

Using JavaScript to Enhance Web Pages

Dr. William A. Hansen
Manta Technologies Inc.
whansen@mantatech.com



Copyright 2010. Manta Technologies Inc.

1

- You have now seen the rules, structure, and key statements of the JavaScript programming language.
- What you have not yet seen is what you can do, now that you have learned basic JavaScript coding.



Copyright 2010. Manta Technologies Inc.

2

Handling Events

- One way to invoke a function is by coding its name (and any required parameters) in a JavaScript statement.
- A function can also be invoked as the result of an *event* that is associated with an HTML element.



Copyright 2010. Manta Technologies Inc.

3

Associating a Function with an Event

- Consider the following HTML example:

```
<input type="button" onClick="terminate()" value="Exit" />
```

- This creates a button labeled "Exit".
- The **onClick** attribute associates a click *event handler* with this button.
- When the user clicks the **Exit** button, the browser calls the function named **terminate()**.
- This function may perform any tasks available to a JavaScript program.



Copyright 2010. Manta Technologies Inc.

4

Other Events

- Following are some of the other events that can be associated with HTML elements and, therefore, cause a function to be invoked:
 - `onKeyPress`
 - `onChange`
 - `onKeyDown`
 - `onKeyUp`
 - `onMouseDown`
 - `onMouseOver`
 - `onMouseUp`
 - `onMouseOut`
 - `onFocus`
 - `onSelect`



Copyright 2010. Manta Technologies Inc.

5

- For example, the **onMouseDown** event is triggered when the user presses a mouse button.
- The event is triggered on the downstroke, while the **onMouseUp** event is triggered on the upstroke.
- In contrast, a *click* (which triggers an **onClick** event) is a downstroke followed by an upstroke while the pointer is positioned over the same object.
- Following is an example:

```
<p onMouseDown="vote('Bill')">Click here to vote for Bill.</p>
```

- If the user presses a mouse button with the mouse pointer positioned over this paragraph, the function **vote** is called.
- Notice that the function call includes a parameter that identifies which paragraph was clicked.



Copyright 2010. Manta Technologies Inc.

6

- Following is the vote function:

```
function vote(candidate) {  
    switch (candidate) {  
        case "Bill":  
            Billcount ++;  
            break;  
        case "John":  
            Johncount ++;  
            break;  
        case "Kate":  
            Katecount ++;  
            break;  
    }  
}
```

- This function updates one of several counters, depending on the parameter that was passed.
- The counters were defined and initialized as global variables.



Copyright 2010. Manta Technologies Inc.

7

- The definition of each HTML element includes a list of the events that may be associated with it.
- This list varies from one browser to another.



Copyright 2010. Manta Technologies Inc.

8

The onLoad Event

- The JavaScript statements within an HTML document get executed as they are read by the browser.
- This does not include functions, which are not executed until they are called.
- But it does include all statements in the main JavaScript program.
- This creates a potential problem when a JavaScript statement references an HTML element.
- If the line defining the element has not been read when the JavaScript statement is executed, an error will result.



Copyright 2010. Manta Technologies Inc.

9

- The solution is to define an **onLoad** event handler, as follows:

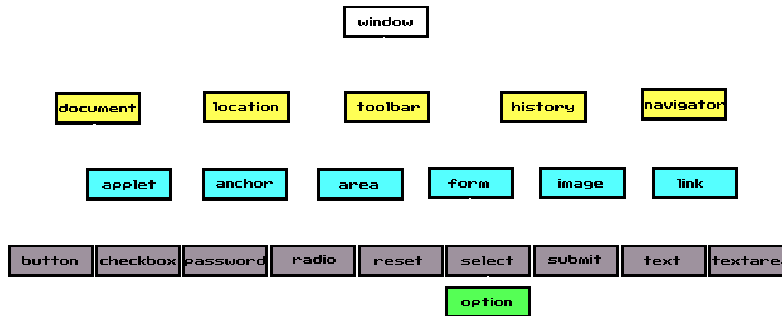
```
<html>
<head>
:
<script type="text/javascript">
function init() {
:
}
</script>
</head>
<body onLoad="init()">
```
- The **onLoad** event can be associated with the **<body>** element.
- It is triggered when the entire document has been loaded by the browser.
- In the example above, the loading of the HTML document causes the function **init()** to be invoked.
- The trick is to leave the main JavaScript program empty, with the possible exception of **var** statements defining global variables.
- Put all other statements that are to be executed immediately into the **init()** function.
- If these statements reference elements within the HTML page, the elements will be defined and available.



Copyright 2010. Manta Technologies Inc.

10

The Document Object Model



Copyright 2010. Manta Technologies Inc.

11

Manipulating Window Objects

- There is one **window** object for each window that is currently open under the browser.
- Each tab is a separate window.
- All other objects available to you are subordinate to this object.



Copyright 2010. Manta Technologies Inc.

12

Window References

- Normally, you will deal with a single window, the one used to display the document containing your HTML statements and JavaScript program.
- You can refer to this window using either the name **window** or the synonym **self**.



Copyright 2010. Manta Technologies Inc.

13

- Either of the following statements displays the URL of the page containing this JavaScript statement:

```
alert(window.location);
```

```
alert(self.location);
```

- From the syntax, you should recognize **location** as a property of the window object.



Copyright 2010. Manta Technologies Inc.

14

alert

- **alert** is a method of the window object.
- It uses one parameter, a message to be displayed in a separate window over the base window.
- In the previous example, neither **window** nor **self** was specified before **alert**.
- Because the base window is always "there," the browser will assume this is the window you are referring to if you leave off the window object reference.
- As a result, the following are all equivalent:

```
window.alert(window.location);  
self.alert(window.location);  
alert(window.location);  
window.alert(self.location);  
self.alert(self.location);  
alert(self.location);  
window.alert(location);  
self.alert(location);  
alert(location);
```



Copyright 2010. Manta Technologies Inc.

15

Creating a Subwindow

- The name of the window becomes important only when the browser is displaying multiple windows.
- You can use the following JavaScript statement to create a subwindow, also called a **pop-up** window:

```
var glsWindow = window.open("glossary.htm","", "height=350,width=600");
```

- The first parameter of the **open** method identifies the HTML document to be displayed in the new window.
- The second parameter specifies an optional window target name, telling the browser where to put the new window.
- The third parameter supplies window attributes, such as the height and width of the window to be created.
- Like other methods, the **open** method returns a result.
- In this case, it is an object reference to the new window.
- Once the above statement has executed, the variable name **glsWindow** can be used to reference the new window.
- Keep in mind that with newer browsers, the user can force pop-up windows to appear in a new tab rather than as a separate window.



Copyright 2010. Manta Technologies Inc.

16

Closing a Window

- The following JavaScript statement will close the window:

```
glsWindow.close();
```

- This statement has meaning only within the first window, which contains the JavaScript statement where **glsWindow** was defined.
- If you wish to close the new window from a JavaScript program within that window, you would need to use one of the following references:

```
window.close();  
self.close();  
close();
```



Copyright 2010. Manta Technologies Inc.

17

- The second window can refer back to the window which created it by using the window name **opener**.
- Following is an example:

```
opener.close();
```

- The user is notified if a JavaScript program attempts to close any window other than a subwindow that it created.
- The window will not be closed unless the user agrees.



Copyright 2010. Manta Technologies Inc.

18

Checking a Window's Existence

- Depending on the logic of your program, you might attempt to close a window that has never been created or has already been closed by the user.
- The latter is not a problem, since the browser will ignore a close request for a window that has already been closed.
- An error will result, however, if you attempt to close a window that was never defined.
- To avoid this problem, you can test for the existence of the window before closing it:

```
if (glsWindow) {  
    glsWindow.close();  
}
```
- Recall that the expression tested by an **if** statement is treated as a boolean value.
 - If the statement creating **glsWindow** was never executed, **glsWindow** is undefined.
 - The value **undefined** is treated like **false** in a boolean expression.
- As a result, the **close** method in the example above will be executed only if **glsWindow** exists.



Copyright 2010. Manta Technologies Inc.

19

- You can generalize this technique to test for the existence of any object.
- In particular, **objectName** will evaluate to **true** if the object has been defined, or to **false** if it has not.
- This assumes, of course, that **objectName** is not a numeric or boolean variable.
- Recall that a numeric variable is also considered false if it contains zero.



Copyright 2010. Manta Technologies Inc.

20

Changing Properties

- Many properties can be changed dynamically by your JavaScript program.



Copyright 2010. Manta Technologies Inc.

21

Changing the Window Status

- The *status bar* is the area at the bottom of the browser window.
- This area displays the URL of the page that is being loaded in the window.
- It is also used to display messages from the browser.
- By setting the **status** property of the window, you can cause your own message to be displayed in this area.
- Following is an example:

```
window.status = "Please fill in the form and click Submit.";
```



Copyright 2010. Manta Technologies Inc.

22

Changing the Window Location

- The **location** property specifies the URL of the page that is currently loaded in the browser window.
- By changing the location property, you also change the loaded page.
- In effect, this executes an automatic link to another web page.
- Following is an example:

```
window.location = "http://www.mantatech.com";
```

- Keep in mind that linking to another page replaces the page containing your JavaScript code.
- Any functions to be performed by the new page must be contained there.



Copyright 2010. Manta Technologies Inc.

23

Prompting the User

- The **alert** method of the window object can be used to display a message to a user.
- **alert** displays the indicated message in a new window, as shown below.



Copyright 2010. Manta Technologies Inc.

24

- Sometimes a message is not sufficient.
- You want the user to respond, either by entering a value or by giving consent to an action.
- The **prompt** and **confirm** window methods can be used for these purposes.



Copyright 2010. Manta Technologies Inc.

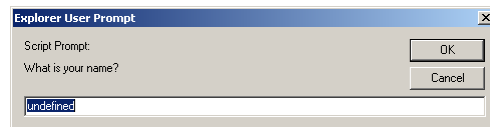
25

The Prompt Method

- Following is an example of the **prompt** method:

```
var userName = self.prompt("What is your name?");
```

- This causes the following window to be displayed:



- When the user presses the **Enter** key or clicks the **OK** button, the string that was entered in the input field becomes available to the program in the **userName** variable.
- If the user clicks the **Cancel** button, the variable is set to **null**.



Copyright 2010. Manta Technologies Inc.

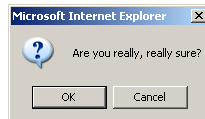
26

The Confirm Method

- If all you need is a "yes" or "no" type of response, use the **confirm** method instead:

```
var answer = window.confirm("Are you really, really sure?");
```

- This causes the following window to be displayed:



- The **confirm** method returns a boolean value, so **answer** will contain **true** if the user selects **OK** or **false** if the user clicks **Cancel**.



Copyright 2010. Manta Technologies Inc.

27

Manipulating Document Objects

- After the **window** object, the next most important object is the **document** object.
- The document object corresponds to the actual HTML document that is displayed in the browser window.



Copyright 2010. Manta Technologies Inc.

28

Changing the Title

- Among the properties of a document object is its **title**.
- Normally, the title of a document is defined by the **<title>** HTML element.
- However, you can display or change the title of a document by using the document **title** property.
- For example, the following statement changes the title displayed at the top of the browser window:

```
document.title = "Mailing List Form";
```



Copyright 2010. Manta Technologies Inc.

29

Writing to the Document

- Your JavaScript program can dynamically change the contents of the browser window by writing HTML statements to the document object.
- The browser displays a document file using a two-step process.
 - First, the browser reads the file, organizing its elements in memory using the browser's document object model.
 - Second, other browser routines (called the *display engine*) display the document by mapping the objects comprising the document into pixels for presentation on the screen.
- You can change the window contents by changing the document object in memory.
- To do so, you can use JavaScript statements to perform the first step of the display process.
- Alternatively, you can use JavaScript methods to modify the HTML objects in memory.



Copyright 2010. Manta Technologies Inc.

30

- Changing the document object is a three-step process:
 1. Before you can write data to the document object, it must be opened. You can open a document object explicitly with the **open()** method. However, the browser will also open the document for you as required.
 2. You use the **write(string)** method to write HTML statements to the document object. In general, you will write an entire HTML document to replace the entire window contents.
 3. Finally, you should use the **close()** method to close the document object. Once a document has been closed, new lines cannot be appended to it. Instead, any new lines replace the contents of the browser window.



Copyright 2010. Manta Technologies Inc.

31

- Timing plays an important role in determining the result of your actions.
- For example, suppose your JavaScript program includes the following statement:


```
document.write("Hi");
```
- This writes the character string "Hi" to the browser window.
- If the JavaScript statement is positioned in the main program before the body elements in the document, the line will be written before the remainder of the document.
- If you use an **onLoad** event to postpone the execution of the statement, the browser will be able to write the entire document and close it before the **onLoad** handler is invoked.
 - In that case, the **write** method will replace the entire contents of the browser window with the new line.
- In general, the reason for using the **write** method is to replace the current document with a new document that is created dynamically by the JavaScript program in response to user input.
- Because of this, the following examples assume that the **write** method is within a function that will not be invoked until the original HTML page has been displayed.



Copyright 2010. Manta Technologies Inc.

32

- A common technique for creating output for a **write** operation is to concatenate string literals.
- Be sure to include everything required by a complete web page, as in the following example:

```
var outputHTML = "<html><head><title>";
outputHTML += "Sample Web Page";
outputHTML += "</title></head><body>";
outputHTML += "This is the document body.";
outputHTML += "</body></html>";
```

- Assume that an HTML document has been displayed, and some event has caused a JavaScript function to be invoked.
- To replace the existing document, write the new content as follows:

```
document.open();
document.write(outputHTML);
document.close();
```

- The **open()** method is optional in this case, because writing to a closed document will cause the browser to replace the existing document.



Copyright 2010. Manta Technologies Inc.

33

Writing to a Subwindow

- Rather than writing to the existing window (which contains your JavaScript program), it is also possible to write to a subwindow that has been created for this purpose.
- Suppose we have created a new window with this statement:

```
var msgWindow = window.open("blank.htm", "", "height=350,width=600");
```

- Also suppose we have defined a string named **outputHTML** containing the HTML statements to be written to the new window.



Copyright 2010. Manta Technologies Inc.

34

- If it is possible that the user may close the new window before you write to it, you can check for that event and re-create the window if necessary:

```
if (msgWindow.closed) {  
    msgWindow = window.open("blank.htm", "", "height=350,width=600");  
}
```

- As this example suggests, **closed** is a property of a window object that is either *true* or *false*.



Copyright 2010. Manta Technologies Inc.

35

- Then write to the new window using statements like the following:

```
msgWindow.document.open();  
msgWindow.document.write(outputHTML);  
msgWindow.document.close();
```

- The only difference from the previous example is that each document object name is preceded by the name of the target window, **msgWindow**.
- Similar techniques can be used to replace a document in an **iframe**.



Copyright 2010. Manta Technologies Inc.

36

Referencing Form Data

- A JavaScript program can reference windows or frames and the documents they contain.
- We will now go deeper into the browser's document object model to reference forms and input elements.
- My purpose is to illustrate the most common application for a JavaScript program, to edit input data before sending it to a CGI program or servlet for processing.



Copyright 2010. Manta Technologies Inc.

37

- Editing input data is a task that can be performed either on the client side (using JavaScript) or on the server side (by the CGI program or servlet).
- Sending form data to the server and having it check the data for errors can take a long time, particularly over a slow communication link or using an overworked server.
- This can be frustrating to a user who makes a series of simple mistakes, such as not filling in all required fields.
- If a user has to wait excessively for each error message, the probability increases that he or she will simply terminate the dialog rather than continue.
- This can mean angry customers and lost sales.
- In contrast, the fastest response is achieved by performing as much checking as possible on the client system, using a JavaScript program within the original web document sent to the user.
- Only when all required fields are filled in and other data is checked for validity is the form data sent to the server for processing.



Copyright 2010. Manta Technologies Inc.

38

- Our ultimate goal is to enhance the web page on the right.
- Currently, the form data is sent to a CGI program without checking.
- Our job will be to add tests to find as many errors as possible on the client, before sending the form data to the server.
- Before we write the code to edit the input data, however, you need to see how to reference the text fields and other elements within a form.

Please supply the following information.

Dr. ☐ Mr. ☐ Mrs. ☐ Ms. ☐

First Name:

Last Name:

Phone:

Address:

City: State: ZIP:

E-Mail address:

Company:

How did you find out about our services?

Select:



Copyright 2010. Manta Technologies Inc.

39

Referencing the Form

- Because a document can, in theory, contain multiple forms, the forms within a document are stored in an array called **forms**.
- The browser constructs this array as a child of the document object.
- As a result, you can use the following name to reference the first form within the document:

```
document.forms[0]
```
- Other document objects can be referenced similarly.
- For example, all images within a document (as specified with the **** element) are stored in the array named **images**.



Copyright 2010. Manta Technologies Inc.

40

Naming the Form

- An easier way to reference document elements is by name.
- Suppose we add the **name** attribute to the **<form>** element, as follows:

```
<form method="post" action="cgi-bin/mycgipgm.pgm" name="userData">
```

- The form can then be referenced by the name
document.userData
or simply as
userData



Copyright 2010. Manta Technologies Inc.

41

Identifying Input Fields

- To your JavaScript program, each **<input>** element can be referenced as a child of the form object.
- For example, the first text field in the sample form was defined as follows:

```
<input name="firstname" size="40" maxlength="40" />
```

- The JavaScript program can identify this field as
userData.firstname
or
forms[0].firstname
- Unlike the **window** and **document** names, you cannot leave off the form identifier.



Copyright 2010. Manta Technologies Inc.

42

- Form fields will always be named, because the name is used to pass the value to a CGI program or servlet.
- However, form fields are actually stored within the **elements** array, which is a child of the form object.
- As a result, the first text field in the example can also be referenced as follows:

```
forms[0].elements[4]
```

- This field is element 4, because it follows four radio buttons that are elements 0 through 3.



Copyright 2010. Manta Technologies Inc.

43

Retrieving Text Values

- The **value** property can be used to retrieve the contents of a form element.
- The following example copies the contents of the first text field to a variable named **firstName**:

```
var firstName = userData.firstname.value;
```



Copyright 2010. Manta Technologies Inc.

44

- You can also retrieve a value using the array notation, as follows:

```
var firstName = forms[0].elements[4].value;
```

- A handy use for this notation is to loop through every element in a form containing only text fields, testing whether the user typed anything in each field.

```
var numberOfElements = document.forms[0].length;  
for (var i = 0; i < numberOfElements; i++) {  
    if (forms[0].elements[i].value == "") {  
        action when element is empty  
    }  
}
```



Copyright 2010. Manta Technologies Inc.

45

- The previous examples reference a field defined as an **<input>** element with **type="text."**
- The same technique can be used to retrieve the contents of **password**, **hidden**, and **textarea** elements.
- In addition to retrieving the values of text fields, you can determine the settings of other input element types, including checkboxes, radio buttons, and pull-down lists.



Copyright 2010. Manta Technologies Inc.

46

Checkboxes

- A checkbox is defined with an HTML statement like the following:

```
<input type="checkbox" name="mailFlag"/>Please send me spam daily.
```

- You can assign a value to a checkbox with the **value** attribute.
- For example, had we included **value="mail"** in the example above, **forms[0].mailFlag.value** would contain "mail" if the box were checked.
- For JavaScript purposes, it is more common to leave off the **value** attribute and use the **checked** property.
- This property returns **true** or **false** depending whether a checkbox is checked or not.

```
if (forms[0].mailFlag.checked == true) {  
    action when the box is checked  
} else {  
    action when the box is not checked  
}
```



Copyright 2010. Manta Technologies Inc.

47

Radio Buttons

- Radio buttons are a little more complicated than checkboxes, because multiple boxes are treated as a group.
- In the example, the user is to select a title from the following:

```
Dr.<input type="radio" name="title" value="DR" />  
Mr.<input type="radio" name="title" value="MR" />  
Mrs.<input type="radio" name="title" value="MRS" />  
Ms.<input type="radio" name="title" value="MS" />
```

- Because the radio elements all use the same name (**title**), the browser will let the user select no more than one of the buttons in the set.
- From the perspective of the JavaScript program, the buttons are represented as an array under the form object.
- The name of the array is the name given to the group.
- In our example, because each of the elements contains **name="title"**, the group can be identified as follows:

```
userData.title
```

- Similarly, you can test whether the first button is checked using the following reference:

```
userData.title[0].checked
```



Copyright 2010. Manta Technologies Inc.

48

- The following code loops through all the buttons in a set, checking whether any has been checked:

```
var foundChecked = false;
for (var i = 0; i < userData.title.length; i++) {
    if (userData.title[i].checked == true) foundChecked = true;
}
```

- The boolean variable named **foundChecked** holds the result of the test.



Copyright 2010. Manta Technologies Inc.

49

Pull-Down Lists

- Following is the definition of a sample pull-down list:

```
<select name="color" size="1">
    <option value="c1">blue</option>
    <option value="c2">green</option>
    <option value="c3">red</option>
    <option value="c4">white</option>
    <option value="c5">black</option>
    <option value="c6">cyan</option>
    <option value="c7">magenta</option>
    <option value="c8">gray</option>
    <option value="c9">yellow</option>
    <option value="c10">orange</option>
    <option value="c11">brown</option>
    <option value="c12">pink</option>
    <option value="c13">brown</option>
    <option value="c14">lime</option>
</select>
```

- The **<select>** element identifies the list as a whole.
- Each item in the list is identified with an **<option>** element.



Copyright 2010. Manta Technologies Inc.

50

- This structure is reflected in the document object model.
- The name specified on the **<select>** element identifies the group as a whole.
- Under the group is an **options** array, containing one entry for each **<option>** element.
- Thus, you can refer to an individual option of the previous example with a reference like the following:

```
forms[0].color.options[5]
```

- Here, **color** is the name specified on the **<select>** element.



Copyright 2010. Manta Technologies Inc.

51

- The **selectedIndex** property of the select object tells you which option has been selected.

```
var i = forms[0].color.selectedIndex;
```

- If this number is 2, for example, the user must have selected **option[2]** (the third option).



Copyright 2010. Manta Technologies Inc.

52

- Each option has two properties of interest:

```
forms[0].color.options[i].text  
forms[0].color.options[i].value
```

- In our example, with **i** set to 2, the **text** value is "red," the contents of the **<option>** element.
- In contrast, the **value** of this element is **c3**, which was specified by the **value** attribute.



Copyright 2010. Manta Technologies Inc.

53

Passing a Form to a Function

- In general, the least confusing way to refer to form elements is to include a **name** attribute on every element and to use this name in your JavaScript statements.
- The disadvantage of this technique is that it forces the JavaScript statements to depend on the HTML names.
- This conflicts with the recommended programming practice of writing generic functions that can be used in a variety of situations.
- The solution is to pass objects to functions as parameters, whose names can be defined within each function.



Copyright 2010. Manta Technologies Inc.

54

- Suppose you define a text field as follows:

```
<input name="referredfrom" size="50" maxlength="50"
      value="web site" onChange="check(this)" />
```

- The **value** attribute initialized the text field with the specified value.
- The **onChange** event is triggered if the user changes the value.
- This occurs when the focus leaves the field, not when each character is typed.
- The JavaScript keyword **this** identifies the object in which the word is specified.
- In this case, this object is the input element, which is passed to the function as a parameter.



Copyright 2010. Manta Technologies Inc.

55

- Suppose the function is defined as follows:

```
function check(x) {
    :
}
```

- The function identifies the passed object by the name **x**.
- As a result, the following references are identical within the function:


```
var data = x.value;
var data = window.document.forms[0].referredfrom.value;
```
- In function terms, **x** is a parameter, while **window.document.forms[0].referredfrom** is a global variable.
- Both are input objects, not string variables.



Copyright 2010. Manta Technologies Inc.

56

- The previous example passed a single form element.
- You can pass the entire form (together with all elements) by using the name **this.form**, as follows:

```
<input type="button" value="Check" onClick="validate(this.form)" />
```

- When this button is clicked, the form containing this button is passed as a parameter to function **validate**.



Copyright 2010. Manta Technologies Inc.

57

- Suppose that the **validate** function is defined as follows:

```
function validate(inForm) {  
    :  
}
```

- Also suppose the document contains a single form.
- Then, within this function, the name **inForm** is synonymous with **window.document.forms[0]**.



Copyright 2010. Manta Technologies Inc.

58

Editing Input Data

- Now that you have seen how to reference the elements within a form and retrieve their values, we can turn our attention to the JavaScript statements used to edit input data.



Copyright 2010. Manta Technologies Inc.

59

Performing Validity Checks

- You may occasionally need to write a validity-checking program that is unique to a particular application or input field.
- In general, however, you will find yourself performing the same tests on similar fields.
- For most text fields, for example, the only test you might perform is to check whether the user entered data.
- For fields that are expected to contain numbers, you can test whether the string contains only the digits 0 through 9 and the characters "." and ",".
- To make such programs easy to reuse, I recommend putting your validity checks into independent functions that can be added to a given JavaScript application as needed.
- This technique also makes the overall program easier to code and understand.
- We will start off your collection with a set of functions that you can use in typical ebusiness applications.



Copyright 2010. Manta Technologies Inc.

60

- Most of the functions that we will create will accept a single parameter, the string value to be tested.
- They will return a boolean value indicating whether the string passes the test or not.
- For example, suppose we want to test whether the input element named **firstname** is not empty.
- We might assign this value to a temporary variable, as follows:

```
var firstName = document.forms[0].firstname.value;
```
- We could then pass this value to the function:

```
if (isEmpty(firstName)) { ...
```
- Because function **isEmpty** returns **true** or **false**, it can be used as the expression in an **if** statement.
- Alternatively, we could skip the temporary variable and reference the field directly:

```
if (isEmpty(document.forms[0].firstname.value)) { ...
```



Copyright 2010. Manta Technologies Inc.

61

Testing Required Fields

- The first function we will create, **isEmpty**, checks whether the object passed contains any data.

```
function isEmpty(field) {
    if ((field == null) || (field == "")) {
        return true;
    } else {
        return false;
    }
}
```

- Here, we have used the JavaScript boolean OR operator (**||**) to test whether either of two conditions is true.
- This function **returns true** if the input string is either null or empty.
- As a result, you can use it in more situations than just checking an input field (where the string will be empty).



Copyright 2010. Manta Technologies Inc.

62

- When designing similar functions, it is up to you to decide whether **true** represents a "good" or "bad" value.
- Rather than designing **isEmpty** as we did, we could have switched the **true** and **false** values and called the function **isNotEmpty**.
- The important thing is that the name is indicative of the value returned.
- If you have one of these functions, there is no need to create the other.
- The **!** (not) boolean operator switches **true** and **false** values.
- As a result, **!isEmpty(x)** is identical to the proposed function **isNotEmpty(x)**.



Copyright 2010. Manta Technologies Inc.

63

Testing Individual Characters

- A shortfall of the **isEmpty** function is that it returns **false** if the string is not empty, but nevertheless contains nothing but spaces.
- We can solve that problem by testing each character in turn, to see if it is equal to the space character (" ").
- Following is the complete function to perform this test:

```
function isBlank(value) {
    for (var i = 0; i < value.length; i++) {
        if (value.charAt(i) != " ") {
            return false;
        }
    }
    return true;
}
```

- The loop tests each character in turn, from character 0 up to the last character of the string.
- It returns **false** if any character is not a blank.
- Otherwise, it returns **true**.



Copyright 2010. Manta Technologies Inc.

64

- An empty string (which has length 0) also returns **true**.
- This function will not work with null strings, however, because the length of a null string is undefined.
- To test for null strings, empty strings, and blank strings, you will need to use a combination of the **isBlank** and **isEmpty** functions.



Copyright 2010. Manta Technologies Inc.

65

Testing Numbers

- We can use a variation of the **isBlank** loop to test whether each character meets specified criteria.
- The following function tests whether the string contains only digits.

```
function isPositiveInteger(value) {
    for (var i = 0; i < value.length; i++) {
        if ((value.charAt(i) < "0") || (value.charAt(i) > "9")) {
            return false;
        }
    }
    return true;
}
```

- This makes use of the fact that the digits 0 - 9 correspond to consecutive ASCII characters.
- So, any character that is less than zero or greater than nine is not a digit.



Copyright 2010. Manta Technologies Inc.

66

- The following function goes a step further to also accept a leading - or +.
- In addition, it uses a flag to accept one — and only one — decimal point.

```
function isNumeric(value) {
    var index1;
    var oneChar = value.charAt(0);
    var foundDot = false;
    if ((oneChar == "-" || oneChar == "+")) {
        index1 = 1;
    } else {
        index1 = 0;
    }
    for (var i = index1; i < value.length; i++) {
        oneChar = value.charAt(i);
        if ((oneChar == ".") && !foundDot) {
            foundDot = true;
        } else {
            if ((oneChar < "0") || (oneChar > "9")) {
                return false;
            }
        }
    }
    return true;
}
```



Copyright 2010. Manta Technologies Inc.

67

Custom Format Checking

- More complex validation functions can be constructed for input fields that must follow other predefined rules.
- Our sample form contains four examples.
 - The telephone number
 - The state abbreviation
 - The zip code
 - The email address



Copyright 2010. Manta Technologies Inc.

68

Telephone Numbers

- In the case of the telephone number, you might check for only digits and special characters, such as "(", ")", "-", " ", and ".".
- Or, you might attempt to match particular formats, such as **(nnn) nnn-nnnn**
- Alternatively, you can be flexible in what you accept, and transform the telephone number into a standard form for the CGI program.



Copyright 2010. Manta Technologies Inc.

69

E-Mail Addresses

- Similar strategies exist for the email address.
- The address should contain only letters, numbers, periods, hyphens, and underscores.
- It should also contain a single "@" character that appears in the string no earlier than position 1.
- At least one period is required after the "@".
- You might also check that the address ends in ".com", ".org", or one of the other recognized names.



Copyright 2010. Manta Technologies Inc.

70

Zip Codes

- For the zip code, the problem is complicated by whether you accept 5- or 9-digit zip codes and non-U.S. mailing codes.
- You may also have a server-based program available that checks whether the zip code is valid, given the other address information.



Copyright 2010. Manta Technologies Inc.

71

State Codes

- For the state abbreviation, we could compare what the user entered with a table of acceptable values.
- When the user is required to enter only certain values, however, it is better to use a pull-down list than a text field.
- Using a pull-down list, the user has no choice other than to select an acceptable value.



Copyright 2010. Manta Technologies Inc.

72

Validation Functions

- The Internet is an excellent source for JavaScript validation functions.
- Simply use your favorite search engine to search for "JavaScript."
- You will be directed to many sites that offer free downloads of JavaScript functions to perform various tasks.



Copyright 2010. Manta Technologies Inc.

73

Displaying Error Messages

- Once you have constructed a validation function for a particular field, the overall validation routine can be built using a series of tests in the following format:

```
if (fieldNotValid(field)) {  
    Display error message  
    Set error flag on  
}
```



Copyright 2010. Manta Technologies Inc.

74

- In our validation routine, we will use a global flag that can be tested to determine whether any error has been found.
- It is defined as follows:

```
var everythingOK = true;
```

- If any error is encountered with the user's input, we will set this flag to **false**.



Copyright 2010. Manta Technologies Inc.

75

- The easiest way to display an error message is by using the **alert** method.
- For example, we could test the first text field as follows:

```
if (isBlank(inForm.firstname.value)) {  
    alert("You need to enter a first name.");  
    everythingOK = false;  
}
```

- The major disadvantage of using **alert** is that the user must click **OK** to close the alert window before being allowed to correct the underlying screen.
- This is acceptable if there is a single input field.
- With multiple fields, however, you cannot expect the user to remember multiple error messages that are all displayed before he or she can correct any of the problems.



Copyright 2010. Manta Technologies Inc.

76

- A better strategy is to display each error message within the original document, next to the field in error.

Dr. ☐ Mr. ☐ Mrs. ☐ Ms. ☐

First Name:

You need to supply a first name.

Last Name:

- There are at least four ways to handle such messages.



Copyright 2010. Manta Technologies Inc.

77

Method 1: Rewrite the Screen

- The "brute force" method is to rewrite the entire document, adding error messages at the appropriate points.
- One complication is that you will need to capture all user input so that you can prime the input fields during the second display.



Copyright 2010. Manta Technologies Inc.

78

- For example, following is the original definition of the **firstname** field:

```
<tr>
  <td align="right">First Name:</td>
  <td><input name="firstname" size="40" maxlength="40" /></td>
</tr>
```

- We can extract what the user typed with a statement like the following:
var firstName = myForm.firstname.value;
- If this field is left blank, we should write the following string as part of the new page.

```
HTMLOutput += '<tr><td align="right">First Name:</td>';
HTMLOutput += '<td><input name="firstname" size="40"';
HTMLOutput += ' maxlength="40" value="' + firstName + '"/>';
HTMLOutput += '</td></tr><tr>&nbsp;</td>';
HTMLOutput += '<td><span style="color:red;">';
HTMLOutput += 'You need to supply a first name.';
HTMLOutput += '</span></td></tr>';
```

- Notice that we primed the input field with the value entered by the user and added an error message within a new row.
- A similar technique could be used for each field before writing the page as a whole.



Copyright 2010. Manta Technologies Inc.

79

Method 2: Use an Inline Frame

- The second method involves saving space in the form for potential error messages.
- This can be accomplished using inline frames, like the following:

```
<iframe scrolling="no" height="30px" width="300px"
  frameborder="0" name="error1" src="blankwht.htm">
</iframe>
```
- Here, file **blankwht.htm** is a predefined HTML blank document.
- As a result, the frame is initially blank.
- A separate frame will be used for each input field, so that each error message can be positioned under the corresponding field.
- Because the inline frame definition included **name="error1"**, we can write to it using JavaScript statements like the following:

```
var HTMLstart = "<html><head></head><body>";
var HTMLend = "</body></html>";
var errorMsg1 = "You need to supply a first name.";
window.error1.document.open();
window.error1.document.write(HTMLstart + errorMsg1 + HTMLend);
window.error1.document.close();
```
- Because each inline frame is treated like a separate document, we need to write a complete HTML page to it.



Copyright 2010. Manta Technologies Inc.

80

- The advantage of using an inline frame is that you do not have to rewrite the complete document to display an error message.
- Instead, just write the message to the appropriate inline frame.
- One disadvantage of using an inline frame is that you must remember to clear the message when the user corrects the form data.
- Another disadvantage is that using inline frames complicates the document design, since you must leave spaces for the frames at the appropriate spots.



Copyright 2010. Manta Technologies Inc.

81

Method 3: Use a Hidden Message

- Suppose you include the following error message just below the first name input field:

```
<div id="errorMsg1" style="visibility:hidden;color:red;">
  You need to supply a first name.</div>
```
- The property **visibility:hidden** causes the message to be hidden when the document is displayed.
- If an error is encountered, the following JavaScript statement can be used in Internet Explorer to make the statement visible:

```
errorMsg1.style.visibility = "visible";
```
- A similar technique could be used to dynamically change the color of elements (such as the field labels) or cause elements to blink.
- In addition to being the easiest of the methods to implement, this method is also very similar to a technique you may have used with 5250 displays to display an error message by setting an indicator on or off.



Copyright 2010. Manta Technologies Inc.

82

- Today, all popular browsers support the ability to dynamically change not only CSS properties but also the HTML itself.
- However, the technique that has become standard is not the dynamic HTML feature of Internet Explorer, but the use of XML methods to manipulate DOM objects.
- In our example, recall that we coded **id="errorMsg1"** on the statement to be modified dynamically.
- This statement can be found in the HTML using the following reference:

```
document.getElementById("errorMsg1")
```
- The visibility attribute can then be changed using:

```
document.getElementById("errorMsg1").style.visibility = "visible";
```
- While this is more complex, it is accepted by all popular browsers, including Internet Explorer.



Copyright 2010. Manta Technologies Inc.

83

Method 4: Dynamically Updating the DOM

- In addition to dynamically changing a property of an HTML element, you can also change its contents.
- For example, suppose we have defined an empty block element like this:

```
<div id="errorMsg1"></div>
```
- This element can be located and modified with the following JavaScript statements:

```
var lineAfterFirstField = document.getElementById("errorMsg1");  
lineAfterFirstField.innerHTML = "You need to enter a first name."
```
- The **innerHTML** property of any HTML object is simply the HTML child elements contained within the object.
- The result of the previous statements is to put a message within the existing **<div>** element.
- This technique is better than changing the visibility if there are multiple messages that might be displayed in the same area.



Copyright 2010. Manta Technologies Inc.

84

Invoking the Validity Checker

- We will now consider how the existing HTML document should be modified to invoke our validation program.
- Currently, the form data is sent to the CGI program when the user clicks the submit button at the bottom of the screen.

```
<input type="submit" value="Send" />
```

- The data is sent to the CGI program that is identified by the **<action>** attribute of the **<form>** element:

```
<form method="post" action="cgi-bin/mycgipgm.pgm" name="userData">
```

- Rather than sending the data immediately, we first want to invoke a JavaScript function that we will call **validate**.
- Only if all fields pass the validity check do we want to invoke the CGI program.



Copyright 2010. Manta Technologies Inc.

85

- You may be tempted to invoke the **validate** function by associating an **onClick** event with the submit button, as follows:

```
<input type="submit" value="Send" onClick="validate(this.form)" />
```

- The problem with this approach is that, while it calls the validation function, it does not stop the form from being submitted.
- While you can remove the **action** attribute from the **<form>** element, it then becomes your responsibility to invoke the CGI program and pass it the appropriate query string once you have determined that all fields contain valid data.
- Instead, a much easier method is available.



Copyright 2010. Manta Technologies Inc.

86

Using an onSubmit Event

- Rather than associating an **onClick** event with the submit button, a better technique is to associate an **onSubmit** event with the form, as follows:

```
<form method="post" action="cgi-bin/mycgipgm.pgm"
      name="userData" onSubmit="return validate(this.form)">
```

- The **onSubmit** event is triggered when the user clicks any submit button within this form.
- The **validate** function is then invoked.
- A similar event, **onReset**, can be associated with reset buttons.
- The **validate** function will be designed to return a boolean value.
 - If all fields are valid, it will return **true** and the browser will continue, passing the form data to the specified CGI program or servlet using the method specified (**get** or **post**).
 - If any field is not valid, the **validate** function will return a value of **false**. As a result, the browser will interrupt its normal processing and **NOT** invoke the CGI program.



Copyright 2010. Manta Technologies Inc.

87

- The coding within the **onSubmit** attribute is a bit tricky.
- The reason is that everything between the quotation marks in an event definition such as **onSubmit="..."** is processed like a function.
- To continue with submission, the **onSubmit** expression must evaluate to **return true**, which is then executed to return the value **true** as the result of the event.
- Similarly, to cancel the submission, the **onSubmit** expression must evaluate to **return false**.
- Because the **validate** function will return the boolean value **true** or **false**, we coded the **onSubmit** attribute with the "return," as follows:

```
onSubmit="return validate(this.form) "
```

- Be aware that having **validate** return the string "return false" will not achieve the desired result, since the result will be treated as a literal string rather than as a JavaScript statement.



Copyright 2010. Manta Technologies Inc.

88

- The validation function will then have the following overall structure:

```
function validate(inForm) {  
    var everythingOK = true;  
    Check field 0  
    Check field 1  
    Check field 2  
    :  
    return everythingOK;  
}
```

- Because of the way the individual validation routines were designed, it is an easy matter to plug in the test for each field.
- If any test finds a bad field, it sets **everythingOK** to **false**, which is then returned as the value of the overall function.
- If all tests are successful, **everythingOK** remains **true**, which is the value returned.



Copyright 2010. Manta Technologies Inc.

89

The Complete Example

- You can get the complete program at
<http://www.mantanow.com/examples/formExample.htm>



Copyright 2010. Manta Technologies Inc.

90

Questions

whansen@mantatech.com



Copyright 2010. Manta Technologies Inc.

91