Contents

Chapter 1

Going Global

Engaging people and causing positive action is one of the key goals of systems thinking an modeling. The growth of the Internet has opened up amazing opportunities to reach out and connect with people in ways that have never been possible before. This book, for example, would never have been written if it was not for the groundswell of support and enthusiasm which could only be expressed using the Internet.

The Internet makes it incredibly easy for us to share our models with other. Not only can you email someone the results of a model, but you can also build web pages allowing anyone to see the results of the model. And these results do not have to be static, you include an interactive version of your model allowing others to experiment with it. This can be done on your personal web page, on your blog, or anywhere else on the Internet you wish.

Furthermore, the information flow doesn't have to be one-way. On next to your model on the webpage you can include feedback or comment forms that allow anyone to comment and share their thoughts on the model. These comments can be saved right on the page allowing other people to see them and enabling and discussion to start about the model. This creates many avenues for collaboration and learning that would simply be impossible without the Internet.

In this chapter, we'll show you how to develop web pages to showcase your insights and models to the world. We'll also show how to include tools to engage viewers and start a dialogue about your work. Before doing this, we'll have to lay the ground work by introducing the basic principles of web development. Once we have introduced these key principles, we'll walk through developing examples of interactive models.

The Web in a Nutshell

The world wide web is really a collection of many different technologies that work together. When developing a web page there are really three major technologies that you need to be familiar with: HTML, CSS and JavaScript. Each of these technologies plays a different role in developing a web page.

Technology	Acronym	Usage
Hypertext Markup Language	HTML	Page Structure
Cascading Style Sheets	CSS	Page Style
JavaScript	JS	Page Interactivity

When teaching web development many books and sources will just recommend you use some interactive web site builder. That is certainly a great way to get up and running, but ultimately you will find the approach very restricting. To truly harness the different tools available to you, you will need to have some understanding of the underlying technologies and be able to work directly in them. So, rather than using a website builder as a crutch, we recommend jumping right into these three technologies.

In the following sections we'll give you the briefest of introductions to each of these technologies. This introduction will be extremely brief so please do not worry if you do not fully understand everything. This introduction will provide you with everything you need to know in order to be able to get the maximum out of our later examples of interactive models on custom pages.

HTML Basics

HTML defines the structure of a web page or document. An HTML document is made up out of a set of *tags*. Each tag is enclosed in triangular brackets. For instance, there is a tag called "<hr>" that will create a horizontal division line in your document ("HR": "horizontal rule").

Many types of document structure will consist of an opening and closing tag paired together. An opening tag is written the same as a closing tag except the is a backslash immediately after the opening bracket. For instance, you could use the "<b>" tag to make a range of text bold:

```html
This is some text. <b>This text is bold.</b> This text is not bold.
```

Some tags may also have attributes. Attributes are included within the opening brackets of the tag after the tag name. For instance, the "<a>" tag is used to make links. The "<a>" tag has an attribute "href"[1] which is the url the link should connect to. The following HTML creates a link to Google.

[1]The tag name "a" comes from "anchor" and "href" is an abbreviation of hyperlink reference.

```
If you ever need to search something, just go
to <a href="http://Google.com">Google.</a>.
```

Every HTML page contains some general boilerplate that structures the document. This boilerplate contains several unique tags which splits the document into two sections: a "head" section to store the page title or page keywords for search engines, and a "body" section which is what is actually shown to the user. You will spend most of your time editing the body section. The standard template for a web page is as follows:

```
<html>
<head>
    <title>A Sample Web Page</title>
</head>
<body>
    Document contents goes here...
</body>
</html>
```

There are dozens of different tags you can use in your document to structure it. We can't comprehensively cover them here but the following table summarizes a few of the most useful ones:

Tag	Usage	Example
a	Creates a link	<a href="http://google.com">Google</a>.
b	Makes text bold	This text is <b>bold</b>.
i	Makes text italic	This text is <i>italic</i>.
u	Makes text underlined	This text is <u>underlined</u>.
center	Centers a paragraph	<center>In the middle.</center>
p	Creates a paragraph of text	<p>This is a paragraph.</p>
hr	Creates a dividing line	Something <hr> Something Else
h1	Creates a heading	<h1>This is a Heading</h1>
img	Embeds an image	<img src="http://example.com/image.png">

We can combine these tags together to form more complex documents. The following is an example of a full-featured webpage. Why don't open whatever word processor you use on your machine and save this to *MyPage.html* as a plain text file[2]. You can then open it in your web browser (Internet Explore, Firefox, Chrome, Safari, etc. . .) and you will have a fully functional webpage.

```
<html>
<head>
    <title>A Sample Web Page</title>
</head>
<body>
    <h1>Introduction</h1>
        <p>Here is some information about my page.</p>
    <h1>The Content</h1>
        <p>Here we have the meet of the page.</p>
    <hr>
    <h2>For Further Information</h2>
        <p>Here we have links to other sites about this content:<p>
        <p>We could check out <a href="http://BeyondConnectingTheDots.com">
            this book's site</a> for instance.</p>
</body>
</html>
```

For more information and tutorials on HTML, we recommend the Mozilla Developer's Network (https://developer.mozilla.org/en-US/docs/Web/HTML).

CSS Basics

Where HTML is used to define the structure of a document, CSS is responsible for styling this structure. A CSS document is a list of rules each rule consists of two parts: a selector which tells the browser what elements of the page the rule applies to, and a set of styles which tells browser how to style those elements. For example, take the following CSS code.

```
p {
    margin: 20px;
}

h1 h2 {
    font-size: x-large;
    font-color: red;
}
```

[2]Web pages are always stored as plain text. This differs from, for instance a Microsoft Word document (.doc or .docx) or a Rich Text Format document (.rtf). You need to ensure you save your document as a plain text document with the extension ".html" or ".htm". You can use any text editor you want, but if you get an editor designed for writing web pages it will have helpful features such as coloring your tags different from the standard text. We recommend Sublime Text (http://www.sublimetext.com/) as a high quality editor for serious work .

This code has two rules. In the first rule the selector is "p" meaning the rule will apply to all "p" tags in the document. The styling for this rule says to apply a 20 pixel margin around each of these paragraph tags. The second rule has the selector "h1 h2". This means apply the rule to both "h1" and "h2" tags (but not "h3", "h4", or "h5" tags) and to set the contents of those tags to have an extra large font and to be colored red.

There are numerous different aspects of an elements style you can set with CSS. It is impossible to go into them here, but for a full and detailed reference we recommend the Mozilla Developer's Network coverage of CSS (https://developer.mozilla.org/en-US/docs/Web/CSS/Reference).

CSS for a webpage can be placed in a standalone file which is referenced by the webpage or it can be included directly within the web page. Both these can be accomplished by placing a CSS rules within a special tag in the head of the document for example, taking the head section from our earlier document we could either embed the CSS directly:

```
<head>
    <title>A Sample Web Page</title>
    <style>
        p {
            margin: 20px;
        }

        h1 h2 {
            font-size: x-large;
            font-color: red;
        }
    </style>
</head>
```

Or we could save the CSS to an external text file (*MyStyles.css*) and link to it:

```
<head>
    <title>A Sample Web Page</title>
    <link rel="stylesheet" type="text/css" href="MyStyles.css">
</head>
```

JavaScript Basics

JavaScript[3] provides interactivity for web pages. JavaScript is a powerful programming language that you can use to respond to user actions, run calculations,

[3] JavaScript is officially known as ECMAScript. Due to trademark issues Microsoft refers to is as JScript if you are using Internet Explorer. It is important to note that *JavaScript* and *Java* are completely different technologies. They share part of a name due to historic branding purposes but they are completely different languages.

or modify the web page. An example of javascript code to calculate a Fibonacci number[4] is below:

```
function fib(n){
    if(n==1 || n==0){
        return 1;
    }
    return fib(n-1) + fib(n-2);
}

alert("The tenth Fibonacci number is: "+fib(10));
```

Like CSS, there are two ways to embed JavaScript into an HTML document. The first is to include the JavaScript directly in the document like we did for the CSS:

```
<head>
    <title>A Sample Web Page</title>
    <script>
        function fib(n){
            if(n==1 || n==0){
                return 1;
            }
            return fib(n-1) + fib(n-2);
        }

        alert("The tenth Fibonacci number is: "+fib(10));
    </script>
</head>
```

The second way to include the code is save the JavaScript into a text file (*MyScript.js*) and link to it into the document:

```
<head>
    <title>A Sample Web Page</title>
    <script src="MyScript.css"></script>
</head>
```

JavaScript is a very powerful tool but also a very complex one. This chapter will illustrate usages of JavaScript but we cannot hope to teach you how to write new JavaScript yourself in this single chapter. Again, we refer you to the Mozilla Developer Network to learn more about JavaScript (https://developer.mozilla.org/en-US/docs/Web/JavaScript).

[4] Where the first two Fibonacci numbers are 1 and the numbers thereafter are the sum of the two preceding numbers. The Fibonacci sequence starts: 1, 1, 2, 3, 5, 8, 13, 21, 44...

Creating a Live Insight

Now that we've made it through some of the technical details, let's jump into building a web page for an interactive model that users can comment on. Let's lay out the three basic things we want this webpage to have:

1. A description of challenge we are tackling, why we built the model and what the model contains.
2. An interactive version of the model that the user can explore and run simulations with.
3. A discussion forum that users can post comments on and see what others have posted.

This might seem ambitious, and it is! But using freely available technologies and services we'll be able to put this web page together very quickly. We're going to

Starting the Page and Description

Let's decide we want to create a page exploring population growth and whether the Earth can sustain us into the future. We start building our web page by creating an HTML file and putting the following text in it.

```
<html>
<head>
    <title>A Fragile Future</title>
</head>
<body>
    <h1>Introduction</h1>
        <p>This is a model simulation world population
            changes into the future.</p>

    <h1>The Model</h1>
        [Model goes here]

    <h1>Discussion</h1>
        [Discussion forum goes here]
</body>
</html>
```

This creates a page with three sections: Introduction, the Model, and Discussion. We can fill in the Introduction section with text describing the problem and our approach to understanding it in our model. In this example page, we have just put in a single sentence but you could extend it with more details on the model to really explain to the viewer why this is important and why we have modeled it.

The placeholders [Model goes here] and [Discussion forum goes here] are where we will fill in our model and discussion forum later on. For now though, we just want to specify the structure of the page.

Adding an Interactive Model

Now that we have created the structure for our webpage, we can add the interactive model. There are several ways to do this. One way would be to write the model in JavaScript and include it directly in the web page. JavaScript is a full-featured programming language and could be used to implement any of the models described in this book. Although implementing a model in JavaScript is definitely possible, it would require a lot of work. Writing a model in JavaScript would take extensive time and would not be possible without extensive programming experience.

Fortunately, using Insight Maker there is a much easier approach. Insight Maker models can be embedded very easily in a web page without any special effort on your part. So rather than writing, our world population model in JavaScript, we can simply build it Insight Maker. So build your model in Insight Maker just as you would build one normally. You can also take an existing model that you have already built. We'll use the World3 model (http://InsightMaker.com/insight/1954) which has a detailed worldwide model of population change.

Once you have finished constructing you model, click the *Embed* button in the *Tools* section of the toolbar. A window will open containing HTML code that you should paste into your webpage. What this code does is it embeds a version of the insight where it is placed in the webpage. For the World3 model this code is:

```
<IFRAME SRC="http://InsightMaker.com/insight/1954/embed?topBar=1&sideBar=1&zoom=1"
TITLE="Embedded Insight" width=600 height=420></IFRAME>
```

Take this code and use it to replace the [Model goes here] placeholder in your webpage. Open your webpage in a browser and you will now have a rich interactive version of your model directly in your webpage!

There are several features of the embedding that you can control by editing the "<IFRAME>" tag. For instance the "width" and "height" attributes control the size of the embedded model. They are specified in pixels and you can change them to make the embedded model smaller or larger. The "topBar" and "sideBar" parts of the URL control whether the toolbar and the sidebar will be shown in the embedded model. By default, they are set to 1 indicating these elements will be shown. Set them to 0 to hide the bars when the model is shown. The "zoom" part determines whether the model diagram is shown at its natural size or zoomed to fit the window (the default). Set this to 0 to prevent the model diagram from automatically fitting the window.

Adding a Discussion Section

Now we have one last piece to add before we have completed our page. We want people to be able to carry on a discussion about the model directly within the page.

Now, like the model we could program our own custom discussion system. But, like the model this is also a case where it is easier and better to leverage existing free software. A number of free commenting and discussion systems are available. One of these is called Disqus (http://disqus.com). If read a number of different news sites or blogs you have probably already used Disqus as many sites utilize their software.

You will need to sign up for a Disqus account to use them, but fortunately it should be completely free. Once you have completed signing up follow the site for directions on how to embed Disqus in your own webpage. You should be given code that looks similar to the following to place into your webpage:

```
<div id="disqus_thread">Discussion Here</div>
<script type="text/javascript">
    var disqus_shortname = 'SHORT-NAME-DEMO'; // required: replace example with your forum short
    (function() {
        var dsq = document.createElement('script'); dsq.type = 'text/javascript'; dsq.async = tr
        dsq.src = '//' + disqus_shortname + '.disqus.com/embed.js';
        (document.getElementsByTagName('head')[0] || document.getElementsByTagName('body')[0]).a
    })();
</script>
```

First edit this code as instructed (e.g. replace any usernames or ids with the ones you have been provided with) and then replace the [Discussion forum goes here] placeholder in the your page with this code. Load the page and test if it is working. One issue with Disqus is it might not work if the web page is just opened from your computer. You might have to upload it to the domain name you entered when you signed up for Disqus to make is work properly.

Completed Page

We have just put together a really neat site very quickly. Our site let's us share an interactive model with people and allows them to comment directly on it. All that it took was the following completed web page:

```
<html>
<head>
```

```
    <title>A Fragile Future</title>
</head>
<body>
    <h1>Introduction</h1>
        <p>This is a model simulation world population
            changes into the future.</p>

    <h1>The Model</h1>
        <IFRAME SRC="http://InsightMaker.com/insight/1954/embed?topBar=1&sideBar=1&zoo
        TITLE="Embedded Insight" width=600 height=420></IFRAME>

    <h1>Discussion</h1>
        <div id="disqus_thread">Discussion here</div>
        <script type="text/javascript">
            var disqus_shortname = ''; // required: replace example with your forum sh
            (function() {
                var dsq = document.createElement('script'); dsq.type = 'text/javascrip
                dsq.src = '//' + disqus_shortname + '.disqus.com/embed.js';
                (document.getElementsByTagName('head')[0] || document.getElementsByTag
            })();
        </script>
</body>
</html>
```

There is a lot more we could do with the site. Why don't you experiment with it, add some more descriptive text, maybe add some images, and experiment with CSS to adjust the styling.

Flight Simulators and Serious Games

In the preceding section we described how to very quickly develop a web site that contains an interactive model and provides users the ability to comment and discuss the model directly on the page. By leveraging Insight Maker, we were able to get an interactive version of our model embedded in our webpage by only copying a few lines of code.

In many cases this is exactly what you are looking for. However, in other cases its possible that you will wish to provide your users with a unique experience tailored to understanding a specific problem. For instance maybe you would like to develop what is known as a "flight simulator", a simulation tool that puts the user in the position of trying to manage a problem or achieve an outcome. For instance, if you had a model of a business going through a disruptive change, you could place the user in the position of the manager and ask them to adjust parameters in the model in order to safely shepherd the company through the change.

Similarly, “serious games” are tools designed to both engage and teach someone about a system. You can have a simulation model at the heart of a serious game or a flight simulator, but you will generally want to build a custom interface on top of that model that hides the stock and flow diagram and instead displays a control panel type interface to the user.

Fortunately, web technologies provide a rich environment for developing these flight simulators and serious games. Furthermore, using Insight Maker you can build your model and simulation engine using its model building tools and then build a custom interface on top of it to provide the exact experience you want to the user. In the following sections will develop a system custom interface to control our world population simulation.

Setting up the Page

We’ll start by stripping down our page from the previous example. Let’s remove the commenting system and the introduction so the page just contains the model (you can add these back on your own later on as an exercise). After we do this we will be left with a page just containing the embedded world simulation model.

In this case, however, we do not want the user to actually interact with or even see the embedded model. We will be adding our own custom interface and just using the embedded model to run the simulation and calculate results for us. To hide the embedded model we can add a CSS rule that makes the iframe tag invisible:

```
iframe {
    display: none;
}
```

This rule turns off the display of all iframe tags in the page. They are still there and in your page, they just are not shown to the user. The resulting completed template for our page is shown below. When you open this page in your browser you should see a completely blank page.

```
<html>
<head>
    <title>A Fragile Future</title>
    <style>
        iframe {
            display: none;
        }
    </style>
</head>
<body>
```

```
    <IFRAME SRC="http://InsightMaker.com/insight/1954/embed?topBar=1&sideBar=1&zoom=1"
    TITLE="Embedded Insight" width=600 height=420></IFRAME>
</body>
</html>
```

Adding the Control Panel

HTML has a tag called "<input>" that lets you create form elements for users to input data. The input tag has an attribute called "type" which determines what the type of the form element. There are a wide number of types including "number", "text", "color", "textarea", "date", and "button". For our control panel, we'll design it to have to properties of the model the user can configure and one button they can press to run the simulation. In addition to specifying the type of the inputs, we should also specify there initial values. We can do that using the "value" attribute of the input tag.

Lastly, we will need some way to reference the inputs and to load there values later on. Each tag in an HTML document has an option "id" attribute. This attribute can be used to obtain a reference to that element from JavaScript. We'll set the id attribute for our two input fields so we can obtain their values when we are ready to run the simulation.

The resulting control panel will look something like the following code. You should place this code after the iframe tag in your document.

```
<center>
    <p>This is a game to keep the world's population larger than 5 billion in the year
        We have can experiment with the amount of non-renewable resources and the star
        for a clean energy eco-friendly policy.</p>
    <p> Initial Non-Renewable Resources: <input type="number" value="100" id="resource
    <p> Start Policy Year: <input type="number" value="2013" id="year" /> </p>
    <p> <input type="button" value="Test Scenario" /> </p>
</center>
```

This will create two numbered input elements. The first, *Initial Non-Renewable Resources* will allow the user to increase or decrease the amount of non-renewable resources in the model at the start of the simulation. The second, *Start Policy Year* allows the user to specify the start date to implement a clean technology policy causing reduced pollutants being generated in the simulation. A button is also created that lets the user test the scenario in the simulation.

Making it Interactive

We use JavaScript to add interactivity to the web page. Let's define a JavaScript function "testScenario" that we will use to read in the user specified options from the control panel, run the simulation with these parameter values, and then report to the user whether they were successful or no.

We'll fill out this function with these action later; but for now, just add the following to the head section of your web page.

```
<script>
    function testScenario(){
        alert("Scenario tested!");
    }
</script>
```

This creates the function, but we also need a way for the function to be called when the "Test Scenario" button is pressed. There are several ways to do this. The easiest is to set the "onClick" attribute of the button to call the function. The "onClick" attribute of an input may contain JavaScript code that is executed when the button is click. To link up our button with the function, we can change our input button in the HTML to:

```
<p> <input type="button" value="Test Scenario" onclick="testScenario()" /> </p>
```

Implement the web page up to this point and check to make sure that the you see a message pop up saying "Scenario tested!" when you press the "Test Scenario" button.

Now that we have implemented basic interactivity, let's flesh out the *testScenario* function.

Load Parameter Values from the Control Panel

```
var resources = document.getElementById("resources").value;
var year = document.getElementById("year").value;
```

Inject the Parameter Values into the Model

Insight Maker an extensive JavaScript API that can be used to modify and script models. This is the same API that may be used with Button primitives. Refer to the API reference at http://insightmaker.com/sites/default/files/API/files/API-js.html for full details about the API.

The API instructions provide examples about how to integrate with an embedded model. We will adapt those instructions to our own case. First, as the instructions indicate we need to update our iframe tag to add an id attribute. We adjust our iframe tag like so:

```
<IFRAME id="model" SRC="http://InsightMaker.com/insight/1954/embed?topBar=1&sideBar=1&zoom=1"
TITLE="Embedded Insight" width=600 height=420></IFRAME>
```

Now we can get a reference to the model and send API command to it using the *postMessage* function. We use the *findName* API command to get a reference to a specific primitive and then use the *setValue* API command to set it to the value of the parameter in the control panel. Add the following code to the *testScenario* function.

```
var model = document.getElementById("model").contentWindow;

model.postMessage("setValue(findName('Initial Nonrenewable Resources'), '"+(resources/
model.postMessage("setValue(findName('Progressive Policy Adoption'), '"+year+"')", "*"
```

In case your wondering, this whole convoluted *postMessage* mechanism to pass JavaScript commands to the embedded model is a constraint necessitated by your browsers security mechanisms.

Run Simulation and Access Results

To run the model, we use the *runModel* API command. We indicate that the simulation should be run in silent mode so the results are returned. We then use the *lastValue* function to return the final population size for the simulation at the year 2100. We copy this into our web page at the end of the *testScenario* function.

```
model.postMessage("runModel({silent: true}).lastValue(findName('Population'))", "*");
```

So var we have just demonstrated one way communication between the control panel and the embedded model. This is the first example where we need to be able to communicate the other way, to receive things back from the embedded model.

Unfortunately, due to the security constraints imposed by your browser, this is much more complex than it needs to be. In order to receive a message back from the embedded model, we need to register an event handler with our window. Don't worry if you don't fully understand this. Just copy the code below into the script tag of your window.

```
function scenarioComplete(event)
{
    if(event.data){
        var pop = Math.round(event.data);
        if(pop > 5000000000){
            alert("Success! The population size of "+pop+" is larger than 5 Billion!")
        }else{
            alert("Failure! The population size of "+pop+" is smaller than 5 Billion!"
            alert("Please try again.")
        }
    }
}

window.addEventListener("message", scenarioComplete, false);
```

Final Result

The resulting completed webpage is as follows:

```
<html>
<head>
    <title>A Fragile Future</title>
    <style>
        iframe {
            display: none;
        }
    </style>
    <script>
    function testScenario(){
        var resources = document.getElementById("resources").value;
        var year = document.getElementById("year").value;

        var model = document.getElementById("model").contentWindow;

        model.postMessage("setValue(findName('Initial Nonrenewable Resources'), '"+(resources/10
        model.postMessage("setValue(findName('Progressive Policy Adoption'), '"+year+"')", "*");

        model.postMessage("runModel({silent: true}).lastValue(findName('Population'))", "*");
    }

    function scenarioComplete(event)
    {
        if(event.data){
            var pop = Math.round(event.data);
            if(pop > 5000000000){
                alert("Success! The population size of "+pop+" is larger than 5 Billion!")
            }else{
                alert("Failure! The population size of "+pop+" is smaller than 5 Billion!")
                alert("Please try again.")
            }
        }
    }

    window.addEventListener("message", scenarioComplete, false);
    </script>
```

```
</head>
<body>
    <IFRAME id="model" SRC="http://InsightMaker.dev/insight/1954?embed=1&topBar=1&side
    TITLE="Embedded Insight" width=600 height=420></IFRAME>

    <center>
        <p>This is a game to keep the world's population larger than 5 billion in the
            We have can experiment with the amount of non-renewable resources and the
            for a clean energy eco-friendly policy.</p>
        <p> Initial Non-Renewable Resources: <input type="number" value="100" id="reso
        <p> Start Policy Year: <input type="number" value="2013" id="year" /> </p>
        <p> <input type="button" value="Test Scenario" onclick="testScenario()" /> </p
    </center>

</body>
</html>
```

The key goal of this chapter is not to be able to completely understand this, but to be able to adapt it to your own needs. There is a lot of additional changes that could be made to this demo. You could clean up the control panel and make it look more attractive by adding some CSS, you could add inputs to control additional parts of the model, you could show the user the trajectory of the population instead of just the final value. Go ahead and experiment with this example to see what you can make it do.

Additional Tips

Web development is a very complex topic with a lot of nuances. The preceding sections should have given you a brief introduction in how to create interactive models for engaging an audience and encouraging discussion and learning. Although we cannot give you comprehensive introduction in web development, there a few additional tips that will be very useful when you go to develop your own applications.

Debugging Webpages

As you develop your webpage it is almost certain that you will make many mistake and typos as you go along. If you make a mistake within the HTML or CSS of the page you will have an immediate visual indication that something is wrong and you can experiment with you code until it is fixed.

JavaScript errors, on the other hand, generally won't provide any visual feedback that an error has occurred. The most likely indication that a JavaScript error has occurred is that nothing happens when you click a button or expect an action to occur. Debugging issues like these can be quite difficult. Fortunately, with just a little bit of additional work you can get access to very rich and informative JavaScript error messages letting you know exactly what went wrong and when.

There are two approaches to obtain access to the JavaScript error messages. The first is to actually edit your webpage and add code to show an error message when an error occurs. Adding the following to a script tag in the head section of your document will do that:

```
window.onerror = function(message, url, line) {
  alert("JavaScript Error: \n\n" + message + " (line: " + line + ")";
}
```

The second approach you can take is to use the developer tools that are built into your browser to study the webpage and observe errors as they occur. Excellent developer tools are built into all current browsers. These tools let you study the structure of the webpage, profile the performance of your code, and examine how the webpage behaves.

One particular tool is very useful when developing webpages: the JavaScript console. Once you have opened the JavaScript console (search on-line for the exact directions for your specific web browser) errors and messages from the webpage will appear in the console as they occur. What's more that console allows you to evaluate JavaScript commands in the webpage simply by typing the commands into the console.

One approach to debugging code is to put *alert* functions into the code checking on the progression of the code or displaying values of the variables. This works, but can be very clumsy and disruptive. When you have the console open, you can send messages directly to the console providing information on the status of the program. For example:

```
console.log("The value of the variable is: " + myVariable);
console.error("An error has occurred!");
```

Send Complex Data Back and Forth

The message communication technique to send data back and forth to the embedded can only be relied upon across browsers to send strings and objects that can easily be converted to strings (like numbers)[5]. Oftentimes you will want to pass more complex data from the simulation to the containing window. For instance you might want to pass the whole time series of values one or more primitives took on over the course of the simulation.

To handle these more complex objects you must convert them to strings. JavaScript provides a number of techniques to do so. For instance, if you

[5]The specification for this feature provides that any type of JavaScript object should be supported, however a number of recent browsers only support strings.

have an array, you can convert it back and forth from a string using the *join* and *split* functions:

```
var data = [1, 4, 9, 16, 25];
var str = data.join("; "); // "1; 4; 9; 16; 25"
str.split(", "); // [1, 4, 9, 16, 25]
```

By far the most useful and flexible method of converting JavaScript objects to strings are the JSON command. JSON, JavaScript Object Notation, is a general file format for storing data. It is based on the the standard method for declaring JavaScript objects ({key: value}) but has some differences. What is great about it is that your browser has built in commands for converting any JavaScript object (a number, array, or other object but not functions) into a string and then later back into an object.

You can use this technique to send arbitrarily complex objects back and forth from your simulation to your webpage. Let's see how this works:

```
var obj = {title: "I'm a complex object", data: [1, 4, 9]};
var str = JSON.stringify(obj); // '{"title":"I'm a complex object","data":[1,4,9]}'
JSON.parse(str); // {title: "I'm a complex object", data: [1, 4, 9]};
```

Hosting a Web Page

In this chapter we have saved the webpages to your personal computer's hard drive and opened them from there. Once you are already to share your creations with others, you are going to have to move the files to a web-server or web-host so that others can access them over the Internet. There are a number of different options to doing this that range from the simple to the complex and from the free to the expense.

On the simple and free end there are free blogging sites like Blogger (http://www.blogger.com) or WordPress (http://wordpress.com/). These sites allow you to create free blogs but they also allow you to do much more than that. They will generally let you edit the source code of your pages allowing you to implement the demos in this chapter directly within a blog post.

A step up from simple sites like these are shared hosting providers. Shared hosting providers such as DreamHost (http://dreamhost.com) take a server and allow many people to purchase space on the server to run their webpages. There are numerous shared hosting providers available. A more advanced version of shared hosting is virtual private server (VPS) hosting. VPS providers such as RimuHosting (http://rimuhosting.com/) are similar to shared hosting providers in that they fit many customers on a single server. Where they differ is that a VPS host will give each customer a virtualized computer. Each costumer will feel like they have complete control over their own computer and operating system even though they are sharing the actual hardware with others

At the high end of complexity, cost and power are dedicated servers. In this case you purchase or rent a machine solely for the purpose of hosting you projects. This gives you complete control of your hosting situation but is expensive and may take a lot of effort to set up and maintain.

In general, we recommend starting small. Sign up for a Blogger account and experiment with these techniques there. If you keep at it and you site grows, at some point you will have outgrown blogger and at that point you can upgrade to a more advanced solution.