



Javascript Forms Validation
Processing with
NET+OS's Advanced Web Server

1 Document History

Date	Version	Change Description	
02/11/08	V1.0	Initial entry/outline	
2/13/08	V1.1	Fill in sections	
2/15/08	V1.2	Add section explaining application	
2/21/08	V1.3	Add in edits	

2 Table of Contents

1	Document History	2
2	Table of Contents	3
3	Introduction	4
3.1	Problem Solved	4
3.2	Audience	4
3.3	Assumptions	4
3.4	Scope	5
3.5	Theory of Operation	5
4	Basics	6
4.1	How might you do this in native HTML	6
4.2	AWS ways of doing things	8
4.3	AWS stub functions vs. javascript	11
5	javascript functions	11
5.1	Where to place them	11
5.2	Accessing forms	11
5.3	Accessing forms' elements	11
6	Example Application Explanation	11
7	Conclusion	12
8	Appendix	12
8.1	Glossary of terms	12

3 Introduction

This document describes a method for validating forms served by a NET+OS Advanced Web Server (AWS) using javascript.

3.1 Problem Solved

You have created a web-based system, based on web pages and served by an AWS-based web server. This web server is running on one of Digi International's embedded devices. Before any of the forms in the application are posted to the server, you'd like to have the page validate the form's fields. Validated might include any of the following:

- Ensure that certain required fields are not blank
- Ensure that certain invalid combinations are not selected
- Ensure that certain text fields do not contain any invalid characters

Further, you'd like instant feedback sent to the web user, allowing them to correct the issues, thus allowing the form to be posted to the web server.

This paper describes a method for doing all of this along with some recommendations on debugging these types of applications.

3.2 Audience

This document is intended for users with experience developing applications using either of Digi International's NET+OS development environments. This includes both the Green Hills-based and the GNU-based environments. The user should have some experience in developing web-based NET+OS applications using AWS and its pbuilder utility. Additionally to fully utilize this document the user should have at least a basic understanding of writing web pages using html and writing javascript routines.

3.3 Assumptions

This document assumes that the user has access to a Digi International NET+OS-based development environment and embedded development board for trying out the recommendations outlined in this document. You will be able to get some knowledge from reading this document only, but actually developing a test application and deploying to an embedded device will afford the reader a deeper understanding of the material presented herein.

The validation techniques discussed in this document utilize the javascript programming language. The aforementioned techniques do not in any way, shape or form utilize the java programming language. The reader needs to remember to keep javascript and java separate and understand that this document discusses javascript as opposed to java.

3.4 Scope

This document presents techniques for validating forms served by a Digi International NET+OS-based AWS-containing web server. The techniques contained herein are presenting alternative ways of using existing technologies.

The following items are outside the scope of this document:

- A tutorial on javascript programming
- A tutorial on html (web) programming
- A tutorial on C programming
- A tutorial on developing applications using make and cvs
- A tutorial in tcp/ip techniques
- A tutorial on web design
- Forms validation using the Basic Web Server (BWS)

3.5 Theory of Operation

Many devices and applications today, include one or more web pages as a method for collecting user input. In the case of an embedded device, this input might be used to control the operation of an electrical, mechanical or electro/mechanical device that is linked to the embedded device. Conversely, the user input might be used to set informational fields in the embedded device (ip address, serial number, serial port name...etc.). Given the criticality of the user input, the application writer might want to validate and/or edit the input of the user before submitting the information to the device. In this way, only quality data is sent on to the device.

The javascript functions and code described in this document are used to access web pages, forms on those web pages and elements (fields) on the forms. After accessing the elements, additional javascript code is used to compare the contents or settings of the fields under scrutiny against known valid contents and settings. Additionally the javascript code can compare the contents or setting of one field against that of another, thus ensuring that no logical incompatibilities exist between fields. After all fields, that need validation are validated, the javascript function calls the web page's submit method thus causing the page to be posted to the web server. If the validation fails, the web page is not posted. Additionally, if fields were set in an invalid way, the javascript functions can set the fields back to some known and valid state, allowing the user to rethink their submissions.

The main "trick" to validating forms in html is to "catch" the submission processing before the form is actually submitted to the web server. While in this purgatory-like mode, javascript functions can look at the fields of the page and make assessments about their contents. If the page passes validation, submission is allowed to continue, as though nothing happened. If, on the other hand validation fails, you want to be able to return control of the web page to the user, without submitting its contents to the web server that served the page. This affords the user the ability to make modifications to the form's

fields. After updating the form's fields, the hope is that the validation step will succeed and the form will be successfully submitted to the server. If it still fails, though, the page will, again be returned to the user for additional scrutiny.

4 Basics

Given the fact that some of the techniques need to be performed differently under AWS vs native html, this document spends a little time and space first discussing how these techniques might be done using native html. Then this document discusses how these techniques are modified to work under AWS. Last this document points out the distinction between AWS stub functions and javascript functions.

4.1 *How might you do this in native HTML*

For this description, I am using a simple html form to show the basics for html forms validation. To see more extensive use of javascript, please take a look at the application that accompanies this document.

The web application I am using contains a form that allows you to select your favorite pet. The validation portion has the javascript declare the form invalid, if the user selects canary. If any one or more of the other pets are selected, the form is declared valid and submitted to the server. Though simple, I believe this demonstrates the technique in question.

```
<html>
<head>
<title>Test application</title>
<script language="javascript">
// javascript submission function
function doTheSubmit()
{
  myPet = document.getElementById("myPetsCanaryChkbox");
  if(myPet.checked == true)
  {
    alert("sorry, Canary not a valid selection");
    myPet.checked = false;
    return false;
  }
  else
  {
    alert("Pet selection accepted");
    return true;
    theForm = document.getElementById("myForm");
    theForm.submit();
  }
}
</script>
</head>
```

```
<body>
<form name="myForm" method="post" action="submitPage.htm"
onsubmit="return doTheSubmit();">
<input type="checkbox" name="myPets" value="myPetsDogChkbox"
id="myPetsDogChkbox">
Dog
<br>
<input type="checkbox" name="myPets" value="myPetsCatChkbox"
id="myPetsCatChkbox">
Cat
<br>

<input type="checkbox" name="myPets" value="myPetsCanaryChkbox"
id="myPetsCanaryChkbox">
Canary
<br>
<input type="checkbox" name="myPets" value="myPetsPythonChkbox"
id="myPetsPythonChkbox">
Python
<br>
<input type="checkbox" name="myPets" value="myPetsGerbilChkbox"
id="myPetsGerbilChkbox">
Gerbil
<br>
<input type="checkbox" name="myPets" value="myPetsWeaselChkbox"
id="myPetsWeaselChkbox">
Weasel
<br>
<input type="submit" value="submit">
</form>
</body>
</html>
```

As you can see, the web page contains a form entitled myForm. myForm contains a list of pets, set up as check boxes, for the user to select. Additionally you notice that the <form> tag, contains an option entitled onsubmit.

Next you'll notice that up at the top of the html code is the command function. Between the { and the } of the function is some simple javascript code. You'll also notice that the function is entitled doTheSubmit().

Now to tie this portion up, notice that in the <form> tag, the onsubmit option is assigned the function name doTheSubmit. This instructs the forms submission engine to hold off submitting the page to the web server, until the function dotheSubmit() has been executed. Further, if the function dotheSubmit returns false, do not submit the form. If, on the other hand, the function returns true, then do submit the form.

Thus the javascript validation function is tied to the html code and the form in specific by the onsubmit option of the <form> tag.

If you look at the function onsubmit itself, you'll notice the use of the method getElementById(). This allows javascript to access and isolate one element, from the form, for processing. In this case we are accessing the canary element from the group of checkboxes. Notice that we check to see whether the checked element of the canary check box is set to true. If it is true, we deem this submission invalid. Further, since selecting canary is invalid, we set checked to not checked (by setting checked to false) and return false. Returning false tells the browser engine to "hold off" from submitting the form. Last notice that if the checked field of the canary element is not checked (checked is set to false) we deem this form validated, and return true. This signals the browser engine to continue with submitting this form.

4.2 AWS ways of doing things

There are a couple of stumbling blocks that you will run into when trying to migrate the code above into an AWS environment. First, the RpFormHeader tag can not contain a name option. Second, the RpFormHeader tag can not contain an onsubmit option. These may sound like deal-breakers, but there are ways around these issues. In a nut shell, javascript, stores the forms of a web page in an array attached to the web page "document". Thus by knowing the order the forms appear on the page, you can access the form(s) as array elements in the forms array. (document.forms[0] accesses the first form on a page) In the case of the missing onsubmit option, we recommend replacing the use of the onsubmit option to the RpFormHeader tag, by replacing the <input type="submit"> tag with an anchor <a> and an href pointing to the javascript validation function. The code below should help to make this all a lot clearer.

```
<html>
<head>
<!--
//
// example html page demonstrating javascript forms processing
// copyright © 2008 Digi International
//
// -->
<title> this page validates a form using javascript</title>
<script language="javascript">
//
// javascript function demonstrating both forms validation and forms
// submission
//
//
function dotheSubmit()
{
    myPet = document.getElementById("myPetsCanaryChkbox");
```



```
// don't allow canary to be selected
if(myPet.checked == true)
{
    alert("sorry Canary not allowed");
    myPet.checked = false;
}
Else
{
    alert("Pet selection accepted, form submitted");
    document.forms[0].submit();
}
}
</script>
</head>
<body>
</body>
<!-- RpformHeader RpNextPage=pgvalidateMyForm RpFunctionPtr=initFormPage -->
<form name="myForm" onsubmit="return doTheSubmit();">
<!-- RpFormInput type="text" name="thePhoneNumber" size="15" maxlength="15"
value="(123) 456-7890"
    RpGetType=Function RpGetPtr=getMyPhoneNumber
    RpSetType=Function RpSetPtr=setMyPhoneNumber -->
<input type="text" name="thePhoneNumber" id="thePhoneNumber" value="(123) 456-
7890">
<!-- RpEnd -->
<br>
<br>
What type of pet do you own?
<br>
<!-- RpFormInput TYPE=checkbox NAME="myPetsDogChkbox"
    RpGetType=Function RpGetPtr=getDogCheckBox
    RpSetType=Function RpSetPtr=setDogCheckBox -->
<input type="checkbox" name="myPets" value="myPetsDogChkbox"
id="myPetsDogChkbox" -->
<!-- RpEnd -->
Dog
<br>
<!-- RpFormInput TYPE=checkbox NAME="myPetsCatChkbox"
    RpGetType=Function RpGetPtr=getCatCheckBox
    RpSetType=Function RpSetPtr=setCatCheckBox -->
<input type="checkbox" name="myPets" value="myPetsCatChkbox"
id="myPetsCatChkbox" -->
<!-- RpEnd -->
Cat
<br>
<!-- RpFormInput TYPE=checkbox NAME="myPetsCanaryChkbox"
```

```

        RpGetType=Function RpGetPtr=getCanaryCheckBox
        RpSetType=Function RpSetPtr=setCanaryCheckBox -->
<input type="checkbox" name="myPets" value="myPetsCanaryChkbox"
id="myPetsCanaryChkbox" -->
<!-- RpEnd -->
Canary
<br>
<!-- RpFormInput TYPE=checkbox NAME="myPetsPythonChkbox"
        RpGetType=Function RpGetPtr=getPythonCheckBox
        RpSetType=Function RpSetPtr=setPythonCheckBox -->
<input type="checkbox" name="myPets" value="myPetsPythonChkbox"
id="myPetsPythonChkbox" -->
<!-- RpEnd -->
Python
<br>
<!-- RpFormInput TYPE=checkbox NAME="myPetsGerbilChkbox"
        RpGetType=Function RpGetPtr=getGerbilCheckBox
        RpSetType=Function RpSetPtr=setGerbilCheckBox -->
<input type="checkbox" name="myPets" value="myPetsGerbilChkbox"
id="myPetsGerbilChkbox" -->
<!-- RpEnd -->
Gerbil
<br>
<!-- RpFormInput TYPE=checkbox NAME="myPetsWeaselChkbox"
        RpGetType=Function RpGetPtr=getWeaselCheckBox
        RpSetType=Function RpSetPtr=setWeaselCheckBox -->
<input type="checkbox" name="myPets" value="myPetsWeaselChkbox"
id="myPetsWeaselChkbox" -->
<!-- RpEnd -->
Weasel
<br>
<br>
<a href="javascript" doTheSubmit();"> Submit the form </a>
</form>
<!-- RpFormEnd -->
</body>
</html>

```

Notice that in the anchor tag, that the href, instead of referencing a URL, is instead referencing the javascript function "doTheSubmit()". Also notice that the javascript is similar between the pure html and the AWS form. The main difference, as mentioned before, is that the pure html file has access to a named form. The AWS instance, must access the form through the web pages (document) forms array. Additionally, if you are familiar with AWS comment tags, you'll notice the addition of the comment tags surrounding the html tags.

4.3 AWS stub functions vs. javascript

In AWS, if we are doing forms validation, we are now talking about two sets of functions. It is quite important that we keep them separate. The javascript functions run only within the context of the browser. They do not run in AWS and they are not capable of actually updating device data.

AWS stub functions run only on the device. The “get” functions get device data and returns it, through the AWS engine to the browser. The “put” functions, take data from the browser (validated by the javascript function(s)) and update the device data variables.

Please notice that there is No direct interaction between the AWS stub functions and the javascript functions. They serve two different purposes, in two different places. Further if you want to use AWS and you have device data, your application **MUST** employ stub functions. javascript can NOT be used to update device data. Additionally, if you want to validate your forms and you want to have device data, you **MUST** employ both AWS stub functions and javascript functions.

5 javascript functions

5.1 Where to place them

The javascript functions, to be accessed by the html must be defined prior to their use. Most of the books and web pages that I have scrutinized, have recommended defining the functions within the head section of the html page. You’ll notice that in this example, I have followed that method. We recommend that you define your javascript functions within the head section of your html page.

5.2 Accessing forms

As mentioned above, web pages that are to be processed by the PBuilder utility, can not have a named form. Thus you can not use the getElementById() method of the document. Thus we recommend accessing forms through the document’s forms array, as I did in the example above. Form 1 is document.forms[0], form 1 is document.forms[1]...etc.

5.3 Accessing forms’ elements

Since the Pbuilder utility does not restrict you from naming input elements within the form, you can access these via the getElementById() method. We encourage the use of this method when accessing forms’ elements.

6 Example Application Explanation

The html file of the example application, demonstrates two concepts. As explained above, the href of the anchor tag, references a javascript function that can validate the fields of a form. Further the javascript function can either allow or cancel forms submission, based on the outcome of the validation process.

A second part of the html file demonstrates a method for using javascript to display information on the web page. That is mainly made up of a number of document.write() function calls. Even though you would not include these in a production application, I included them in the sample application, showing a method for debugging the javascript in your application. I found in developing this application, that javascript is quite unforgiving in the area of syntax errors. I was finding that web pages would display with less information than I expected. There was no error information displayed on the page. The method I used to debug these issues was print statements (document.write()) at various points in the file until I identified the errant line. I then used my javascript reference manual to understand why any particular line was failing. Since this technique was helpful to me, I left them in as an aid to your development effort.

7 Conclusion

Though not 100% transparent, we have described to you a method for performing forms validation using javascript functions from within AWS applications. We believe this to be a useful tool in making your web pages more user friendly to your customers.

8 Appendix

8.1 Glossary of terms

AWS – advanced web server. An embedded web server included with Digi International's NET+OS product

BWS - basic web server. A simpler (than the AWS) embedded web server also included with Digi International's NET+OS product

html – hypertext markup language – a language using tags and used to create web pages

java – an object oriented programming language

javascript – a programming language sometimes used within html-based web pages for validating user input

NET+OS – an embedded operating system and development environment, developed and sold by Digi International

Stub Functions – C callback functions that are called by the AWS either when data is updated on a web page and needs to be updated in a device or when a browser needs to get updated information from the device.

[Example Source Files](#)