)**openFile:**(const char *)*fullPath* **withApplication:**(const char *)*appName*

Instructs Workspace Manager to open the file specified by *fullPath* using the *appName* application. *appName* need not be specified with a full path and, in the case of an application wrapper, can be specified with or without the ª.appº extension. The sending application is deactivated before the request is sent. Returns YES if the file is successfully opened, and NO otherwise.

See also: ± **openFile:withApplication:andDeactivate:**

:withApplication:andDeactivate:

± (BOOL)**openFile:**(const char *)*fullPath*
withApplication:(const char *)*appName*
andDeactivate:(BOOL)*flag*

Instructs Workspace Manager to open the file specified by *fullPath* using the *appName* application. *appName* need not be specified with a full path and, in the case of an application wrapper, can be specified with or without the ª.appº extension. If *appName* is NULL, the default application for the file's type is used. If *flag* is YES, the sending application is deactivated before the request is sent, allowing the opening application to become the active application. Returns YES if the file is successfully opened, and NO otherwise.

See also: ± **app:openFile:type:** (Application delegate)

mpFile:

± (BOOL)**openTempFile:**(const char *)*fullPath*

Instructs Workspace Manager to open the temporary file specified by *fullPath* using the default application for its type. The sending application is deactivated before the request is sent. Using this method instead of one of the **openFile:...** methods lets the receiving application know that it should delete the file when it no longer needs it. Returns YES if the file is successfully opened, and NO otherwise.

See also: ± **openFile:withApplication:andDeactivate:**

FileOperation:source:destination:files:options:

± (int)**performFileOperation:**(const char *)*operation*
source:(const char *)*source*
destination:(const char *)*destination*
files:(const char *)*files*
options:(const char *)*options*

Requests the Workspace Manager to perform some file operation, such as compressing or moving files. The files to be manipulated are located in the *source* directory; their names are specified relative to this directory and are listed in *files*. If more than one file is listed, their names must be tab-separated. The list can contain both files and directories; all of them must be located directly within *source* (not in one of its subdirectories).

Some operationsÐsuch as moving, copying, and linking filesÐrequire a *destination* directory to be specified. If not, *destination* should be the empty string (ªº).

The *options* argument is currently ignored. It's reserved for future enhancements.

The permissible values for *operation* are as follows:

Operation	Meaning
WSM_MOVE_OPERATION	Move file to destination

WSM_COPY_OPERATION	Copy file to destination
WSM_LINK_OPERATION	Create link to file in destination
WSM_COMPRESS_OPERATION	Compress file
WSM_DECOMPRESS_OPERATION	Decompress file
WSM_ENCRYPT_OPERATION	Encrypt file
WSM_DECRYPT_OPERATION	Decrypt file
WSM_DESTROY_OPERATION	Destroy file
WSM_RECYCLE_OPERATION	Move file to recycler
WSM_DUPLICATE_OPERATION	Duplicate file in source directory

Note: WSM_ENCRYPT_OPERATION and WSM_DECRYPT_OPERATION might not be available on all systems.

This method returns a negative integer if the operation fails, 0 if the operation is performed synchronously and succeeds, and a positive integer if the operation is performed asynchronously. The positive integer is a tag that identifies the requested file operation. When the operation is completed, the delegate of the Application object is informed in an **app:fileOperationCompleted:** message. The tag is passed as the second argument in the message.

e:inFileViewerRootedAt:

± (BOOL)**selectFile:**(const char *)*fullPath*
inFileViewerRootedAt:(const char *)*rootFullpath*

Instructs Workspace Manager to select the file specified by *fullPath*. If a path is specified by *rootFullpath*, a new file viewer is opened. If *rootFullpath* is an empty string (ªº), the file is selected in the main viewer. Returns YES if the file is successfully selected, and NO otherwise.

ge:from:to:

± (void)**slideImage:**(NXImage *)*image*
from:(const NXPoint *)*fromPoint*
to:(const NXPoint *)*toPoint*

Instructs Workspace Manager to animate a sliding image of *image* from *fromPoint* to *toPoint*, specified in screen coordinates.

tAndEjectDeviceAt:

± (BOOL)**unmountAndEjectDeviceAt:**(const char *)*path*

Unmounts and ejects the device at *path*. Returns YES if the device is successfully unmounted or *path* is badly formed; returns NO otherwise.