

*Building a Usability  
Prototype in Visual Basic,  
Flash, and Dreamweaver:*

*A How-To Guide for  
CSC318*

*University of Toronto, Department of Computer Science*

*Daniel Wigdor, September 2002*

# 1. Abstract

This document is written in support of CSC318 as a quick introduction to building prototypes using three different tools: Microsoft Visual Basic, Macromedia Flash, and Macromedia Dreamweaver. It begins by giving a quick introduction to building a prototype, discusses selecting the correct tool, and then proceeds to give a step-by-step guide to creating a prototype using each of the tools. The prototype being built will be a simplification of the Virtual Science Fair, as outlined in your text and on the web (<http://moosburg.cs.vt.edu>).

# 2. Table of Contents

1.	<i>Abstract</i> .....	2
2.	<i>Table of Contents</i> .....	2
3.	<i>What this Document is for, How to Use It</i> .....	3
4.	<i>Selecting the Tool</i> .....	3
4.1	<b>Intro to Visual Basic</b> .....	4
4.2	<b>Intro to Dreamweaver</b> .....	4
4.3	<b>Intro to Flash</b> .....	4
5.	<i>Building a VB Prototype</i> .....	5
5.1	<b>Hello World Application</b> .....	5
5.2	<b>Building a Virtual Science Fair Prototype in VB</b> .....	9
5.3	<b>Suggested Readings for VB</b> .....	17
6.	<i>Building a Dreamweaver Prototype</i> .....	18
6.1	<b>Hello World Application</b> .....	18
6.2	<b>Accessing Objects in JavaScript – the DOM</b> .....	23
6.3	<b>Building a Virtual Science Fair Prototype in Dreamweaver</b> .....	24
6.4	<b>Suggested Readings for Dreamweaver:</b> .....	30
7.	<i>Building a Flash Prototype</i> .....	31
7.1	<b>Hello World Application</b> .....	31
7.2	<b>Building a Virtual Science Fair Prototype in Flash</b> .....	38
7.3	<b>Suggested Readings for Dreamweaver:</b> .....	46
8.	<i>Author’s Thoughts</i> .....	47

### **3. What this Document is for, How to Use It**

This document is written in support of CSC318 as a quick introduction to building prototypes using three different tools: Microsoft Visual Basic, Macromedia Flash, and Macromedia Dreamweaver. It begins by giving a quick introduction to building a prototype, discusses selecting the correct tool, and then proceeds to give a step-by-step guide to creating a prototype using each of the tools. The prototype being built will be a simplification of the Virtual Science Fair, as outlined in your text and on the web (<http://moosburg.cs.vt.edu>).

I recommend following the instructions in this document to build the prototypes as outlined herein. Systems at the lab have been setup with each of Microsoft Visual Basic 6.0, Macromedia Flash MX, and Macromedia Dreamweaver MX. Note also that the source code for all of the prototypes in this document is available for download from the course website (or directly: <http://www.cs.toronto.edu/~dwdor/CSC318/prototypes>).

### **4. Selecting the Tool**

When building a prototype of any system, it is important to balance the breadth and depth of the prototype. Breadth comes from depicting as many functions as possible, while depth comes from actually implementing functionality. For example, a prototype of a cash register is broad if it includes all the buttons for the functions that can be performed, and certain functions are deep if pressing one of the buttons carries out the required task. Depth can be simulated, like for example requiring that the total of an order be a particular value in order for the tax function to work correctly, or it can be actual, by building the ability to calculate tax on any order.

In picking the tool you will use to build your prototype, you should consider what aspects of the system you wish to include, your expertise with any tool, and the level of depth required for your prototype. Any of the three tools outlined here, with sufficient expertise, could be used to build a broad and deep prototype, but different tools are better for highlighting particular aspects of a system quickly.

Each of the tools used here for developing prototypes is designed to build either complex presentations, complete web sites, or sophisticated desktop applications. We'll be using only a small subset of the functionality of each of the tools in order to build our prototypes in 318.

## **4.1 Intro to Visual Basic**

Visual basic is an IDE (Integrated Development Environment) with integrated editor, debugger, and GUI builder. The environment is designed to very rapidly develop fully functional applications. VB is your best bet for building deep prototypes quickly.

## **4.2 Intro to Dreamweaver**

Macromedia Dreamweaver is the HTML editor of choice for 318. HTML is the language of the web, and Dreamweaver generates HTML (with JavaScript built-in). Anything that can be done in HTML and JavaScript can be done in Dreamweaver. Because web pages are somewhat client (web browser) dependant, and are rendered differently by different browsers, it's difficult to build deep functionality into a web page, but it can be done. I suggest using an HTML tool only when the system you are prototyping is a web site. I also suggest that you work with your TA to decide on which web browser they will be examining your prototype.

## **4.3 Intro to Flash**

You have no doubt seen Macromedia Flash animations on the web, and have some idea of what they can do. Flash is a good tool for building broad prototypes, and has a scripting language that allow for a certain amount of depth. Its most common use is for quickly adding animation, sound, and video to a prototype.

## 5. Building a VB Prototype

As an introduction to Microsoft Visual Basic 6.0, we will first build a “Hello World” application, while giving an overview of the tool. Next, we will develop a prototype of the MOOsberg Virtual Science Fair in VB. Lastly, you will find a list of suggested readings.

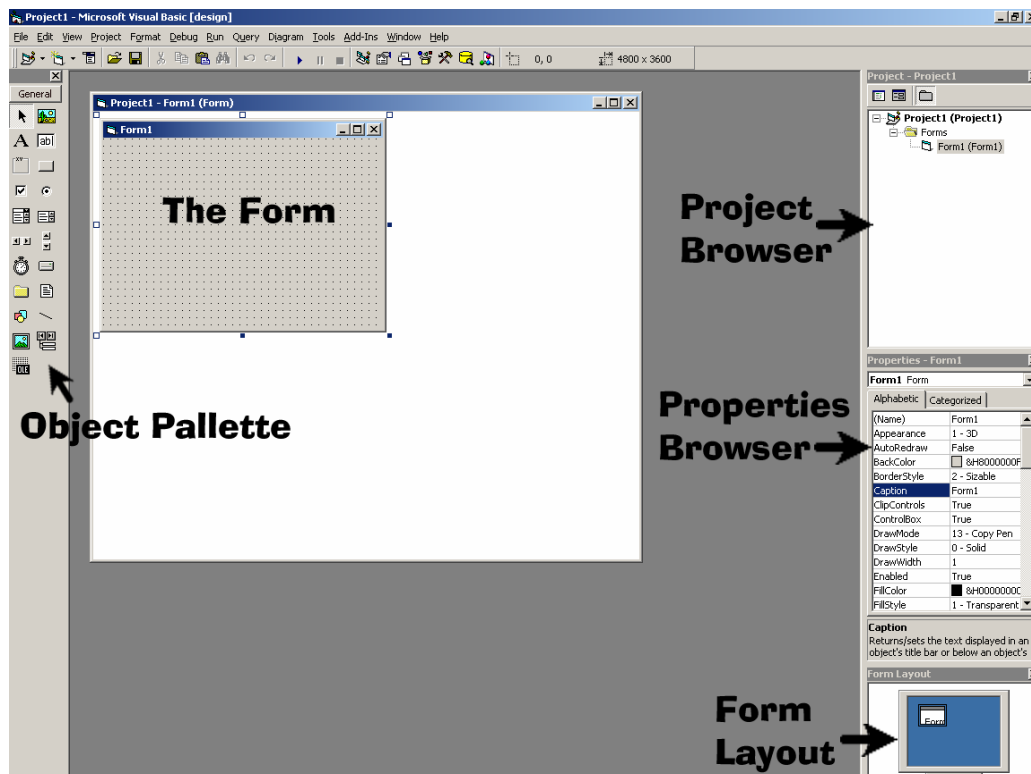
### 5.1 Hello World Application

Our Hello World application will give you a quick intro to prototyping, and will include both a “Hello World” message and a counter that increments on a button click, so you’ll see some basic deep functionality of the tool.

#### 5.1.1 Overview

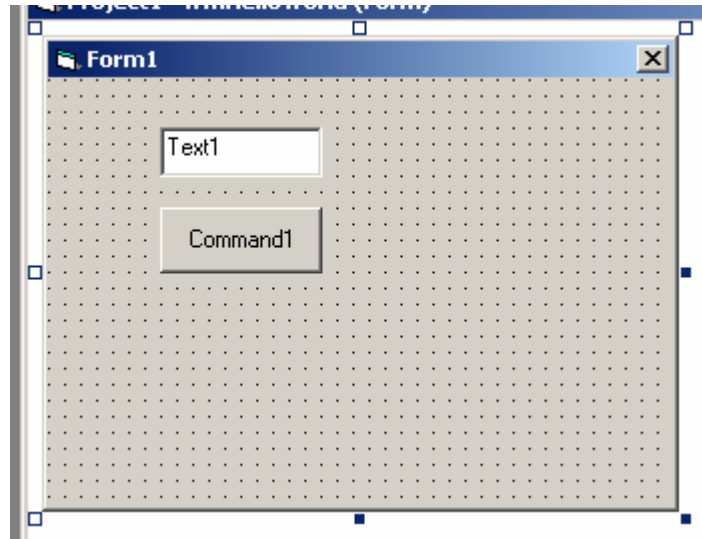
Visual basic applications are essentially groups of “forms”. A form is window, on which GUI widgets (like buttons, text boxes, graphics) can be placed. Projects can contain multiple forms, but one “main” form will load to start the project.

When you enter VB, select “standard exe”. This will create a new project with a blank form called *Form1*. Note several areas of the IDE (see below): on the left, we have the object palette. The object palette allows you to select new objects to be drawn on your form. In the top right is the Project Browser, which shows the forms that make up your project. In the middle on the right is the Properties Browser, to set various properties of the various objects. The Form Layout window in the bottom right allows you to place your forms on the screen (they will go to their appointed positions at runtime).



### 5.1.2 Adding GUI Widgets

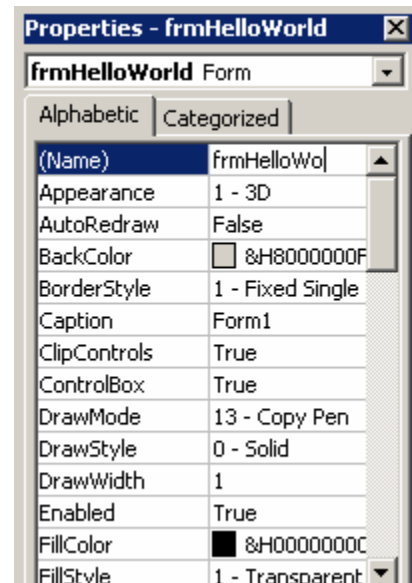
We'll start by adding a text box to the form. Do this by double-clicking the text box icon in the object palette (the right column, second from the top). You will then see a text box appear on your form. Next, add a command button (this is directly below the text box icon in the palette).



Press the “run” icon (labeled like the control on a VCR) at the top of the screen (pressing “F5” on the keyboard also runs the project). You should see your application load on the screen. Though you can enter text in the text box and click the command button, nothing actually happens yet, since we have not specified their behavior. Stop the project (by clicking the “stop” button at the top of the IDE).

### 5.1.3 Editing Properties

You should now have three objects in your application: the form (*Form1*), the text box (*Text1*), and the command button (*Command1*). Objects in VB have properties that can be edited at both design time and run time. First, we'll change the names of the various elements. Click on the form. Its properties should now be listed in the Properties Browser on the right side of the screen. The top-most property is (*name*). Change this to *frmHelloWorld*. Next, change the name of *Text1* to *txtInputBox*, and *Command1* to *cmdAlert*. We prefix the object names with the three-letter code for their type by convention (it comes in handy when building large applications). Let's also change the *BorderStyle* of *frmHelloWorld* to “fixed single” (note also “none” as an option – this may come in handy when you build future prototypes). Next, change the colour of the text in *txtInputBox* by changing its *ForeColor* property.



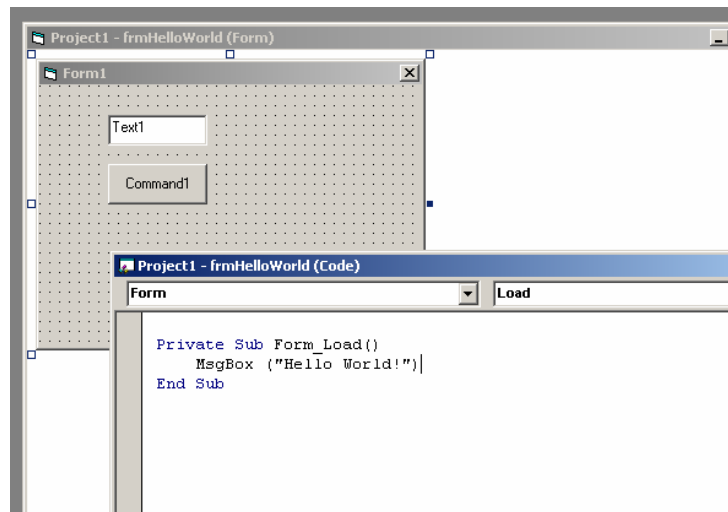
Any time you wish to change the appearance of a GUI widget or other object, this will be done through its properties.

### 5.1.4 Adding Behaviour to the Objects

We'll now get the program to actually do something! VB is event driven, which means that methods run when some action is taken. For example, there are methods that can run when a mouse is moved over a GUI widget, or on the tick of every second of a timer, or on the load of a form. Let's write a couple.

Double-click on the form. This will bring up a new window for coding, with a method pre-written with the header `Private Sub Form_Load()`. We'll behaviour to the form when it loads. Specifically, add this line:

```
MsgBox ("Hello World!")
```

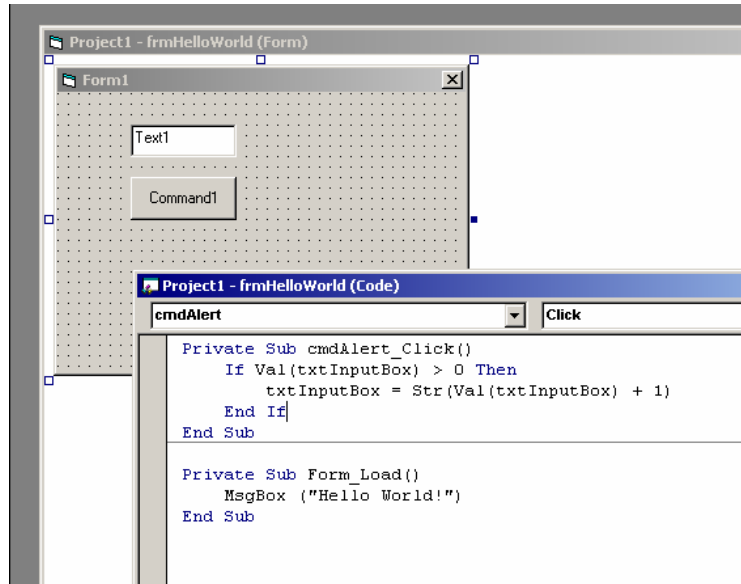


This will run the function (sub routine) `MsgBox` with the string parameter “Hello World”. The `MsgBox` sub routine in Visual Basic will bring up a prompt for the user. Test your application by running it (press “F5”). You will see a message box that you will need to dismiss before continuing the program. Click “OK”, and then stop the program.

At any time, you can get more information about a method by placing your cursor over its name and pressing “F1”. Do this for the `MsgBox` call in the `Form_Load` subroutine.

Let's add some more behaviour. Go back into the coding window and select the command-button from the left drop-down menu. By default, it will create a subroutine for the “Click” event. Note the other events available by clicking the right drop-down menu. We'll write code that will increment the value in `txtInputBox` as follows:

```
If Val(txtInputBox) > 0 Then  
    txtInputBox = Str(Val(txtInputBox) + 1)  
End If
```



This will check to see if *txtInputBox* currently has a numerical value. If it does, it will increment it by taking its numerical value, adding one, then converting it back to a String and save it in the box.

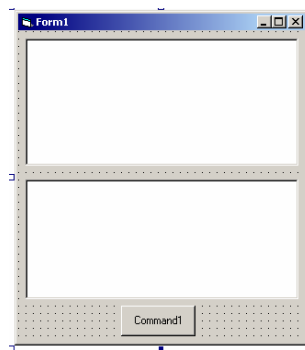


## 5.2 Building a Virtual Science Fair Prototype in VB

We'll now build a small part of the Virtual Science Fair (VSF) from the MOOsburg project from the course text (images on page opposite 203). The user interface of the VSF includes three components: an instant chat window, a map, and a project viewer; we'll build each of these below. Create a new project and call it "VSF".

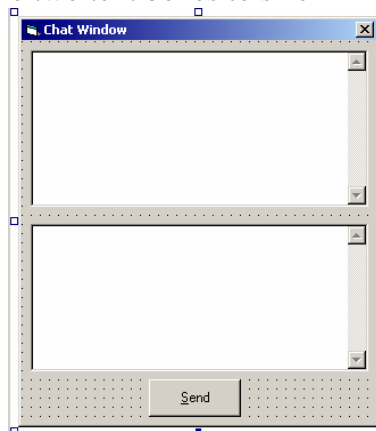
### 5.2.1 Instant Chat Window

Create a new form (or use the default *Form1*), and resize it to be roughly the same dimensions as the chat window in the VSF. Change its name to *frmChat*, and add two text boxes and a command button, with the names *txtChatLog* (top box), *txtChatInput* (bottom box), and *cmdSend* (command button).



We'll now edit a few properties to make the window a little more usable:

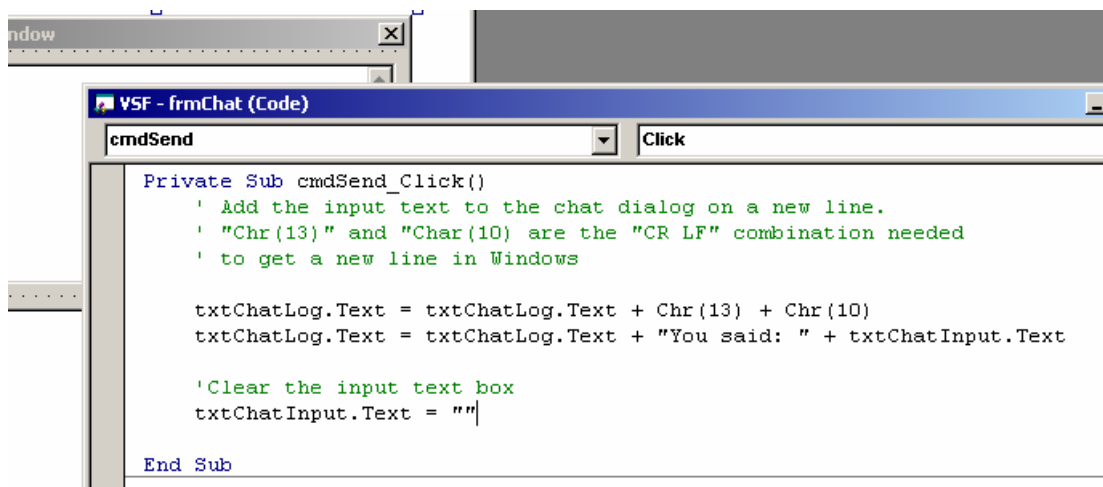
1. Change the *Caption* of *frmChat* to *Chat Window*
2. Change the *BorderStyle* property of *frmChat* to "1 – Fixed Single"
3. Change the *Caption* of *cmdSend* to "&Send" (the "&" is what sets which letter the user can use with the "alt" key to initialize the *click* event)
4. Change the *Multiline* property of the two text boxes to True
5. Change the *ScrollBars* property of the two text boxes to "2 – Vertical".
6. Change the *Font* of the two text boxes to size 12



The basic idea behind the Chat Window is that the user will type text into the bottom text box, click the Send button, and the text will be sent to other chatters. The running dialog of the chat will appear in the top text box. We'll now add some depth to our prototype to simulate the chat.

When the user clicks "Send", we want the text from the input text box to be copied into the dialog chat box. So, we'll add code to the "Click" event of the `cmdSend` button to make this happen. Specifically, when the user clicks "Send", each of the following will happen:

1. The text from the bottom text box will be added to the top on a new line, with a "You said:" prefix
2. The input box will be cleared.

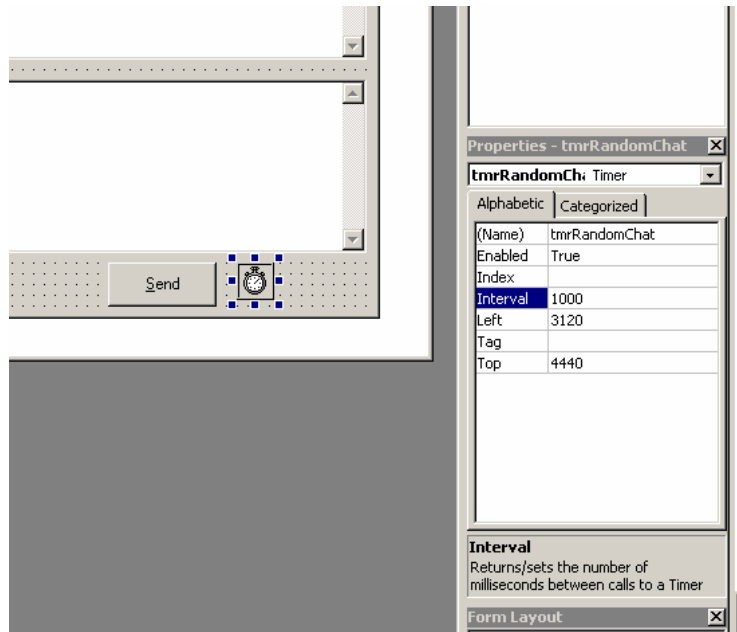


Try using your chat window to enter a few lines of text into the chat dialog. You'll notice that you have to repeatedly click the Send button, then click again into the input text box to type another message. You can save your user some clicks by setting the focus back to the input text box after they click "submit". Simply add this line:

```
txtChatInput.SetFocus
```

To better simulate a realistic chat, we'll add some dialog from a fictitious user. To accomplish this, we'll add a "Timer" object (on the Object Palette, this looks like an analog stop-watch) to the form. This object is invisible to the user, and allows you to have some code run at given intervals. Set the Timer's properties as follows:

1. (Name) to `tmrRandomChat`
2. Interval to 1000 (this is the amount of time between each running of your code, in milliseconds)



Now, we'll add the code to the timer to insert the phrases. Double-click on the timer object to access its *Timer()* method (this event occurs every *n* milliseconds, where *n* is the value of the timer's *Interval* property). We'll define an array to store the possible phrases and populate it with the phrases that our fictitious user will spout.

```

VSF - frmChat (Code)
tmrRandomChat Timer

txtChatLog.Text = txtChatLog.Text + Chr(13) + Chr(10)
txtChatLog.Text = "You said: " + txtChatInput.Text

'Clear the input text box

txtChatInput.Text = ""

txtChatInput.SetFocus
End Sub

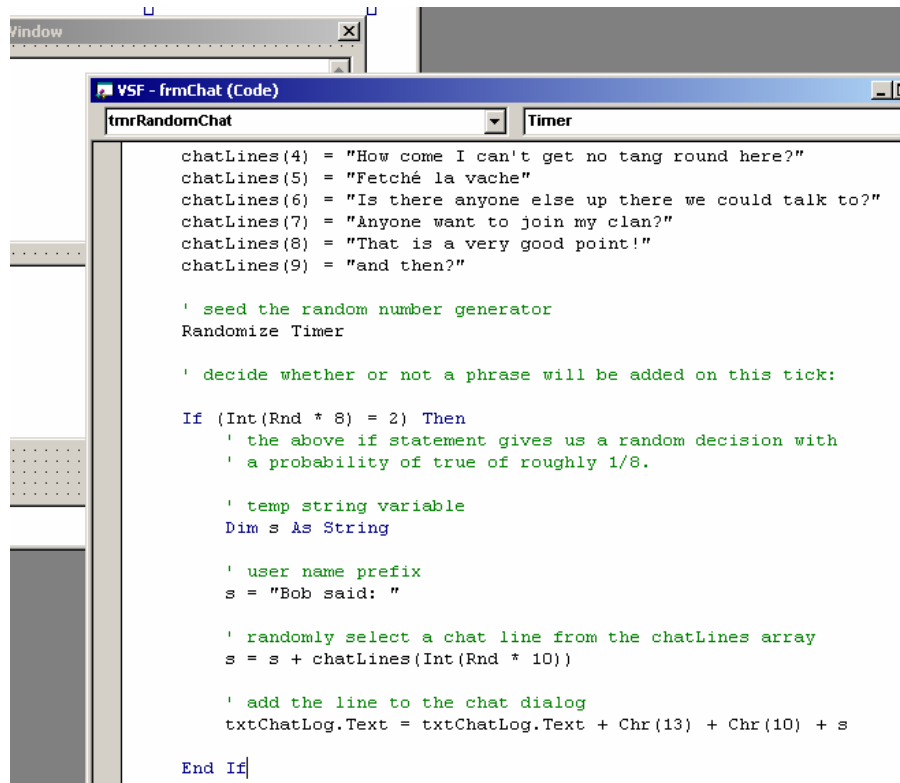
Private Sub tmrRandomChat_Timer()

' create an array of strings
Dim chatLines(0 To 9) As String

' populate the array
chatLines(0) = "Have you seen my marbles?"
chatLines(1) = "The Virtual Science Fair sure does rock!"
chatLines(2) = "I'm having a great time"
chatLines(3) = "I'm sentient, honest!"
chatLines(4) = "How come I can't get no tang round here?"
chatLines(5) = "Fetché la vache"
chatLines(6) = "Is there anyone else up there we could talk to?"
chatLines(7) = "Anyone want to join my clan?"
chatLines(8) = "That is a very good point!"
chatLines(9) = "and then?"

```

Now, we'll add the code to randomly pick and insert a phrase. So that the interval between phrases is random, and seems more realistic, we'll set the probability of inserting a phrase on any given tick of the timer to be 1/8. Here's how:



```
chatLines(4) = "How come I can't get no tang round here?"
chatLines(5) = "Fetché la vache"
chatLines(6) = "Is there anyone else up there we could talk to?"
chatLines(7) = "Anyone want to join my clan?"
chatLines(8) = "That is a very good point!"
chatLines(9) = "and then?"

' seed the random number generator
Randomize Timer

' decide whether or not a phrase will be added on this tick:
If (Int(Rnd * 8) = 2) Then
    ' the above if statement gives us a random decision with
    ' a probability of true of roughly 1/8.

    ' temp string variable
    Dim s As String

    ' user name prefix
    s = "Bob said: "

    ' randomly select a chat line from the chatLines array
    s = s + chatLines(Int(Rnd * 10))

    ' add the line to the chat dialog
    txtChatLog.Text = txtChatLog.Text + Chr(13) + Chr(10) + s

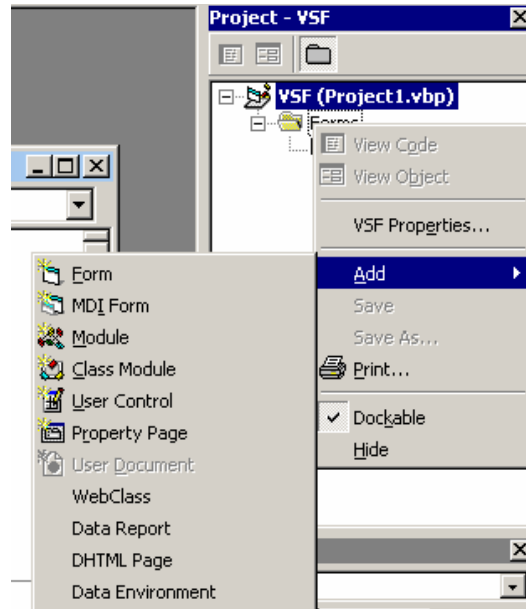
End If
```

*Rnd* is a random variable with an (effectively) uniform distribution in the range [0,1), and can be used any time you need a random number generated. The *Int()* function takes the floor of a number (eg: *Int(5.5)* is 5 ).

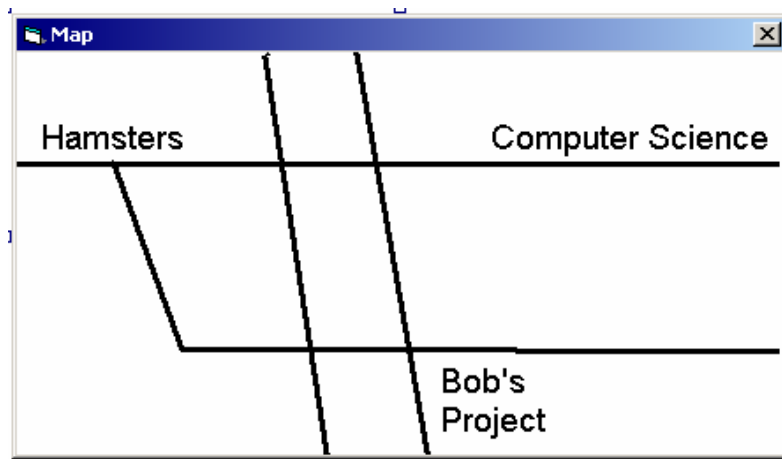
We now have a prototype of the chat client with a reasonable amount of depth.

### 5.2.2 Building the Map & Project Viewer

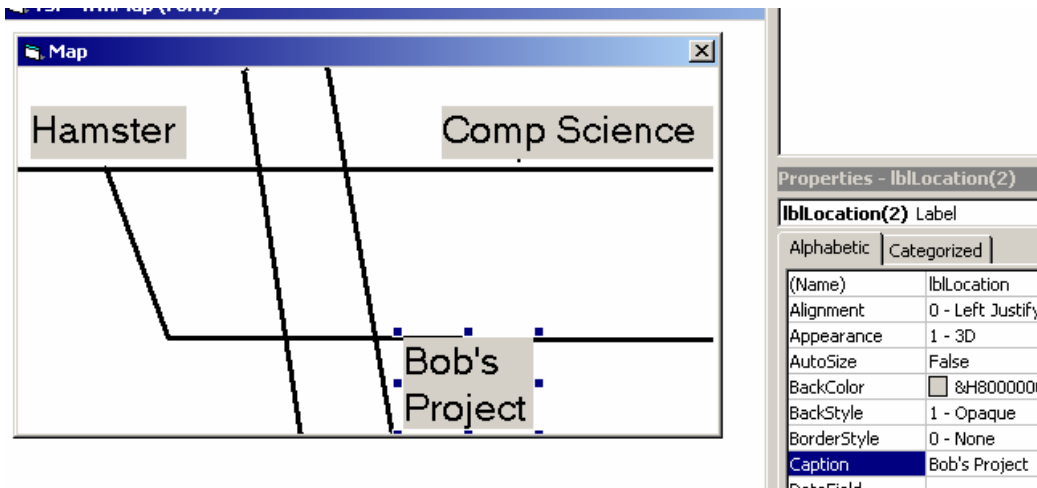
The map in the VSF allows users to click a location, and have the project viewer change to display that location. Create a new form that will serve as the map window, by right clicking on “Forms” in the Project Browser, then “Add”, then “Form”.



Change this form’s name to *frmMap*, and set its caption to “Map”, and add a Picture Box object. (top right corner of the palette). The map used below was drawn in Windows Paint, but the picture box can display any image. Set it by setting the *Picture* property.



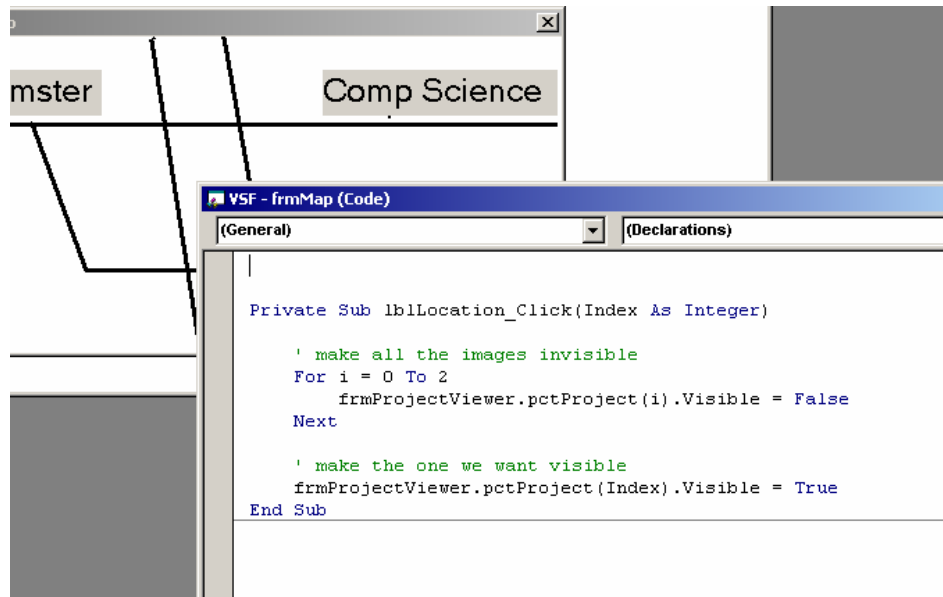
In order to easily allow someone to click on the map, we’ll put label boxes over each of the locations, and add code to their *click* events. To do this, add the Hamsters label first, setting the font as you might like, and set its name to *lblLocation*. To draw the next box, copy the “Hamsters” label, and paste it onto the form. VB will ask you if you’d like to create a “control array”. Click “Yes” (more on this later).



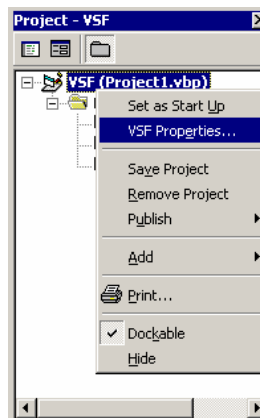
Now we'll add another form, *frmProjectViewer*, that will display the different projects when an area of the map is selected. On this form, we'll again create a control array, this time of picture boxes. Create the first picture box, call it *pctProject*, and load the picture of the hamster there. Then, copy & paste it, answer "yes" to the dialog box when it asks you if you wish to create a control array. Load the three different pictures into the three different picture boxes so that the image for each has the same index for the labels in *frmMap* (that is, if Hamster is *lblLocation(0)*, then make the picture of the hamster in *pctProject(0)*). Arrange the three image boxes on top of each other. Set each of the picture boxes to be invisible initially (set their "Visible" property to "False").



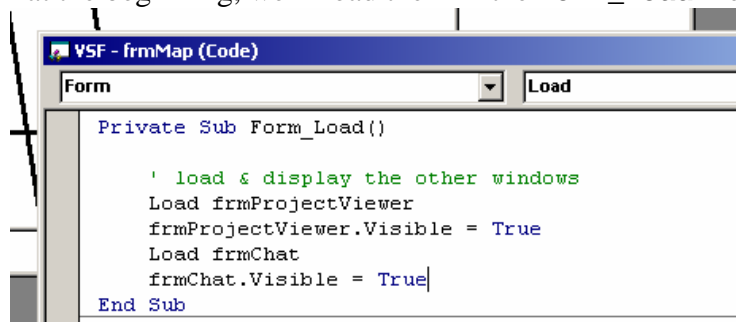
Now we'll add the code to the *frmMap* to change the images as the user clicks a location. Note the way that objects in another form are accessed using the ".":



Note we can access the different *pctProject* objects using their index, and that the *Index* parameter to the “Click” method tells us which of the *lblLocation* objects was clicked. All that’s left to do now is to pick which of our forms will load by default when the project loads. Since the map is the central component of the project, we’ll make it our main form. Do this by changing startup object to *frmMap*, under the properties of the project.

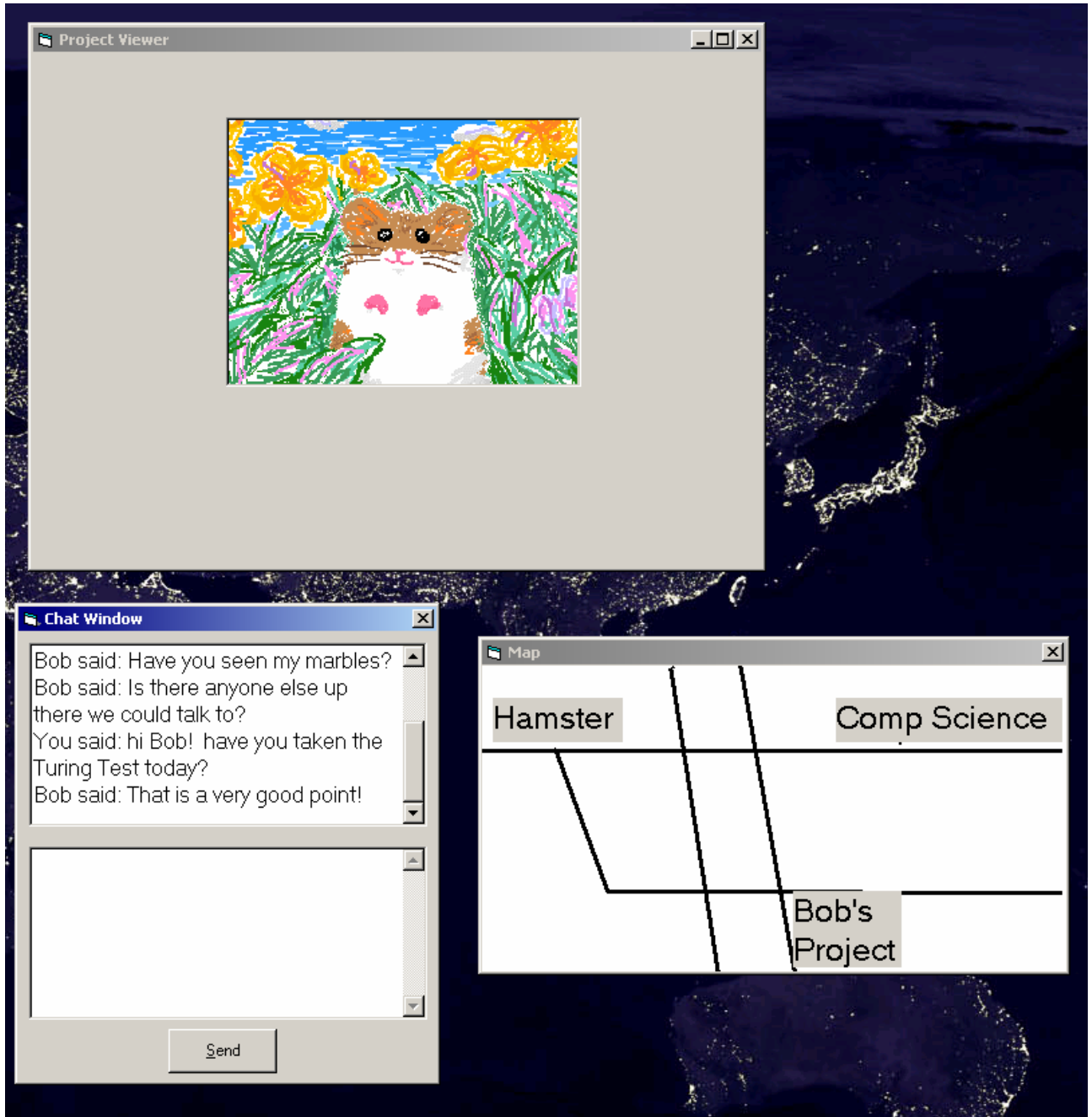


So now, when someone runs your prototype, it will load only the *frmMap* form. You will need to load the other forms yourself, and make them visible. Since we want all our forms to be open at the beginning, we’ll load them in the *Form\_Load* method in *frmMap*:



### 5.2.3 Completed Prototype

You now have a complete prototype! The finished project, when run, should look something like this:





## **5.3 Suggested Readings for VB**

### **5.3.1 Online/Free Help:**

Installed with Visual Basic is the Microsoft Developers' Network documentation. This includes complete lists of properties, functions, and Objects in VB. Pressing "F1" in the code on any keyword will bring up the MSDN documentation for that keyword. I strongly encourage you to use the MSDN documentation in building your prototypes.

### **5.3.2 Books:**

*Programming Microsoft Visual Basic 6.0* by Francesco Balena

*Microsoft Visual Basic 6.0 Reference Library* by Microsoft Corporation

## 6. Building a Dreamweaver Prototype

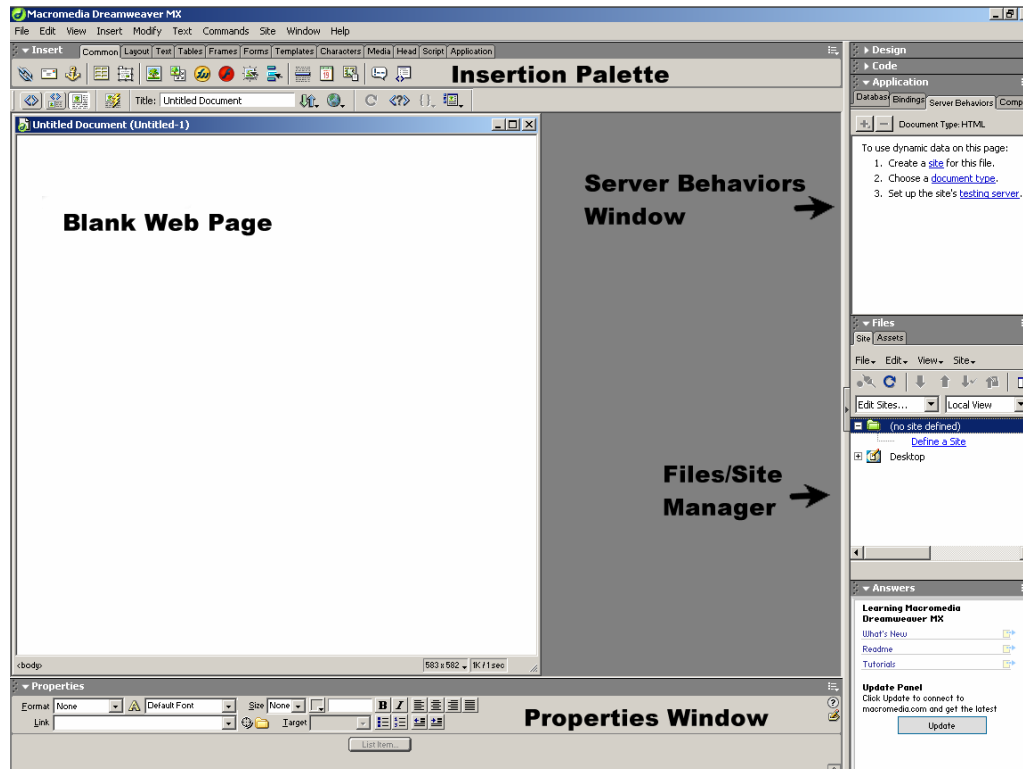
As an introduction to Macromedia Dreamweaver MX, we will first build a “Hello World” application, while giving an overview of the tool. Next, we will develop a prototype of the MOOsberg Virtual Science Fair in Dreamweaver. Lastly, you will find a list of suggested readings.

### 6.1 Hello World Application

#### 6.1.1 Overview

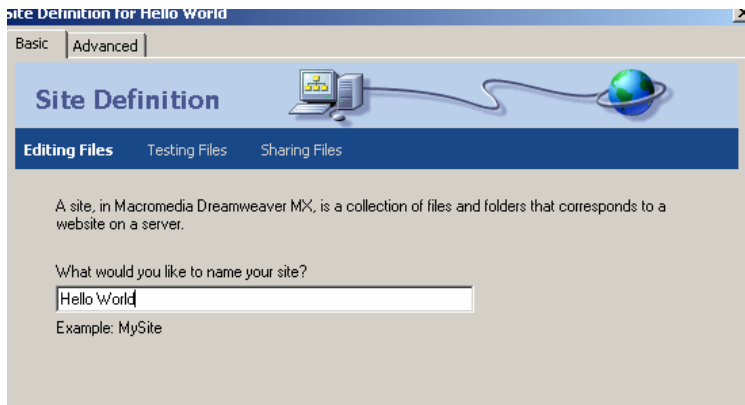
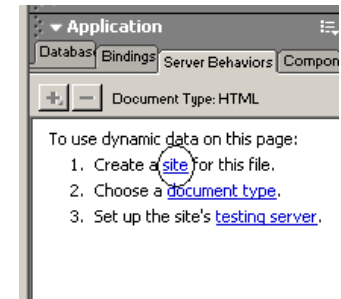
Dreamweaver projects are web sites, and pieces of the project are web pages. Dreamweaver allows you to build web pages as easily as you build word processing documents, hiding many of the complexities (such as HTML tags) that underlie the documents. Note that very complicated projects are possible, but the sites we develop in this document, as you should for 318, will be basic HTML/JavaScript pages.

When you start Dreamweaver, you get a blank page that will be your first web page in this project. Note the different areas of the workspace (see below). The Insertion Palette allows us to add objects to our web page. The properties window allows us to edit properties of the objects, the Server Behaviors window edits the project, and the files/site manager window is where the pieces of the project (web pages) are managed.



## 6.1.2 Defining Your Site

When building your prototype in Dreamweaver, the first step is to setup your project. This is done by building the “site”, which is selected from the Server Behaviors Window. We’ll call our first site “Hello World”, and it will contain only one page, *index.html*.

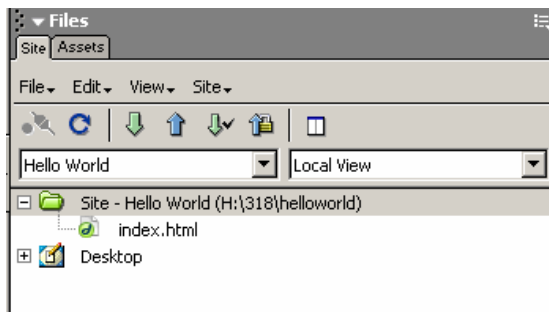
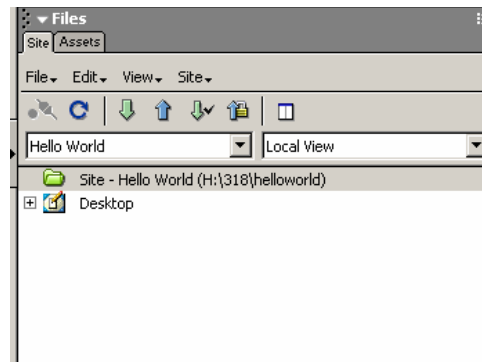


Use the wizard to create the Hello World site, answering as follows:

1. “No” to the question about technology in part 2
2. “Edit local copies on my machine...” and “h:\318\helloworld” in part 3
3. “Local/Network” connection and “h:\318\helloworld” for remote folder in part 4
4. “no” to checkin/out question in part 5

You’ll now see that you have a site setup:

We’ll save the currently blank web page as *index.html* in our site (*index.html* is the page that web servers display by default). Once you save the page, you’ll see that it is listed as part of the site:



We’re now ready to start our prototype!

### 6.1.3 Adding Content

Remember that as you add content to your web page, the way it looks in Dreamweaver is not necessarily the same as it will look in any given browser. Further, how it looks in Internet Explorer is different than how it will look in Netscape. You should also come to an agreement with your TA as to which browser they'll be using to view your prototype.

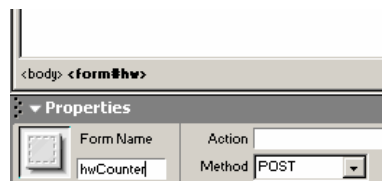
Adding content to a web page in Dreamweaver is very simple. Start by adding text, "Hello World!", and adjust its *Format* (under the Properties Window) to "Heading1". The basic types of text in an HTML document are headings & paragraphs. Marking something as a heading tells the browser to render that text in a special way (usually by increasing the font size, and possibly by using a different font).



Next, we'll insert some GUI objects. In HTML, GUI objects are elements of forms, so we'll need to insert a "form" before we can start to add the GUI widgets. Click the "Forms" tab on the Insertion Palette, and click the "Form" button.



All elements of the document must have a name, set the name of the new form to *hwCounter*.



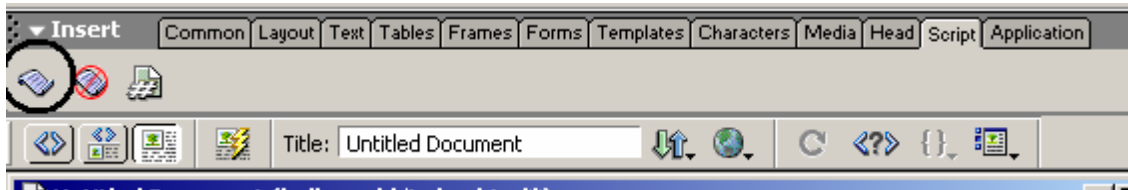
Now, add a text field named *countBox* and a button called *increment* (change the *increment*'s caption to match). Both these widgets are in the Insertion Palette.



### 6.1.4 Adding Behavior

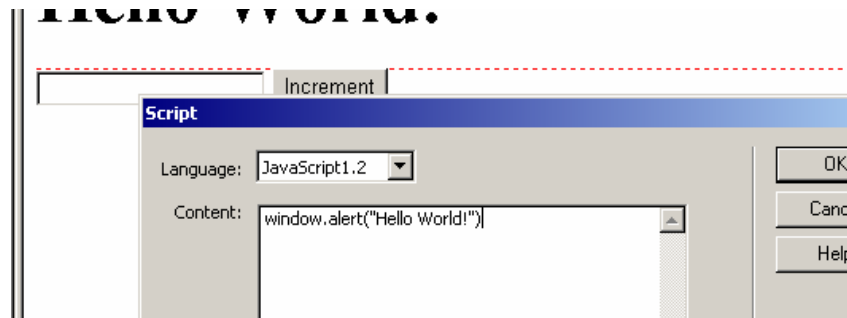
To make the increment button increment the text field, we'll need to add some script to the document. There are several scripting languages supported by the various browsers, but the closest to being standard is Java Script, so we'll use that.

Let's start by defining a script that will alert the user to the content of the page when they load it. To do this, add a Script object to the document by selecting "Script" from the Insertion Palette, then the "Script" icon.



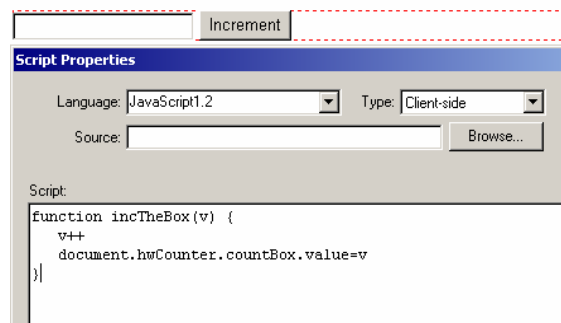
**Note that HTML is parsed in order**, so if you put the script at the top of the page, it will be run before opening your document, if you put it at the bottom, your document will be loaded before the script is run.

We'll make the script alert the user by asking the browser to open an alert window, like this:

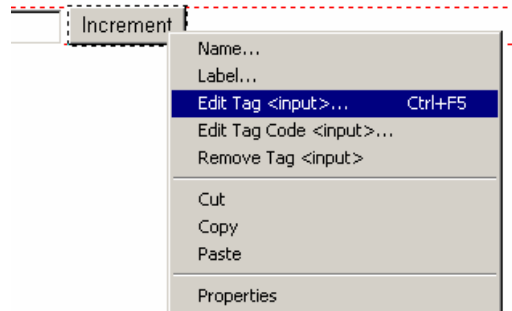


We'll now add script elements to the GUI widgets to increment the text box. First, we'll write a function that takes a number as parameter, and sets the text box to be one greater than that number. Do this by adding another script at the very top of the document that looks like this:

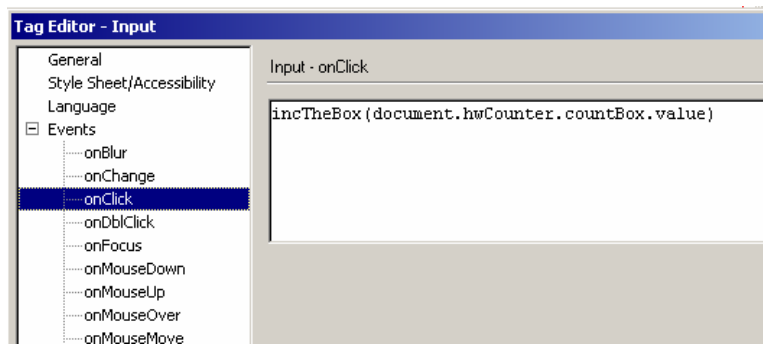
## Hello World!



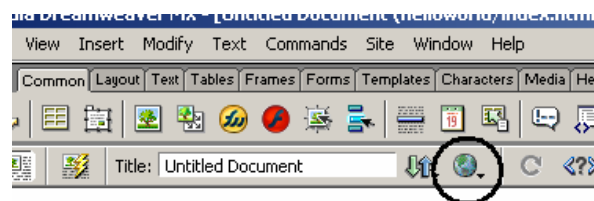
Now, all that needs to be done is to have the command button call this function when it is clicked. To do this, we need to add the `onClick` property to the command button. To access this property, right-click the “increment” button and select “Edit Tag <input>”.



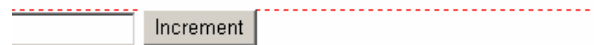
Then, select the `onClick` event from the “Events” list, and add the code to call the function, passing the text field’s current value as a parameter:



We now have a complete “Hello World” web page. Preview the page to see what it will look like in a web browser:



ello World!



## 6.2 Accessing Objects in JavaScript – the DOM

The Document Object Model (DOM) is the hierarchy of objects in a web page. Any element of a web page can be addressed through its DOM address. DOM addresses were seen above, for example, in the line:

```
document.hwCounter.countBox.value = v
```

*document* is the top level object that refers to the HTML document in the current window, *hwCounter* was the name of the form, the *countBox* was the name of the text field, and *value* is a property of the text field.

Getting the right DOM address can sometimes be frustrating, but remember that the top level objects are always either “window” or “document”, and you proceed down into embedded object through embedded object. Don’t think of these as references, but rather as being like the parts of a phone number: first the area code, then the exchange, then the local number, each getting more specific.

The complete DOM can be found online:

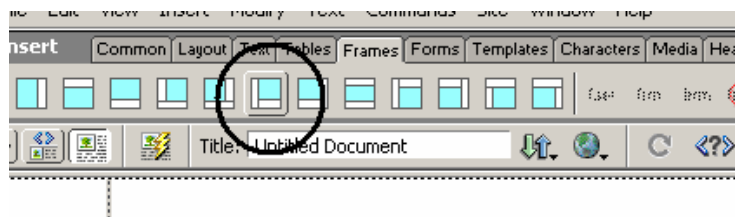
<http://www.w3.org/TR/2002/CR-DOM-Level-2-HTML-20020605/>

### 6.3 Building a Virtual Science Fair Prototype in Dreamweaver

We'll now build the VSF in Dreamweaver. Create a new site and call it "VSF" (as before, click the "site" option in the Server Behaviors window). The user interface of the VSF includes three components: an instant chat window, a map, and a project viewer; we'll build each of these below. We'll use HTML frames to separate the pieces, and have different HTML files represent each of the pieces.

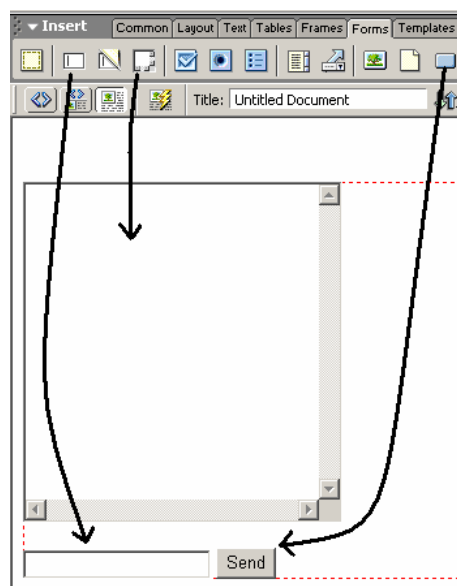
#### 6.3.1 Defining the Frameset

Frames work by having one HTML page, commonly called the "frameset", define the frames and what goes inside them. Rather than tagging specific content for the frames, though, the HTML file that defines them merely places pointers to other HTML files that will occupy the frames. To create your frameset page, select the "Frames" tab in the Insertion Palette, then select the frameset that best reflects the VSF layout:



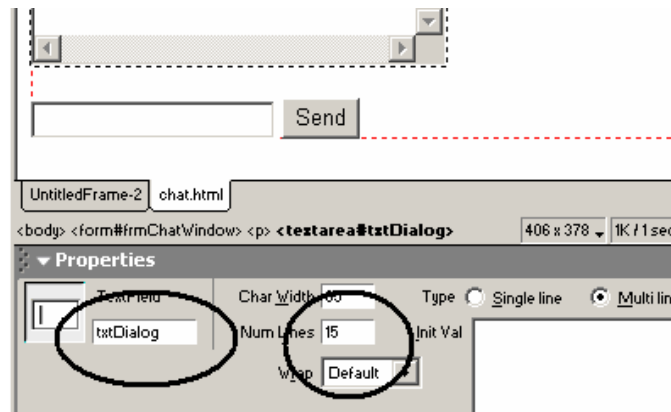
#### 6.3.2 Building the Chat Window

Now that we've defined the frameset, create a new HTML file that will become the chat window. Place a form onto it and name it "chatWindow", and on the form place a textarea, a textfield, and a button, to roughly approximate the look of the VSF chat window:

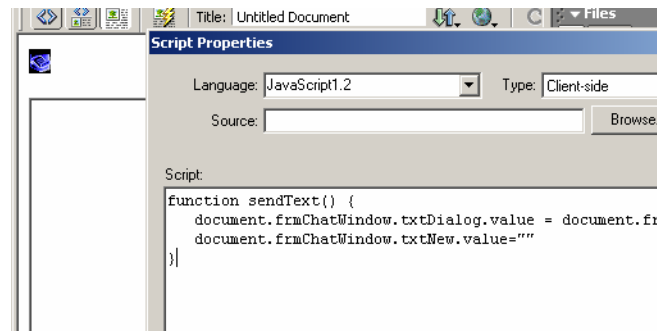




Adjust the properties of each of these objects, and give them the names *txtDialog* (text area), *txtNew* (text field), and *cmdSend* (button). You can adjust the height & width of the text area to make it look like the one above:



Preview this page in your web browser. It has the right look of a chat window, but we don't yet have the ability to actually do any chatting. To do this, we'll add some JavaScript to the file. The first script we'll write will be a function that will take the contents of *txtNew* and add them to *txtDialog* with a "You said:" prefix. This script will go at the top of the page:



The complete script is (note that the first line of the function wraps here, it must all be on one line in your script):

```
function sendText() {
    document.frmChatWindow.txtDialog.value =
        document.frmChatWindow.txtDialog.value + "You said: " +
        document.frmChatWindow.txtNew.value + '\n';

    document.frmChatWindow.txtNew.value=""
}
```

Save the file as *chat.html*. All that's left is to have *cmdSend* call this function when it is clicked. As before in the Hello World example, add the function call to the *onClick* event of *cmdSend*.

We'll now update the frameset to have the chat window in the left frame. Go back to your frameset page, and select the frameset (the easiest way to do this is to click the border between the left & right frames):



We'll now switch modes so that we can see the underlying HTML code of the frameset by selecting "Code and Design" from the "Views" menu. Change the contents of the first frame to "chat.html":

```
7
8 <frameset rows="" cols="275,*" frameborder="NO" border="0"
9   <frame src="chat.html" name="leftFrame" scrolling="NO" no
10  <frameset rows="*,80" frameborder="NO" border="0" framesp
11    <frame src="Untitled-1.htm" name="mainFrame">
12    <frame src="UntitledFrame-3.htm" name="bottomFrame" scr
13  </frameset>
14 </frameset>
15 <noframes><body>
16
17 </body></noframes>
18 </html>
```

A screenshot of a web browser window showing the frameset after the HTML code has been updated. The browser's address bar is empty. The frameset now consists of three frames: a top frame, a left frame, and a bottom frame. The left frame contains a chat window icon. The bottom frame contains a scrollable area with a red dashed border.

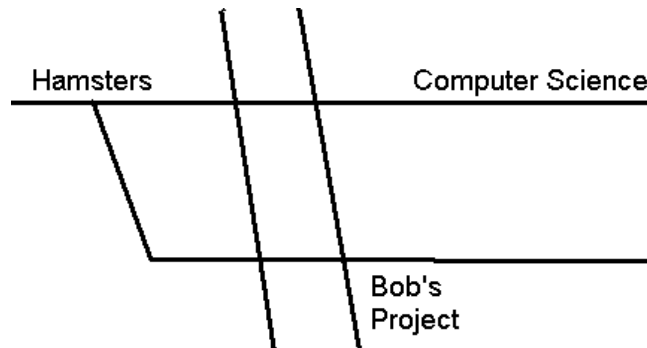
This will load the chat window in the left frame of the frameset.

### 6.3.3 Building the Project Viewer

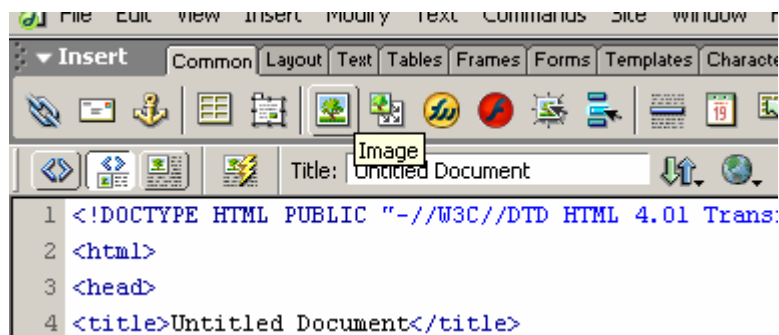
The Project Viewer window is simply a place where a particular science fair project can be viewed. Since the window is already setup for us (by the frameset), we need only setup different HTML pages for each of the projects we wish to simulate. In this prototype, we'll have three projects: Hamster, Comp Science, and Bob's Project. Make up pages for these projects and save each of them as *hamster.html*, *compScience.html*, and *bobs.html*. We'll setup the map so that when the user clicks on a particular project name, we'll get the appropriate HTML file.

### 6.3.4 Building the Map

For this part, we'll need a graphic that is the map of the science fair. You can draw one in Windows Paint and save it as a JPG. **Be sure to save it in the same folder as your HTML files.** Here's one I drew:

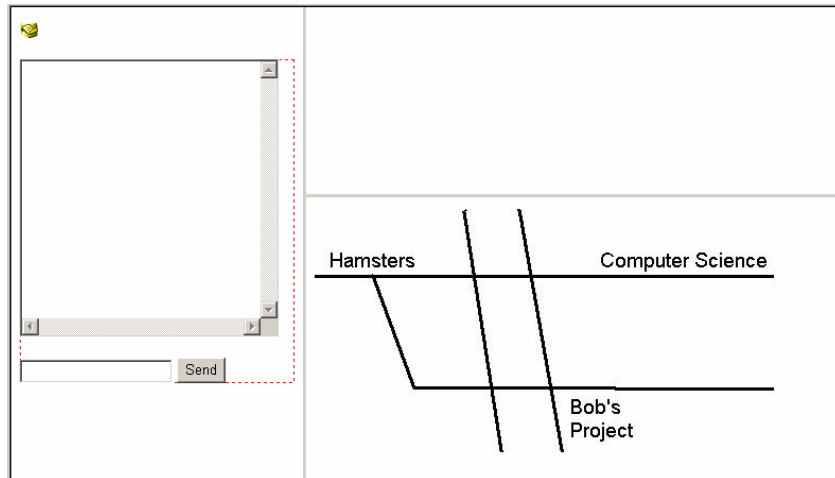


We'll now create a new html file, *map.html*, which will have only one thing on it: this map. Add the graphic to the new HTML file by clicking the Image button under the "Common" tab on the Insertion Palette:

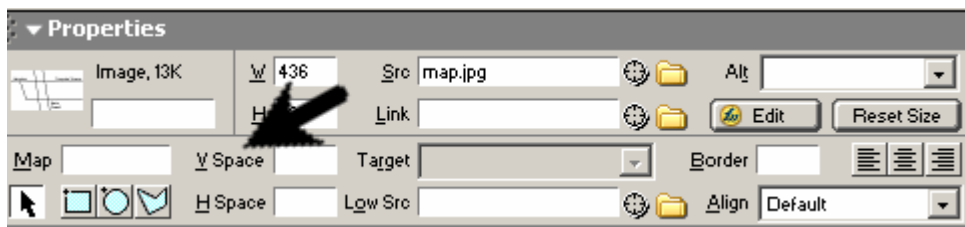


If you saved your map in the same folder as your HTML files, it will be easy to add. If you didn't, move it there now.

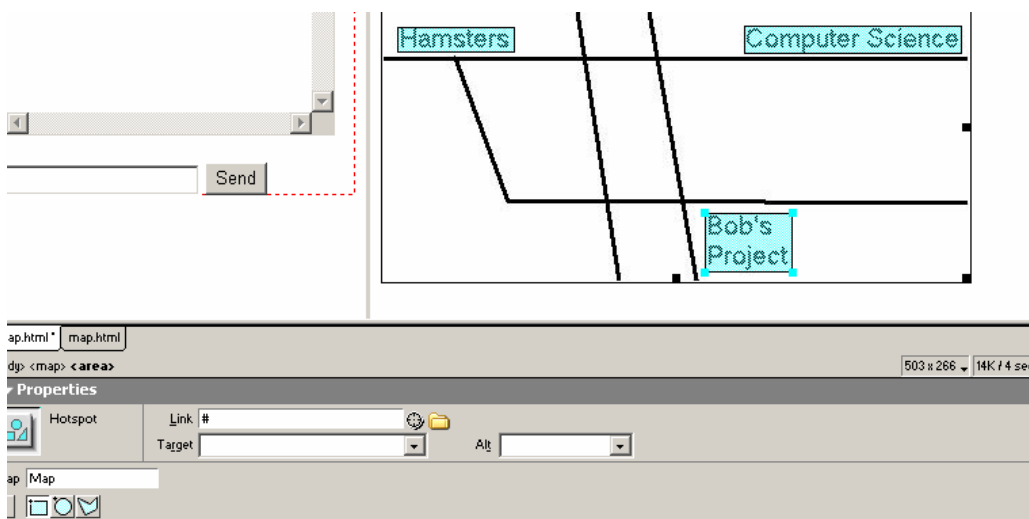
Save your HTML file, and add it to the bottom frame as we did for the chat window above. Your frameset should now look like this:



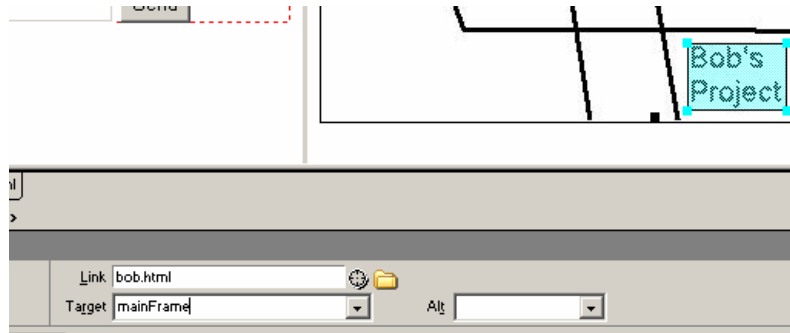
All that is left to be done is to add the ability to click on each of the names in the image map and have them load the appropriate html file in the main frame. We do this by creating an image map using the properties of the image (click on the graphic and note the properties that allow us to create an image map):



What the image map allows us to do is to create links within the image, so that when the user clicks on part of the graphic, it takes us to a particular location. Draw squares over the names of the projects in the map (using the square drawing tool in the bottom left corner of the image map tools):

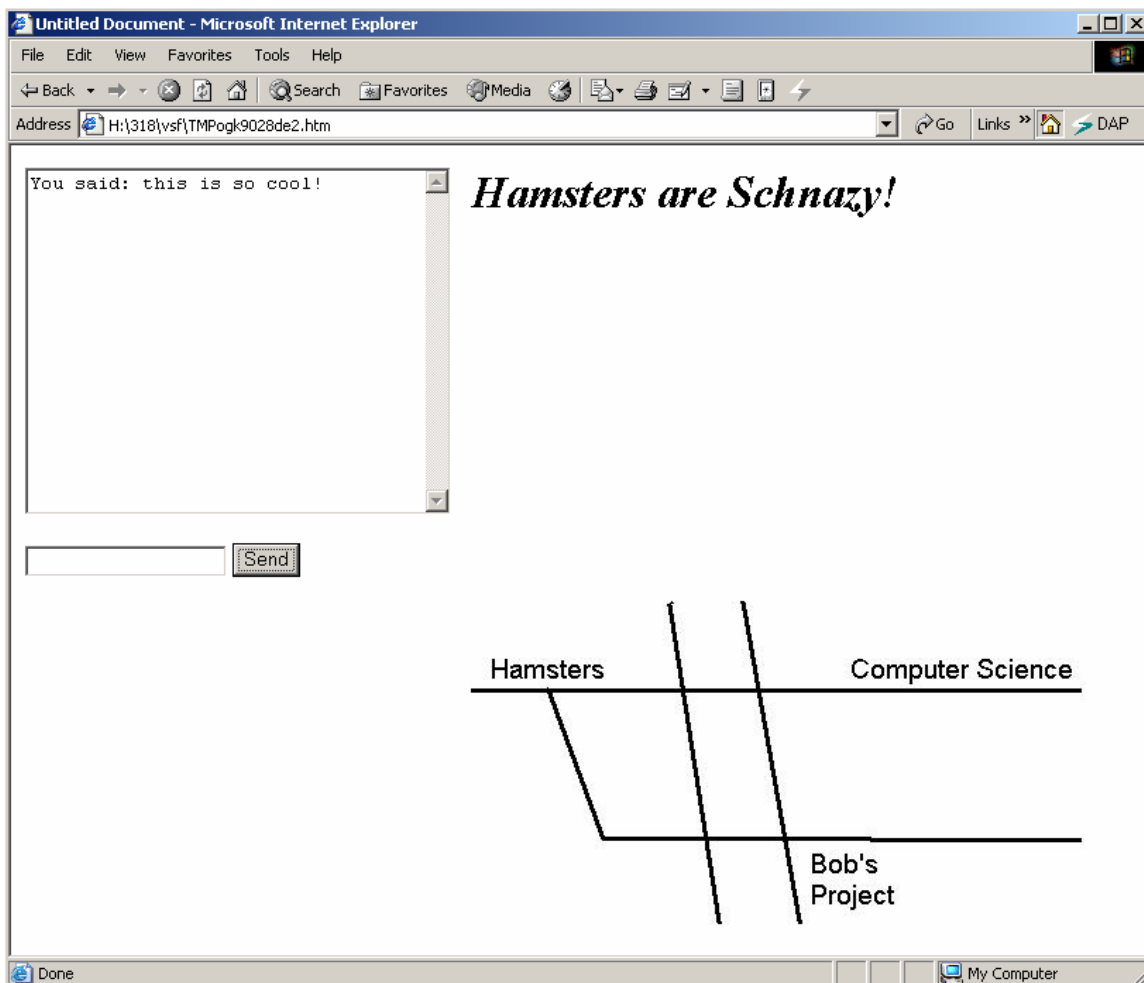


Set the links to be the appropriate HTML file (for the “Hamsters” project, the link is the *hamster.html* file you created previously, and so forth). You also need to set the *target* of each of the links to be the main frame of the frameset (so that when the user clicks a link, it loads the linked page into the main frame, and not into the map’s frame):



We now have a completed prototype! Preview your prototype in a web browser to see what it will look like.

### 6.3.5 Completed Prototype



## **6.4 Suggested Readings for Dreamweaver:**

### **6.4.1 Online/Free Help:**

Dreamweaver Specific Help:

The references built-in to Dreamweaver are excellent – following their lessons will help you to build very powerful prototypes.

Java Script Help:

<http://www.mozilla.org/js/> (java script home page)  
<http://www.geocities.com/SiliconValley/Station/4320/>

General HTML Help:

<http://www.w3c.org>

### **6.4.2 Books:**

*Macromedia Dreamweaver MX: Training from the Source* by Khristine Annwn Page

*Macromedia Dreamweaver MX for Windows and Macintosh (Visual QuickStart Guide)*  
by J. Tarin Towers

## 7. Building a Flash Prototype

As an introduction to Macromedia Flash MX, we will first build a “Hello World” application, while giving an overview of the tool. Next, we will develop a prototype of the MOOsberg Virtual Science Fair in Flash. Lastly, you will find a list of suggested readings.

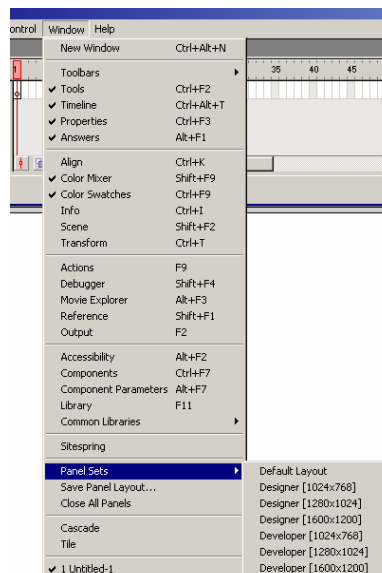
### 7.1 Hello World Application

Our Hello World application will give you a quick intro to prototyping, and will include both a “Hello World” message and a counter that increments on a button click, so you’ll see some basic deep functionality of the tool.

#### 7.1.1 Overview

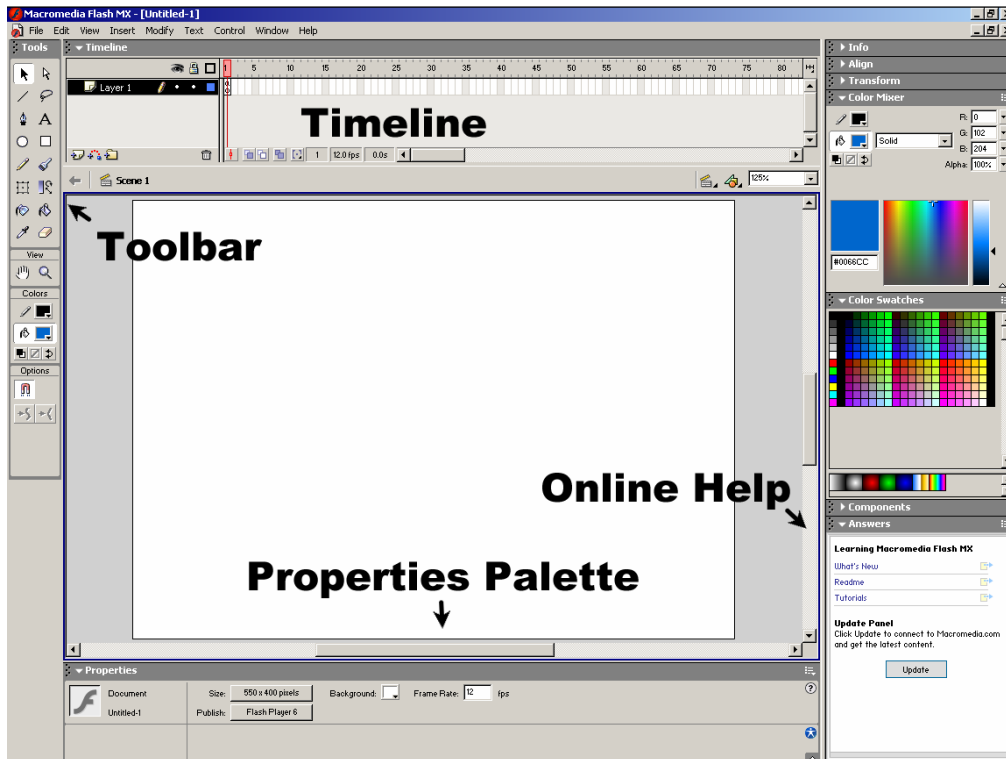
Unlike the previous two systems, Flash is not a programming language, and is not designed to build applications. It does have a scripting language, but Flash is a system for developing multi media presentations and interfaces, which makes it a great way to prototype, but a very different UI developing environment than you’re likely used to.

There are two different “views” in Flash – one designed for developers, which includes access to advanced features, and one for designers, which hides many of those.



We’ll use the “Designer” layout for this guide.

The basic components of the Flash desktop are the Toolbar, which allows you to add and edit objects, the Timeline, used for animation, the Properties Palette, where the properties of objects in the movie are edited, and the Online Help area.



### 7.1.2 Adding Content

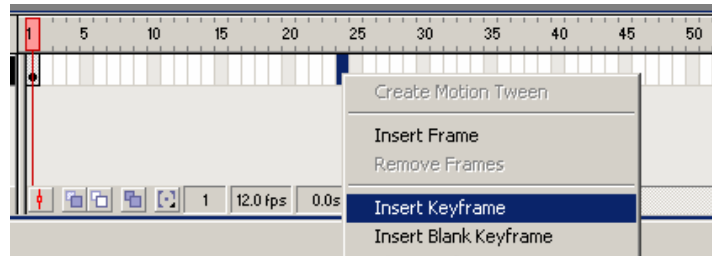
Adding content to a Flash movie is very simple. Use the text tool in the Toolbar to add “Hello” to your movie, and place it in the top-left corner. Next, add “World” to the bottom-right.



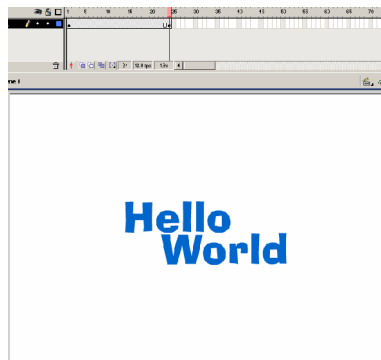


### 7.1.3 Understanding Animation: The Timeline

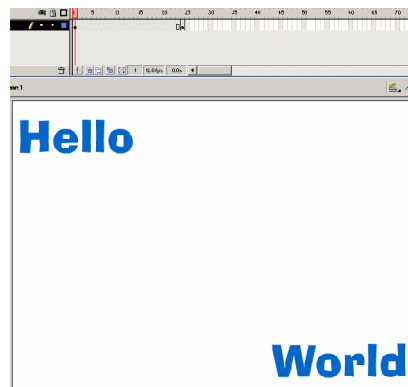
As you no doubt know, animation is achieved by having a series of still images, with each sequential image slightly different from the adjacent image. Each of these still images is called a “frame” of animation. The Flash timeline allows you to set the contents of the various frames of the movie. We’ll setup the animation so that “Hello” and “World” move together over the course of 2 seconds. We do this by adding a “key frame”, and moving the text together on that frame. The default movie speed is 12 frames per second, so our new key frame will be frame 24:



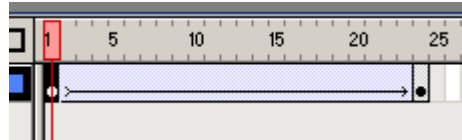
For our animation, by the time we get to frame 24, we want “Hello” and “World” to have moved to the center of the screen. While frame 24 is active (it’s the one with the red line through it on the Timeline), move “Hello” and “World” to the center of the screen:



Click on frame 1 – note that the text is where it was before. Moving the text on frame 24 does not affect the contents of frame 1:



Play your animation by selecting “Play” from the Control menu. You’ll see that the text stays in the corner for 23 frames, then jumps to the middle on frame 24. Obviously, this is not what you’d like to see – it would be much better to have the text move consistently between frames 1 and 24. This is done by a “motion tween”. Right-click on **frame 1**, and select “Create motion tween”. This will automatically create animation on the intermediate frames between key frame 1 and key frame 24:



You now would expect that the “Hello” and “World” would move together slowly over the course of 2 seconds. Play the animation (select “Control”, then “Play”), and see what happens:



Instead of getting the smooth transition we were expecting, the words seem to hover for a while, then jump to the middle. What happened?

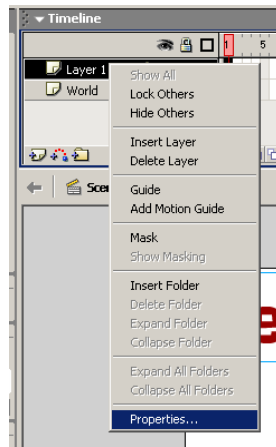
**Layers in Flash move as a whole.** “Hello” and “World” seem to be separate objects here, but together they make up a single layer in our movie. The motion tweens in Flash won’t move objects, but rather layers. To make the tween work, we’ll need to put the two words on different layers. Create a new animation, and place “Hello” and “World” in the appropriate corners:



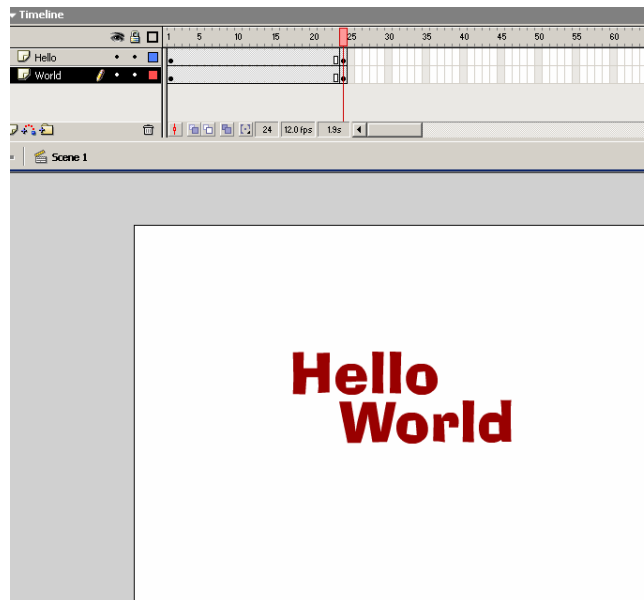
We now need to move one of the objects to a new layer. Do this by right-clicking on “World”, and select “Distribute to layers”. You’ll now see a second layer has been created:



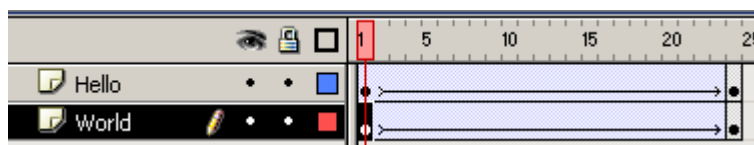
Rename “Layer 1” to “Hello”, to better match its content. Do this by right-clicking on the layer and selecting “properties”, then changing the name:



We’ll now setup the animation to have the text move-together. Once again, create a key frame on frame 24. **Note that you’ll have to do this twice, once for each layer**, since key frames are layer-specific. Select frame 24, and move “Hello” and “World” to the center of the screen:



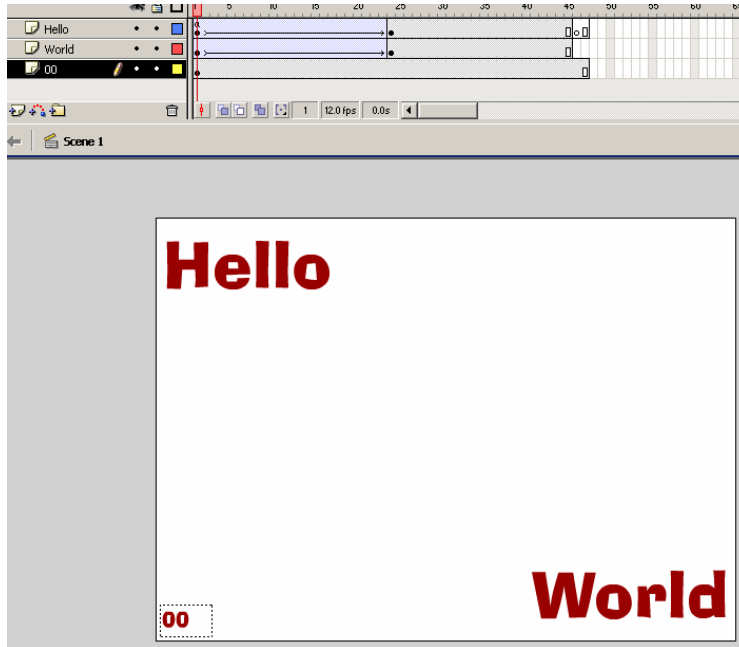
Once again, right-click on frame 1, and select “Create motion tween” (do this for both layers):



Now, play your animation – it works!

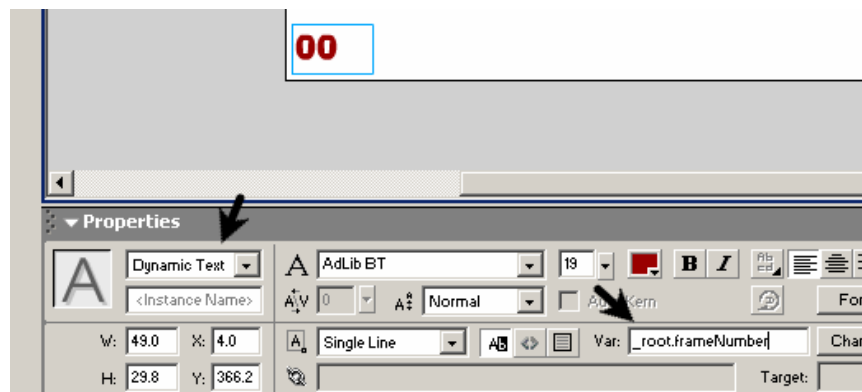
### 7.1.4 Adding Scripts

Flash has a built-in scripting language. We'll now use it to add a frame counter to the bottom-left corner of our movie. To do this, we'll need first to add a new text object in the bottom-left corner, with the contents "00" (simply use the text tool in the Toolbar). Put the new text on its own layer by right-clicking it, then selecting "distribute to layers":



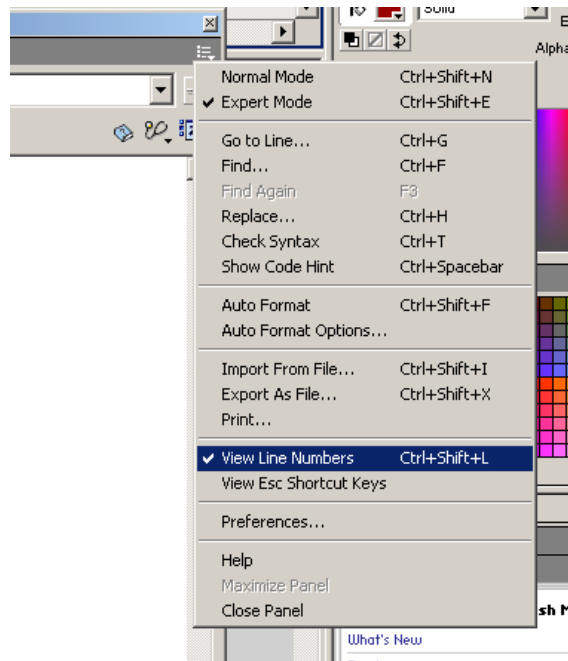
Rename the "00" layer as "Counter" for easier reference.

In order to be able to edit the object's contents in a script, we'll need to set a couple of its properties. In the Properties Palette, set its type to "Dynamic Text" and its *Var* to "`_root.frameNumber`" (we use `_root` as a prefix to the name because `_root` is the root object in the object hierarchy in Flash):

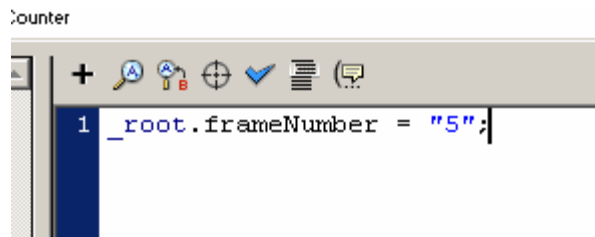


We'll now add an action to Frame 5 of *Counter* to update the contents of the counter to reflect the current frame. To do this, right-click on frame 5 in the *Counter* layer, and add

a keyframe. Next, right-click on keyframe 5 in the counter layer, and select “Actions”. In the Actions window, select the sub-menu in the top right, and ensure *Expert Mode* and *View Line Numbers* are both enabled:



We’ll now add code to set the content of the text box – simply type the code into the code window:



Repeat the procedure of adding keyframes & code to frames 10, 15, 20, and 24, each time setting the contents of *\_root.frameNumber* to the value of the current frame:



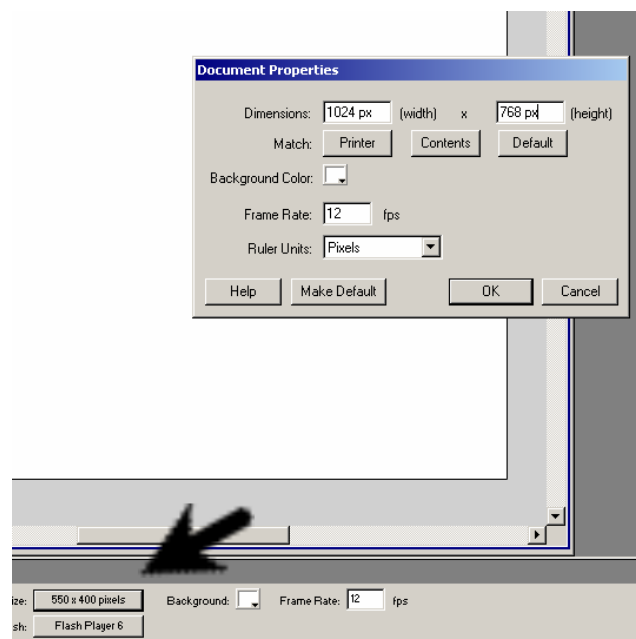
Test your movie by selecting “Control”, then “Test Movie” (note that scripts are not run when you select only “Play”).

## 7.2 Building a Virtual Science Fair Prototype in Flash

We'll now build a small part of the Virtual Science Fair (VSF) from the MOOsburg project from the course text (images on page opposite 203). The user interface of the VSF includes three components: an instant chat window, a map, and a project viewer; we'll build each of these below. Create a new project and call it "VSF".

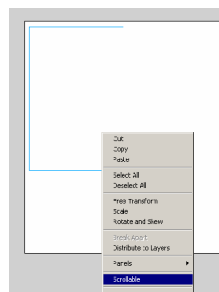
### 7.2.1 Setting up a Prototype

We'll now build a Flash movie to be our VSF prototype. Start by making a new movie, and saving it as "VSF". Set the size of the document to be the full size of your screen (assumed to be 1024 x 768):



### 7.2.2 Building the Chat Window

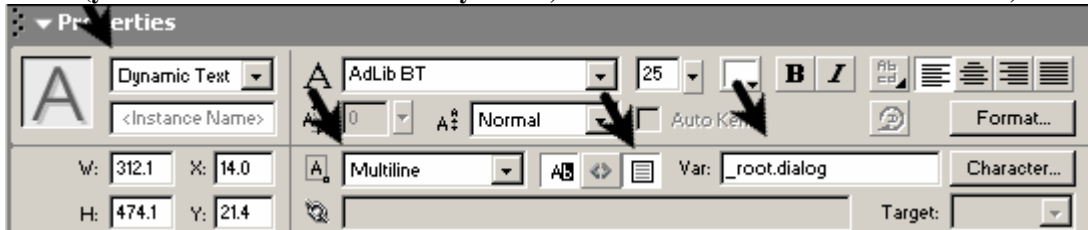
We'll now build the chat portion of the VSF. On the left side of the screen, we'll draw 3 objects: 2 text boxes (one to enter new text, one for running dialog) and a button. Draw the dialog text box first, making it tall and thin, like a chat window. Right-click on it, and make it "Scrollable":



Next, edit its properties as follows:

1. Make it a “Dynamic Text” window.
2. Set its name (*Var*) to “\_root.dialog”.
3. Set it to have a border.
4. Set it to be a “Multiline” box

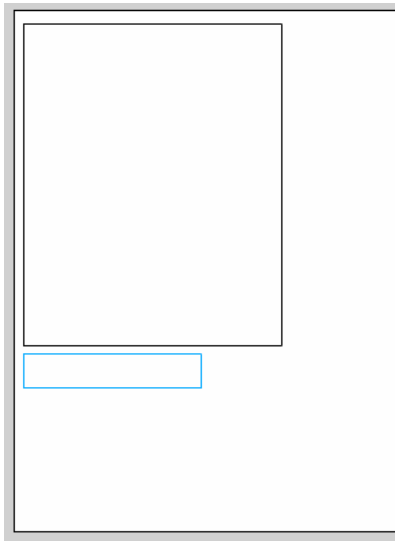
(you can set the font to whatever you like, but be sure that the colour is not white!)



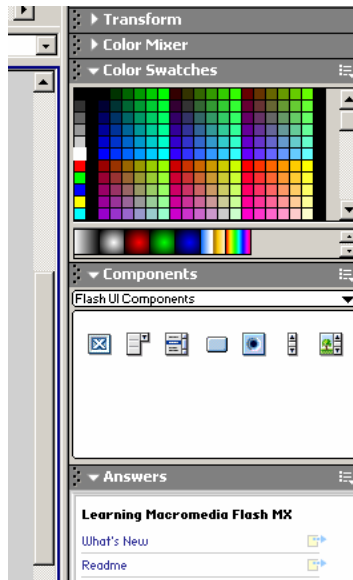
Next, we’ll add the text field where the user will type the message they wish to send. Draw the box below *\_root.dialog*, but a little narrower so there will room for the “Send” button. Set its properties as:

1. Make it an “Input Text” window.
2. Set its name (*Var*) to “\_root.inputMSG”.
3. Set it to have a border
4. Set it to be a “Single Line” text box

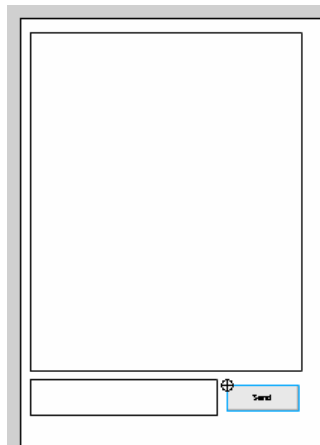
Your document will now look something like this:



Lastly, we’ll add a push button. Flash has several common GUI widgets pre-built. We’ll add a command button from its “Components Palette” (if you don’t see the Components Palette, click on its name on the right side of the screen to display it):



The push button is the fourth widget from the left. Drag and drop one onto your document, immediately to the right of `_root.inputMSG`. You can change the size and shape of the push button by right-clicking on it, and selecting *Scale*. Once it is positioned as you'd like, set its *Label* to "Send":



We now need to add code that will take the text from `_root.inputMSG` and copy it to `_root.dialog` (adding a "You said:" prefix). The *Send* button already has a lot of code attached to it. To get it to properly respond to a "click", we need to first write a function to move the text, then set it as the click event for the button.

To add a function to the button, right-click on it and select "edit". This will take you into a detailed view of the button, allowing you to change very low level aspects. We'll add a function by right-clicking on the first frame of the "Actions" layer, and clicking "Actions". You should see a LOT of code that was generated for you. Add a new function "sendMessage" to the top of this code:

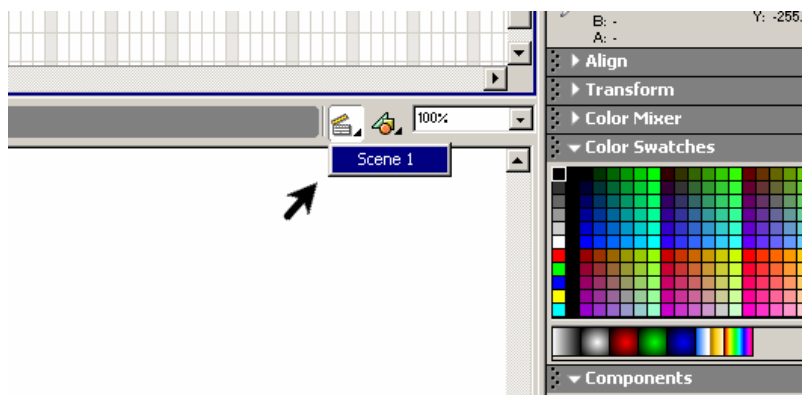


```

1 #initclip 1
2
3 sendMessage = function() {
4     _root.dialog = _root.dialog + "\n" + "You said: " + _root.inputMSG;
5     _root.inputMSG = "";
6 }
7
8 function FPushButtonClass()
9 {
10     this.init();
11 }
12
13 FPushButtonClass.prototype = new FLIComponentClass();

```

We're done editing the button, so close the actions window, and then switch back to the Document window by clicking these editor selector:



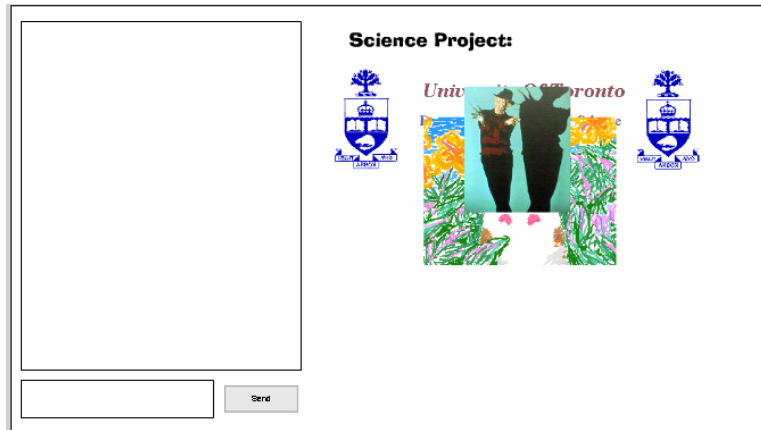
Lastly, we need to set our new function to be the method run when the button is clicked. Do this by changing the *Click Handler* property of the button to "sendMessage":



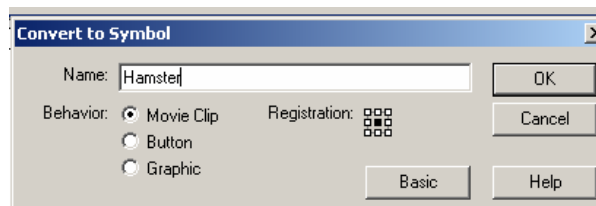
That's it! We now have a working chat prototype. Try running it by selecting "Control", then "Test Movie" (recall that "Play" will not run any scripts).

### 7.2.3 Building the Project Viewer

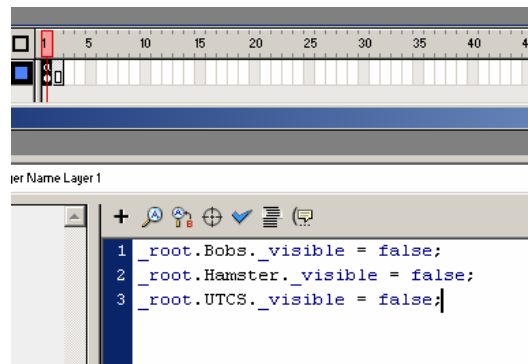
The Project Viewer in this case will be made up of images pasted directly onto the document, which will be made visible when the user selects the appropriate topic. Copy images onto the clipboard from any graphics application, and paste them into Flash. Here, we'll have images for a Hamster, Bob's, and U of T CS projects:



In order to be able to write a script to edit the properties of these images, they must be converted to movie clips (this doesn't mean we'll be adding animation, it is merely an internal flash designation). To do this, right-click on an image, and select "convert to symbol". Set them to be "Movie Clips", and give them the names "UTCS", "Bobs", and "Hamster":



We'll now add script to set each of the clips to be invisible when the movie starts. To do this, right-click on frame 1 of the only layer in your movie, and select "Actions". Write the code to make the clips invisible:

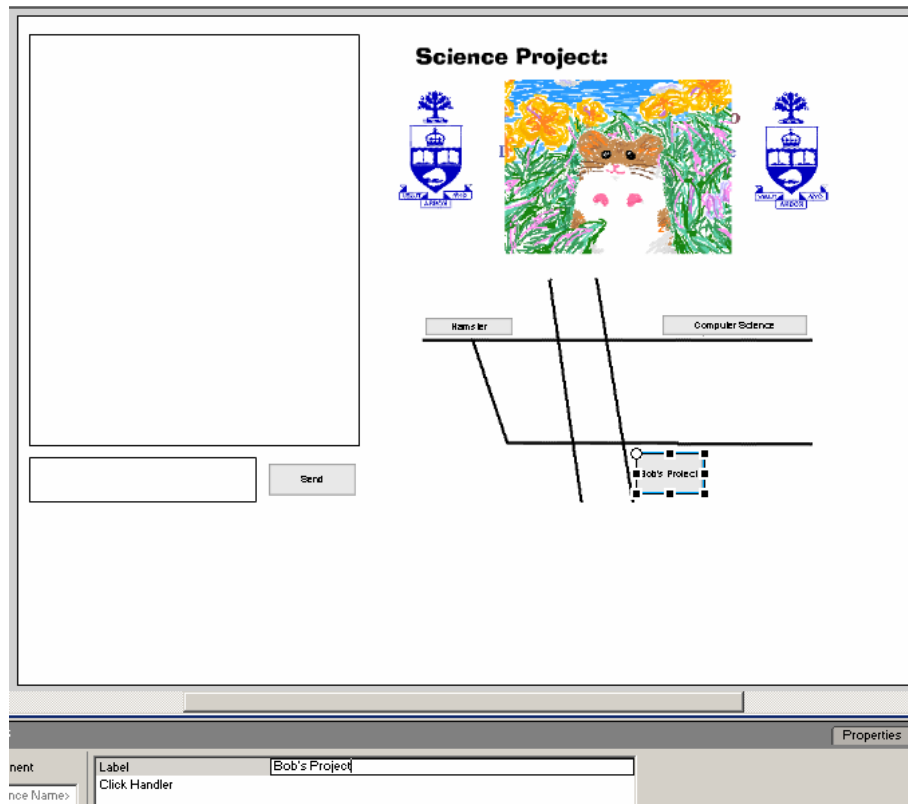


The Project Viewer is now ready – we need only setup the map to enable viewing of each of the projects.

### 7.2.4 Building the Map

The map will simply be a graphic, drawn in Windows Paint, with buttons atop it to allow the user to click on each of the projects. We'll copy & paste the map onto the project. Because the actual scripts will be on buttons above the map, the map itself is only for the use of the user – we won't be adding any functionality.

Paste the map, and drop 3 buttons on top of each of the project names on the map:

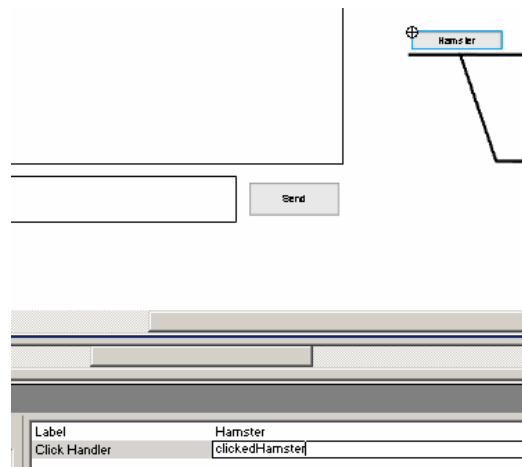


As before with the “Send” button, edit each of the buttons in turn. Add *Click* event handlers for each of them, and set their contents to make the appropriate image visible (and other invisible).

Edit the *Hamster* button, change the actions for the first frame to include functions “clickedHamster”, “clickedBobs”, and “clickedUTCS”, which makes the appropriate image visible, and the other invisible:

```
7 clickedHamster = function() {  
8     _root.Hamster._visible = true;  
9     _root.Bobs._visible = false;  
10    _root.UTCS._visible = false;  
11 }  
12 clickedBobs = function() {  
13     _root.Hamster._visible = false;  
14     _root.Bobs._visible = true;  
15     _root.UTCS._visible = false;  
16 }  
17 clickedUTCS = function() {  
18     _root.Hamster._visible = false;  
19     _root.Bobs._visible = false;  
20     _root.UTCS._visible = true;  
21 }  
22 }
```

Set the *Click Handler* of each button to the appropriate function defined above:



Repeat this for each of the other two buttons.

You now have a working prototype! Test it out by selecting “Test Movie” from the “Control” menu. (be sure to turn off “looping” in the Flash player, or the images will constantly be made invisible again).

## 7.2.5 Completed Prototype

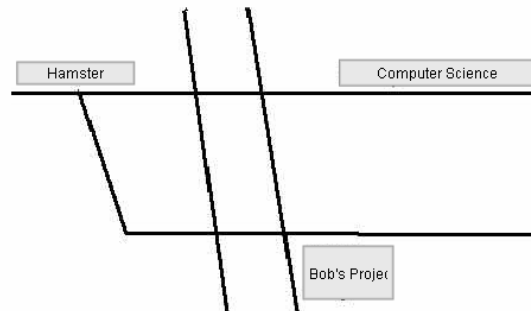
**You said: this is a great science fair!**

**I'm having fun!**

### Science Project:



*University Of Toronto*  
Department Of Computer Science  
World Wide Web Server



## **7.3 Suggested Readings for Dreamweaver:**

### **7.3.1 Online/Free Help:**

The tutorials in Macromedia Flash are time consuming, but well written. Before building your prototype, go through at least the first two tutorials.

### **7.3.2 Books:**

*Macromedia Flash MX: Training from the Source* by Chrissy Rey

*Macromedia Flash MX for Windows and Macintosh (Visual QuickStart Guide)*  
by Katherine Ulrich, Russell Chun

## 8. Author's Thoughts

As a computer scientist, you no doubt have a great deal of experience writing object oriented code in Java and C++. You also likely have little to no experience in designing web pages and doing animation. With this in mind, I recommend to 9 of every 10 students that they use Visual Basic to build their prototypes in 318. It allows very quick prototyping, and offer familiar (object oriented) programming style. To the other 1/10, those who want to make a web page for their 318 project, or those who plan to use a lot of animation, the tools included herein (Flash and Dreamweaver) are industry standard, and will allow you to very quickly build a prototype that you (and your TA) will be happy with.

Be forewarned – if you are working with your chosen tool for the first time, it will take you much more time than you'd expect to build your first prototype. I suggest you use the source code I have provided as a base upon which to build your prototypes. I also suggest that you work with your TA and instructor to determine exactly what is expected in terms of breadth, depth, and final “look and feel” of your system in each of your prototypes. Always keep these in mind as the primary requirements of your assignment, and choose the tool that best fits your desires (VB for deep but somewhat sterile prototypes, Dreamweaver for web site prototypes, and Flash for systems that require sound, video, and animation).