MiscDependency

Adopted By:

Declared In: misckit/MiscDependency.h

Protocol Description

The MiscDependency protocol is designed to account for data dependencies that exist between objects. The notion of a data dependency roughly coincides with the C concept of a pointer. Whenever one object *foo* points to another object *fum*, it implicitly creates a dependency between the two objects:

```
707265_paste.eps ¬
```

As a simple example, *foo* might be a window and *fum* one of its subviews. Information moves

across the dependency link in both directions: the window might ask the view to respond to an event, and the view might ask the window to move it to a new location in the responder chain.

The dependency graph of an application determines how information flows between objects. It is important to keep track of this structure for two reasons:

1. Resource Management
2. Synchronization of the Application's State

The dependency structure determines when an object can be safely freed and its resourcesÐparticularly its memoryÐreclaimed. When an object becomes isolated in the dependency graph, it is no longer needed and can be deleted. A reference counting memory management system, such as one might find in a LISP environment, uses this idea to do automatic garbage collection. Such a system keeps track of the number of incoming dependency links to an object, and automatically frees it when the count falls to zero.

The dependency structure also determines how changes in one object propagate to other objects in the application. The simplest analogy is a spreadsheet: whenever one cell of a spreadsheet changes, the entire spreadsheet updates itself to reflect the new data. This idea appears throughout the AppKit in the guise of delegation. When a window becomes the key window, for example, it notifies its delegate, which in turn notifies all the objects in the application that need to reflect the change. As another example, when the user selects a drawing in a graphics program, any inspectors associated with the current selection reconfigure themselves to the new selection.

As long as an application's dependency structure is a tree, everything is simple and straightforward. Each object can manage its own children, freeing them when necessary, and an object can keep a

backpointer to its parent, as a view does with its window, so that requests can easily be sent in both directions. The dependency structure of the NEXTSTEP AppKit is not really tree, but it's close enough that one can usually get by with this simple style of organization. However, as the dependency graph gets more and more convolved, it becomes increasingly necessary to maintain the dependency structure explicitly, so that the problems of resource management and synchronization can be kept under control.

Failure to keep track of an application's dependency structure can easily lead to any or all of the following problems:

1. Memory Leaks
2. Messages Sent to Freed Objects
3. Parts of the Application Reflecting Stale Data

The MiscDependency protocol provides the framework to avoid these problems.

Using the MiscDependency Protocol

The MiscDependency protocol is simple to use. In standard C or Objective-C, when setting up a pointer from *foo* to *fum* one just writes,

```
foo->someVar = fum;
```

Using this construct, *foo* knows that it depends on *fum*, but *fum* is unaware of the new dependency

link. If *fum* conforms to the MiscDependency protocol, one should use instead,

```
[foo->someVar removeUser: foo];
[foo->someVar = fum  addUser: foo];
```

This combination does two things: it removes the dependency link to the old contents of *someVar*, and it creates the new dependency link to *fum*. The object *fum* now guarantees, at the very least, that it will not free itself without first making some effort to notify *foo*. By convention, this notification takes the form of a **willFree:** message sent from *fum* to *foo*; the argument of the message is the object that is being freed, in this case *fum*. This guarantee eliminates the most serious problem of *foo* sending a message to *fum* after *fum* has been freed.

It is the responsibility of individual classes that conform to MiscDependency to decide what, if anything, they wish to do beyond this simple guarantee. An object might, for example, use the dependency information to do both reference counting garbage collection and to broadcast announcement messages as its state changes. These services are provided by the MiscAnnouncer class.

The MiscDependency protocol also supports a weaker notion of dependency through the **addListener:** and **removeListener:** methods. These methods create and destroy one way links. A window's delegate, for instance, often doesn't care whether or not the window suddenly frees itself, since it never sends any messages to the window directly. It simply sits and listens for announcements from the window; if none come, it never takes any action. In such cases one should use the construct,

```
[fum  addListener: foo];
```

This message says the *foo* would like to be informed about changes to *fum*. It is up to *fum*, though, to decide howÐor even ifÐit will send out such notifications. The primary difference between a user and a listener is that an object makes no promises about when it will free itself to its listeners. In the current example, *fum* is allowed to vanish from the earth whenever it wants, without ever saying a word. If *foo* decides to free itself, though, it should make sure to send *fum* a **removeListener:** message to delete the dependency link.

Method Types

- addListener:
- addUser:
- removeListener:
- removeUser:

Instance Methods

addListener:

- **addListener:** *who*

Adds *who* as a new listener of the receiver. This method semantically creates a dependency link in which information flows in only one direction. It is up to the receiver to decide what services it wishes to provide for its listeners.

See also: - **addListener:** (MiscAnnouncer)

addUser:

- **addUser:** *who*

Adds *who* as a new user of the receiver. The receiver promises not to free itself without making some effort to notify *who*. It is up to the receiver to decide what other services it wishes to provide for its users.

See also: - **addUser:** (MiscAnnouncer)

removeListener:

- **removeListener:** *who*

Deletes *who* as a listener of the receiver.

See also: - **removeListener:** (MiscAnnouncer)

removeUser:

- **removeUser:** *who*

Deletes *who* as a user of the receiver. If it wishes, a class that conforms to MiscDependency can use this information to implement automatic garbage collection.

See also: - **removeUser:** (MiscAnnouncer)