Creating Examples:

How it works:

This program is designed so that an instructor may add or remove examples without any instructions as to how the program should respond. An example must consist of a single window that may contain any of the controls or a Matrix of controls (such as a matrix of buttons or sliders). It may also contain user defined objects, such as a controller object. There are several cases that need to be known.

This program manages to redirect targets and outlets to itself. It

then processes the message by animating a row of dots and then displaying the message sent in the scroll view associated with the window. It DOES NOT catch messages sent from local variables such as when a control responds to "sender". In that particular case, the program has a special case built in which animates a second message that it assumes is sent in response to a "takeDoubleValue:" message or any other "take" message.

For Example:

```
(In object1, maybe a Text Field)
        [object2 takeDoubleValueFrom:self];
(In object2, possibly a Slider)
        - takeDoubleValue:sender
        {
        theValue=[sender doubleValue:self];
```

```
}
```

The redirections of outlets are handled by parsing the header file of a user defined class (such as Example2.h) which must be located in the same directory as this application. Only those instance variables that are declared to be of type 'id' are checked to see if they correspond to a known object. Known objects are all objects in the window that inherit from the View class. The elements within a

Matrix are also known. The targets of all of the control objects and Matrix cells are also added to the table of known objects. This allows controller objects to be found since they are usually targets of controls on the window.

How to Create an Example:

Start a new module in Interface Builder. The file owner will be

unimportant, so one could make the file owner class 'Object'. Do not make it an Application since this will bring up a menu when the Example is chosen. Save the new .nib as "Example?.nib" with a number between 1 and 9. Acceptable names are "Example2.nib" or "Example5.nib".

Constructing the contents of the Window:

The objects within the window will be assigned the names according to the name listed the in *Inspector* under *Miscellaneous* (Command - 4). Altering the default names would be significantly more helpful than the default name system given by the NeXT. This listed name will be replaced if it is connected to an outlet of a user defined object.

User defined classes and objects of them can be used so long as the interface file(.h) and the implementation file(.m) are in the same

directory as the application file. These classes do need to be added to the *IB.proj* file. The new classes do NOT need to be named similar to the *Example* name. To do this the entire application must be recompiled after the new class is added to the project. If only a .nib file was added without any new classes, no recompilation is needed. The outlets within the class will rename the objects that they are connected to.

If a new class was added, recompile the application with 'make'. Otherwise, if only a new .nib was added, run the application.

Possible difficulties:

Since this program doesn't intervene in the message sending structure or set any trace bits (as *gdb* probably does), it succeeds in intercepting messages by redirecting through parts of itself. This

can create particualarly dangerous programs if any of the outlets are passed as arguments in a method. One such example would be removing a subview from a window. If the subview was connected to an outlet, this outlet would be reset at run-time to point to an intervening object. That particular object would then be passed as a subview. Since it is not even a view, looking through the subview list for it would be futile.

This program was intended for teachers to write simple demos to demostrate some aspects of the objective C language. It was not intended as a visual *gdb* and would not work for such a purpose. Keep the flaws of this 'interception' approach in mind as you form new examples.