observetester Has been written to act as a testbed for the ObserveDispatch object and general message dispatching paradigm. It is very simple and only includes a few interface objects. Simply compile the program and run.

Classes of this application:

Controller A very simple object to hold onto some instance variables and perform some initialization after the application has been initialized.

ObserveDispatch This is the primary workhorse object. It is responsible for maintaining the list of observers that are interested in individual actions. In addition it makes sure the secondary actions are forwarded to thos objects who are interested.

SimpleField A simple sub-class of TextField which contains an instance of the ObserveDispatch object.

Other files:

observetest.nib	The main nib file. Contains the Busy Box window.
IB.proj,	These files created by Interface Builder.
Makefile,	
observetest_main.m,	
observetest.iconheader	

Topics of interest from BusyBox:

Overriding forward:: to do message dispatching (ObserveDispatch)
Using selector names as keys in a hash table (ObserveDispatch)

GENERAL DISCUSSION

An examination of the features and benefits of different paradigms of process control in deterministic and non-deterministic systems,
or
The Use of Message Dispatching In the Implementation of Multi-threaded non-deterministic Interactive Computer Systems.

or
What use are distributed objects anyway?

This document describes 3 different paradigms which are in use today in the area of process flow of control within an application. By process flow I mean the mechanism by which a program determines what its next action will be at any given moment.

Event Driven Systems

The event driven application paradigm can be described as one in which events occur in the environment and they are placed onto a queue from which they are processed.

Normally the processing of the event queue is handled by a single process. A single event may cause multiple other objects to react, but their reaction can be characterized as a linear cascade of events through a single thread of control.

A deterministic system usually has an event loop in which each event is pulled off the event queue and a pre-determined action is performed based on the event type. This method of control is indicative of the early Macintosh system and MS Windows. The application program is completely responsible for pulling the events off the event queue and determining what to do with them.

Responder Chains

The responder chain goes one step further up the event driven ladder. The application does not have to explicitly determine what action will happen for each event in the queue. A general event handling loop passes events down a responder chain. This is the design of the NeXTSTEP environment. The event loop pulls the events off the queue and tries to pass it to objects that are in the responder chain. Once one of the objects responds to the event, it is no longer passed on. The responding object is responsible for exhibiting the desired response to the event. The response is much the same as in the first event driven system in that a chain of events may occur as a response, but it is still a single thread of execution.

This relaxed determinism allows for a more object oriented environment which can be much more dynamic than the static deterministic system. It also brings a level of simplicity to the control loop. It is generally good for single threaded user applications.

Triggered Broadcast Systems

As a more non-deterministic system, triggered broadcast systems are an extension of the responder chain. Instead of an event being sent down a single responder chain, there may be a number of objects that are interested in the occurence of an event. In this case, the event handling loop would broadcast the event to all those objects that are interested in it. This is well suited to an environment in which there can be multiple threads of control. In addition to adding the broadcast mechanism, events can take on more variety. Typically events are well defined hardware and software entities (mouse, keyboard, timer). This is very limiting in that not all events are of this variety. Another class of events are those defined by an application. These events can be more temporal (a price changed, a value is too high, etc.). These events can also be broadcast if the event queue is expanded to include object messages as well as normal hardware event information.

ObserveDispatch - object
This object is a general purpose implementation of message broadcasting. It facilitates the development of Triggered Broadcast Systems. An object that intends to broadcast messages should instantiate an ObserveDispatch object. Certain methods within the object are designated as observable methods. These methods will be responsible for broadcasting whenever they are called.

Example:

Glossary
Primary Function - In the broadcast paradigm, a primary function is one that has a primary interface to the outside world. The primary function is the one that will broadcast its designated broadcast messages.

Designated Broadcast Messages - These are the messages that an object will broadcast while it is performing a primary function.

Problem: Given a main field in a form, how do you make a number of other objects automatically change their values whenever the main field changes its value.

Primary Method		**Designated Broadcast Message**
takeIntValueFrom:	==>	setIntValue:

In this implementation of message dispatching, the dispatcher does not look at any return values from the method calls. The objects that receive the broadcast message must determine what action they would like to take and respond back to the originating object if that is part of their response. Another implementation of the Dispatcher could dispatch messages synchronously or asynchronously. When dispatched synchronously (default), the primary function will not return until all the observers have executed their designated broadcast messages. When broadcasting asynchronously, the primary method does not wait for the observers to execute their designated broadcast messages.

This implemenation is very simple and straightforward. As you look at the size and complexity of the code you quickly realize this. But the power of the methodology is tremendous, especially when used in an environment that uses distributed objects as well. If you look past the simple example here and think of other situations (database, spreadsheet, etc) it becomes apparent how useful this might be. The code is there for the using but is merely an example of what can be done using this paradigm.

If you have any questions or comments address them to **adams@adamation.com**