

# Aspects of AJAX

---

Published online at <http://www.mathertel.de/AJAX/AJAXeBook.aspx>

By Matthias Hertel, 2005-2007

Version 1.2 published 1. May 2007

# About this book

This book is about an AJAX Framework and an AJAX Engine for JavaScript, XML, SOAP, WSDL und ASP.NET using standard Web Services on the server.

This book is containing the updated articles and samples from my Blog "Aspects of AJAX", available at <http://ajaxaspects.blogspot.com/> together with some new and rewritten articles.

The implementation of the Samples, the AJAX Engine and a lot of web controls can be found on <http://www.mathertel.de/AJAXEngine/>.

## The License

This book and all the articles on my blog are licensed under a Creative Commons Attribution 2.0 License that can be found at <http://creativecommons.org/licenses/by/2.0/de/>.

The software itself is licensed under a BSD style license that can be found at <http://www.mathertel.de/License.aspx>.

## State of this book

This book is still not finished and will be updated and extended from time to time.

You will find more information when reading the Blog or downloading a new copy of this book. There are still a lot of aspects undocumented or undiscovered.

## State of the Software

The AJAX engine is working fine in many projects I do myself and I've heard about and you can use it where ever you want. The license model I've chosen to publish the information and the source code allows also a commercial use of it.

For ASP.NET a set of web controls are available that make using the AJAX technology a lot easier because of the declarative approach that comes with using web controls.

For JAVA the AJAX Engine is available without any tag-libraries.

Since 2006 also other "new" a.k.a. Web 2.0 topics where added.

April 2007

# Abstract

The basic elements for an application using the AJAX technologies, JavaScript and the XMLHttpRequest object, are not difficult to understand and there are many articles on the web that show how to use this object and declare that being AJAX. I think there are a lot more topics that should be understood and talked about.

The right selection from the available technologies as well as a suitable abstraction in using these elements is important for the success of the realization of an application. One main goal behind the architecture of the AJAX engine was to build an AJAX framework that you can reuse every time you want some asynchronous processing or when you need a smart way to refresh information on the current web page.

When targeting applications with some hundred sides and WebServices and with in sum about a thousand methods the developer must have a clear and simple kind and pattern for coding the JavaScript code on the client to avoid errors and to not think about the implementation details. Only by using a simple approach a good quality and maintenance can be achieved.

The idea of the AJAX engine on the client is the simplification of the implementation of the code that we need for implementing a specific functionality on the client. Like with the WebService framework of ASP.NET on the server the details of communication over SOAP on the client are completely hidden from the developer and also the recurring code portion is only realized once in a central place. Also all details about the different implementations of the XMLHttpRequest object in Internet Explorer or the Firefox browsers are hidden from your code.

By using web controls or tag-libraries again a higher level of abstraction and more productivity for the developer can be reached. Because these web controls have to deploy JavaScript code to the client to avoid round-trips and enable the local functionality in the browser the JavaScript technology, especially JavaScript Behaviors are a substantial part of an AJAX infrastructure.

The visual effects library adds some more polished User interface elements that can be used together with AJAX or as standalone client side components.

# Index

|                                                                    |           |
|--------------------------------------------------------------------|-----------|
| About this book .....                                              | 2         |
| The License.....                                                   | 2         |
| State of this book.....                                            | 2         |
| State of the Software .....                                        | 2         |
| Abstract.....                                                      | 3         |
| Index .....                                                        | 4         |
| History .....                                                      | 7         |
| <br>                                                               |           |
| <b>Asynchronous programming .....</b>                              | <b>8</b>  |
| Why we need asynchronous programming.....                          | 9         |
| Old style programming .....                                        | 9         |
| Old style programming using HTML.....                              | 9         |
| Asynchronous programming in the browser .....                      | 12        |
| How it's done and why it doesn't work.....                         | 12        |
| Asynchronous programming on the server .....                       | 15        |
| <br>                                                               |           |
| <b>Native AJAX Programming .....</b>                               | <b>16</b> |
| The XMLHttpRequest Object.....                                     | 17        |
| AJAXing the CalcFactors Example.....                               | 17        |
| <br>                                                               |           |
| <b>Client-Server Protocols .....</b>                               | <b>20</b> |
| Best Practices for a WebService implementation.....                | 20        |
| SOAP was made for AJAX.....                                        | 21        |
| Using WebServices in AJAX applications .....                       | 22        |
| A SOAP client for JavaScript.....                                  | 25        |
| Good tools for analyzing network problems.....                     | 28        |
| Generating JavaScript Proxies in ASP.NET .....                     | 29        |
| Generating JavaScript Proxies in JAVA.....                         | 30        |
| Proxy Generator supported datatypes.....                           | 31        |
| <br>                                                               |           |
| <b>The AJAX Engine .....</b>                                       | <b>33</b> |
| Overview.....                                                      | 33        |
| AJAX Actions .....                                                 | 36        |
| AJAX Action Reference.....                                         | 36        |
| Starting an AJAX Action .....                                      | 37        |
| Handling Exceptions .....                                          | 38        |
| Examples that use the AJAX Engine directly .....                   | 40        |
| The AJAX prime factors sample.....                                 | 40        |
| A auto-completion textbox and lookup for city names.....           | 41        |
| An AJAX Engine for Java.....                                       | 43        |
| JavaScript & ajax.js.....                                          | 43        |
| WebServices.....                                                   | 43        |
| XSLT / WebService Proxies.....                                     | 43        |
| Download.....                                                      | 43        |
| AJAX and Forms .....                                               | 44        |
| AJAX Form Services.....                                            | 45        |
| Sample for AJAX Forms.....                                         | 46        |
| Use web services with multiple parameters in the AJAX Engine... .. | 48        |
| Application Aspects .....                                          | 50        |
| Model View Controller (MVC) Pattern - Thinking in patterns .....   | 50        |
| Developer's productivity – Layers of abstraction.....              | 52        |

|                                                                               |           |
|-------------------------------------------------------------------------------|-----------|
| Less waiting on AJAX.....                                                     | 53        |
| Thinking in components.....                                                   | 55        |
| Building AJAX Controls.....                                                   | 55        |
| <b>JavaScript Behaviors.....</b>                                              | <b>56</b> |
| The Behavior mechanism.....                                                   | 56        |
| Building AJAX Controls.....                                                   | 56        |
| Delivering JavaScript functionality.....                                      | 56        |
| Browser specific proprietary behaviors.....                                   | 57        |
| The JavaScript behavior mechanism.....                                        | 59        |
| A step by step instruction.....                                               | 59        |
| Building the JavaScript Behavior Basics.....                                  | 59        |
| Integration into the ASP.NET framework.....                                   | 61        |
| Building JavaScript Behaviors - Properties, Attributes and<br>Parameters..... | 63        |
| Event handling.....                                                           | 65        |
| Details of JavaScript Behavior Definitions.....                               | 68        |
| The common JavaScript include file.....                                       | 69        |
| Cross Browser JavaScript.....                                                 | 71        |
| Introduction.....                                                             | 71        |
| JavaScript Prototypes.....                                                    | 71        |
| JavaScript Prototypes in Mozilla/Firefox.....                                 | 72        |
| Prototypes with HTML objects.....                                             | 72        |
| Firefox compatibility.....                                                    | 73        |
| Conclusion.....                                                               | 73        |
| <b>Building JavaScript enabled web controls ....</b>                          | <b>74</b> |
| Delivering controls to the browser.....                                       | 74        |
| Registering Script includes.....                                              | 75        |
| Registering Script includes without a form element.....                       | 75        |
| Parameters.....                                                               | 76        |
| HTML Code.....                                                                | 77        |
| Programming the Behaviour.....                                                | 77        |
| Registering the script includes.....                                          | 77        |
| Integration into ASP.NET.....                                                 | 78        |
| <b>Connecting Controls.....</b>                                               | <b>79</b> |
| The Page Properties mechanism.....                                            | 80        |
| The Connection Test Sample.....                                               | 80        |
| Simple Controls using page properties.....                                    | 80        |
| The Back Button Problem of AJAX applications.....                             | 83        |
| What the back button does - and why not.....                                  | 83        |
| The favorite problem.....                                                     | 83        |
| Meaningful urls.....                                                          | 84        |
| An Implementation.....                                                        | 85        |
| <b>AJAX enabled web controls.....</b>                                         | <b>87</b> |
| AJAX Actions inside JavaScript Behaviours.....                                | 87        |
| Using AJAX enabled controls in ASP.NET Forms.....                             | 88        |
| The Samples.....                                                              | 89        |
| Custom Validation AJAX Control Sample.....                                    | 91        |
| Displaying huge tables using AJAX.....                                        | 93        |
| The DataTablePager Control.....                                               | 94        |

|                                                                  |            |
|------------------------------------------------------------------|------------|
| The DataTable Control.....                                       | 94         |
| The TableData WebService .....                                   | 94         |
| Tuning the TableData .....                                       | 95         |
| An AJAX enabled bible reader.....                                | 96         |
| A Walk-Through.....                                              | 96         |
| Technology .....                                                 | 97         |
| Treeview AJAX Control .....                                      | 99         |
| An AJAX based Tree View for the Bible .....                      | 101        |
| <b>Visual Effects Library .....</b>                              | <b>103</b> |
| Why AJAX needs visual effects.....                               | 103        |
| HTML + CSS Shadow Effect with real transparency.....             | 104        |
| HTML elements with rounded corners .....                         | 106        |
| Simple sliding sample to move HTML elements around.....          | 108        |
| Drag and drop HTML objects around using CSS and JavaScript ..... | 109        |
| PopUp Information.....                                           | 112        |
| Building a AJAX enabled popup control.....                       | 114        |
| <b>Some HTML and http basics .....</b>                           | <b>115</b> |
| Caching with AJAX applications.....                              | 115        |
| <b>Listings .....</b>                                            | <b>118</b> |
| ~/ajaxcore/GetJavaScriptProxy.aspx .....                         | 118        |
| ~/ajaxcore/wSDL.xslt .....                                       | 119        |
| ~/ajaxcore/ajax.js.....                                          | 122        |
| ~/controls/jcl.js .....                                          | 132        |
| <b>JavaScript Proxy Reference.....</b>                           | <b>138</b> |
| The proxies and service objects.....                             | 138        |
| <b>DataConnections Reference .....</b>                           | <b>140</b> |
| <b>Links .....</b>                                               | <b>141</b> |
| Tools .....                                                      | 141        |
| Behaviors.....                                                   | 141        |
| Cross Browser implementation tips.....                           | 141        |
| Standards .....                                                  | 141        |
| WebServices.....                                                 | 142        |
| JavaScript .....                                                 | 142        |

# History

I started programming for the Web back in 1998 by learning early versions of HTML and JavaScript and using the browsers of these years. I was head of development of a company building an ERP system for the German market and saw the chance to use web technologies for the next version.

I really started programming in 1977 by building a 1 MHZ 6502 board using assembly op-codes.

When IE 5.0 (and later IE 5.5) was released by Microsoft together with the XMLHttpRequest ActiveX object we saw the chance to build an ERP client layer with a rich user experience inspired by Outlook Web Access and the early SOAP specifications. By using a SOAP-based synchronous background data mechanism instead of the common http round-trip using form elements we reached the look & feel of regular desktop applications by using web technologies. We never had a specific name for that piece of our technology.

One of the biggest concerns we had to master was productivity of the developers. We had no huge budget and only 2 years to come out with the first major version... and the project was successful.

When the name AJAX was born early in 2005 the asynchronous way of program flow was new to me. I had many experiences with synchronous JavaScript programming and with synchronous calling WebServices from the browser and this works fine in intranet scenarios where the network just works as expected. Also it is very simple to write down a function that uses the server when calls come back with the result immediately.

When working asynchronously the code gets split into individual pieces of callback methods and event handlers and that may lead to "ugly" readable and highly fragmented code. I searched for something in the available technology stack that helps against this situation and found a declarative way of bundling AJAX coding fragments and settings together as you will see in the AJAX actions you can read about here.

Beyond the basic AJAX programming using JavaScript on the client and Webservice endpoints on the server I also show here how to build WebControls that go further in simplifying the work for the programmer. In the end you can get a good productivity.

# Asynchronous programming

---

This first part is about the problems that arise in simple but long running application tasks and processes and the possible solutions to that general problem. In the end we see how asynchronous programming can be done in web applications.

The mechanism shown here are part of the principles used by AJAX and other Web 2.0 technologies.

If you are an advanced JavaScript programmer already you might already understand how to split long running actions into smaller pieces and working with timers and internal event handlers so you just skip this chapter and start with the chapter Native AJAX Programming.



# Asynchronous programming

## Why we need asynchronous programming

With the introduction of the mouse as input tool the event driven programming won an outstanding meaning in the realization of applications. Programming input sequences and loops for the repetition of tasks in up-to-date programs is no more accepted by the users.

### Old style programming

To illustrate this I realized a HTML page this way of programming as we would have it done in beginning of the 80's (at that time in Microsoft basic on Apple II computers and with line numbers and goto).

```
// calc prime factors
var inputText, outputText;
var prime; // try this factor (only primes will match!)
var number; // product of the remaining factors

while (true) {
  outputText = "";
  inputText = window.prompt("Please enter a number (or 0 to exit):", "");

  if ((inputText == null) || (inputText.length == 0) || (inputText == "0"))
    break;

  prime = 2; // start with 2
  number = parseInt(inputText);

  while ((number > 1) && (prime * prime <= number)) {
    if (number % prime != 0) {
      // try the next factor (slowly)
      prime += 1;
    } else {
      // found a factor !
      outputText = outputText + " " + prime;
      number = number / prime;
    } // if
  } // while

  if (number > 1) {
    // the last factor (a prime) is here.
    outputText = outputText + " " + number;
  }

  window.alert("The factors of " + inputText + " are:" + outputText);
} // while
```

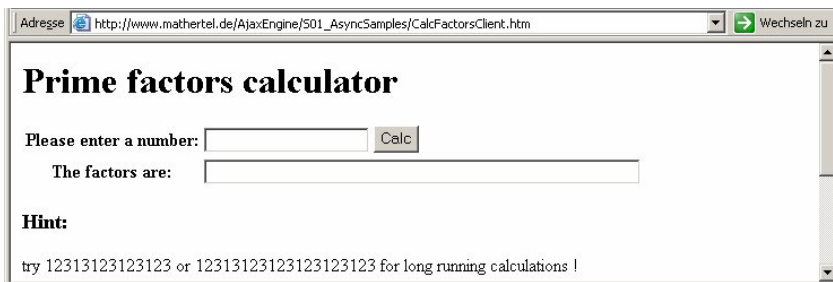
[http://www.mathertel.de/AjaxEngine/S01\\_AsyncSamples/CalcFactorsOld.htm](http://www.mathertel.de/AjaxEngine/S01_AsyncSamples/CalcFactorsOld.htm)

Do you expect that web applications are realized in such a way or that they are present in this way the user? - I don't! Those times are over luckily.

### Old style programming using HTML

Even if one really uses HTML as the in- and output formula the situation doesn't really change:

## Asynchronous programming



The JavaScript coding behind this kind of application is almost the same. The big difference is that instead of coding menus or loops with requests for user input we find an event driven approach.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Strict//EN">
<html>
<head>
  <title>Prime factors calculator</title>
</head>
<body>
  <h1>Prime factors calculator</h1>
  <table>
    <tbody>
      <tr>
        <th><label for="inputField">Please enter a number:</label></th>
        <td><input id="inputField"> <input type="button" value="Calc"
onclick="CalcPrimeFactors()"
        id="Button1" name="Button1"></td>
      </tr>
      <tr>
        <th><label for="outputField">The factors are:</label></th>
        <td><input id="outputField" size="60" disabled="disabled"></td>
      </tr>
    </tbody>
  </table>
  <h3>Hint:</h3>
  <p>try 12313123123123 or 12313123123123123123123 for long running calculations ! </p>

  <script type="text/javascript">

// calc prime factors
function CalcPrimeFactors() {
  var inputText, outputText;
  var prime; // try this factor (only primes will match!)
  var number; // product of the remaining factors

  document.getElementById("outputField").value = "wait...";
  outputText = "";
  inputText = document.getElementById("inputField").value;

  if ((inputText == null) || (inputText.length == 0) || (inputText == "0"))
    return;

  prime = 2; // start with 2
  number = parseInt(inputText);

  while ((number > 1) && (prime * prime <= number)) {
    if (number % prime != 0) {
      // try the next factor (slowly)
      prime += 1;
    } else {
      // found a factor !
      outputText = outputText + " " + prime;
      number = number / prime;
    } // if
  } // while

  if (number > 1) {
    // the last factor (a prime) is here.

```

# Asynchronous programming

```
    outputText = outputText + " " + number;
  }

  document.getElementById("outputField").value = outputText;
} // CalcPrimeFactors

</script>

<hr />
<p>This sample uses HTML and Javascript synchronously.</p>
<p>This page is part of the <a
href="http://ajaxaspects.blogspot.com/">http://ajaxaspects.blogspot.com/</a>
project.</p>
<hr />
</body>
</html>
```

[http://www.mathertel.de/AjaxEngine/S01\\_AsyncSamples/CalcFactorsClient.htm](http://www.mathertel.de/AjaxEngine/S01_AsyncSamples/CalcFactorsClient.htm)

# Asynchronous programming

## Asynchronous programming in the browser

### How it's done and why it doesn't work

Complicated and long running functions have the unpleasant characteristic, that during their execution all the other activities are standing still. In particular really long running functions represent a genuine problem and therefore both browsers the Microsoft Internet Explorer and Firefox are offering the possibility of a brutal termination of execution.



The way out of this dilemma is possible when using a parallel execution of the calculation of the factors and the handling of all user interface events like keystrokes.

In the Browser and with the assistance of JavaScript there is only a very much limited way of parallel execution with the assistance of a timer and events possible. However the problems and the realizations of such parallel algorithms must be suitable for this. When a JavaScript function runs then the Events from the user inputs are no longer directly processed and the application appears blocked again.

### Splitting a long running task into multiple shorter tasks and why it doesn't help

Implementing this idea in a browser can be done in the following approach:

With every keystroke a timeout (`_timer`) is registered to trigger the calculation of the factors. The chosen timeout is just a little bit longer than the time a normal user needs in-between typing the characters. If a new character is typed in, the running timeout is canceled and a new timeout is started. The calculation of the factors remains identical and the look & feel of the application doesn't change.

This is not a real parallel execution of two threads but some kind of pseudo asynchronous programming that can be realized by using JavaScript. Using multiple threads inside a Browser by using JavaScript only cannot be done today.

Here is the JavaScript implementation of this approach:

```
<script type="text/javascript">

var _num = "";
var _timer = null;

function StartCalcPrimeFactors() {
  var inputText = document.getElementById("inputField").value;
  if (_num != inputText) {
    if (_timer != null)
      window.clearTimeout(_timer);
    document.getElementById("outputField").value = "wait...";
    _num = inputText;
    _timer = window.setTimeout("CalcPrimeFactors()", 300, "javascript");
  } // if
}

// calc prime factors
function CalcPrimeFactors() {
  var inputText, outputText;
  var prime; // try this factor (only primes will match!)
  var number; // product of the remaining factors
}
```

## Asynchronous programming

```

_timer = null;
outputText = "";
inputText = document.getElementById("inputField").value;

if ((inputText == null) || (inputText.length == 0) || (inputText == "0"))
    return;

prime = 2; // start with 2
number = parseInt(inputText);

while ((number > 1) && (prime * prime <= number)) {
    if (number % prime != 0) {
        // try the next factor (slowly)
        prime += 1;
    } else {
        // found a factor !
        outputText = outputText + " " + prime;
        number = number / prime;
    } // if
} // while

if (number > 1) {
    // the last factor (a prime) is here.
    outputText = outputText + " " + number;
}

document.getElementById("outputField").value = outputText;
} // CalcPrimeFactors
</script>

```

[http://www.mathertel.de/AjaxEngine/S01\\_AsyncSamples/CalcFactorsAsync1.htm](http://www.mathertel.de/AjaxEngine/S01_AsyncSamples/CalcFactorsAsync1.htm)

For guaranteed short running functions this kind of the implementation is acceptable for the user because the page locks itself while calculating not after every keystroke. The delay in the case, that the calculation is executed is hardly noticed and an input of a further character or a click with the mouse is executed only some milliseconds later.

But with long running functions like with the calculations of the factors of a large number a problem will arise with the time the function needs to finish.

I know 2 Methods of solving this:

1. Divide the long running function into several smaller functions
2. Real parallel execution with the assistance of a server (let's AJAX)

For sure it is possible to changed many algorithms in such a way so that instead of a long running method several shorter steps are lined up. With the sample of the computation of the prime factors this is also possible.

The timer is used in very short timeout intervals to check a new prime number candidate.

```

<script type="text/javascript">

var _num = "";
var _timer = null;

var prime; // try this factor (only primes will match!)
var number; // product of the remaining factors
var outputText;

function StartCalcPrimeFactors() {
    var inputText = document.getElementById("inputField").value;
    if (_num != inputText) {
        if (_timer != null)

```

## Asynchronous programming

```

    window.clearTimeout(_timer);
    document.getElementById("outputField").value = "wait...";
    _num = inputText;

    prime = 2; // start with 2
    number = parseInt(inputText);
    outputText = "";

    _timer = window.setTimeout("CalcPrimeFactors()", 50, "javascript");
  } // if
} // StartCalcPrimeFactors

// calc prime factors
function CalcPrimeFactors() {
  _timer = null;

  if (number == 1) {
    // finished. all factors found
    outputText = outputText + " finished.";
    document.getElementById("outputField").value = outputText;
  } else if (prime * prime > number) {
    // the last factor (a prime) is here.
    outputText = outputText + " " + number + " finished.";
    document.getElementById("outputField").value = outputText;
  } else {
    // Debug: window.status = prime;

    if (number % prime != 0) {
      // try the next factor (a little bit faster)
      prime += (prime == 2 ? 1 : 2);
    } else {
      // found a factor !
      outputText = outputText + " " + prime;
      document.getElementById("outputField").value = outputText;
      number = number / prime;
    } // if
    _timer = window.setTimeout(CalcPrimeFactors, 0, "javascript");
  } // if
} // CalcPrimeFactors
</script>

```

[http://www.mathertel.de/AjaxEngine/S01\\_AsyncSamples/CalcFactorsAsync2.htm](http://www.mathertel.de/AjaxEngine/S01_AsyncSamples/CalcFactorsAsync2.htm)

However, the result is very discouraging because the running time of a complete computation rises up with an intolerable factor so far, that the practice fitness is no longer given.

Beside the problem that everything runs more slowly exists also the problem that local variables cannot be used any more. All information about the current state must be stored in global variables to be available for the next step. That is not very useful kind of programming and again reminds me of the "good old times" of the 80's.

### Using multithreading

All the different ways of implementation up to here are not using any parallel execution at all. That is because JavaScript is not offering any mechanism for starting threads or for controlling a parallel execution. Here the browser platform fails completely.

The server platforms on the other side do offer this kind of approach. Particularly with regard to multiple users requesting for information web servers do have a built-in mechanism for parallel execution based on unrelated

# Asynchronous programming

parallel incoming requests. AJAX uses this fundamental feature of web servers as you will see.

On the client only 2 events will remain and must be processed one after the other:

- The event that starts processing:  
This event is started by the user for example by clicking a button or typing a key.
- The event that ends processing.  
This event is started on the client when the operation is finished on the server and when the result of the server side execution is available in the browser.

On the server the real work is implemented and will execute independent of the client.

The mechanism we need between the client and the server to get the complete work done synchronized is passing 2 messages around: one that initiates the execution on the server and one that comes back to the client with the result of the server's work.

## Asynchronous programming on the server

On the server asynchronous programming is also possible. During the composition of an answer to a Web server request multiple threads can be started in parallel to gather different information parts that will be combined into the (only) answer for the client. Some web server platforms (ASP.NET, Java ...) can be used for this. This kind of programming does not have to do ANYTHING with AJAX but can be used in combination with it.

# Native AJAX Programming

When using an AJAX style of programming the old, classic approach programming functionality must be given up. There is no form submit any more that posts all the client state to the server and requests for a complete new page description using HTML.

Instead of loading several pages until the functionality is done only one page is loaded and will stay in the browser until the end of functionality. With the Ajax model the execution of this web page is processed in 3 different phases:

## Phase 1: Loading the page

During this synchronously implemented phase the client loads the "static" part of the application. That corresponds to displaying a form or a list without any concrete data. (We will see later that it makes a lot of sense in some scenarios to include a first set of data into this first page loading reduce the time until the user can use it and to achieve content recognition for search spiders.)

The code for the page is assembled and delivered to the browser. It makes no big difference if plain HTML, generic JavaScript or JavaScript includes are used.

The http answer of this first call can be delivered in many situations with a hint for the client that the local browser cache can be used to store the page for a while and therefore the server will not be asked again for this code. The displaying of a page in this situation is very fast and efficiently because the bytes are retrieved from the local cache and no network delays occur.

[http://msdn.microsoft.com/library/en-us/dnwebgen/html/ie\\_introfiddler2.asp?frame=true](http://msdn.microsoft.com/library/en-us/dnwebgen/html/ie_introfiddler2.asp?frame=true)

## Phase 2: Loading data from the server using AJAX techniques

This phase is used for retrieving more information from the web server and then combining it into the already delivered html. This can be repeated several times while the initial page is still loaded.

That can be the current list of the emails, (Outlook Web ACCESS, GMail) or the data changed since the last call (GMail) or additional information to a search word (Google Suggest).

There are also non HTML/JavaScript solutions possible. Google Maps uses variable URLs with parameters on image objects to retrieve the bitmaps that are part of the current shown map.

The AJAX model must be considered in both loading phases. In the first phase the necessary (JavaScript) code must be delivered. In the second phase this code must be used to get the information from the server.

## Phase 3: Interaction with the page using AJAX techniques

Web applications need a mechanism to push information back to the server.

The most known and old solution to this is the mechanism of the built-in <form> element that allows to post back the information stored in <input> elements to the server. The answer from the server contains a complete new page that will be displayed.

In contrary the AJAX solution to this is to collect and transfer the pure information that is needed by the server and to retrieve from the server some new information that can be integrated into the already loaded page.



# Native AJAX Programming

We will see how this can be done by using the built-in XMLHttpRequest object. The more comfortable solution will be shown later by using a Webservice.

## The XMLHttpRequest Object

There are multiple mechanisms available that can be used to call the server without totally refreshing the current page:

- An invisible <iframe> element or a frameset with an invisible frame. The url of this element is set to a specific value containing all parameters.
- A <script> element is used to include some script fragments from a dynamic location (not a static JavaScript file). The server creates some JavaScript and by executing the script the results of the call are embedded into the page
- The XMLHttpRequest object is used.

The AJAX Engine always uses the XMLHttpRequest approach because. It is available in most up to date browsers and (I hope) will soon be accepted as a standard. Have a look at

<http://www.w3.org/TR/XMLHttpRequest/>

There you can find a good reference too.

## AJAXing the CalcFactors Example

The core algorithm of the example, the computation of the prime factors of a number, can also be implemented quite simply on the server. A url with the number in a parameter is requested from the server and the http answer will return the list of factors. The JavaScript method from CalcFactorsAsync1.htm can be converted quite easily into C# source code and can be called using a URL like this:

[http://www.mathertel.de/AjaxEngine/S01\\_AsyncSamples/CalcFactorsRequest.aspx?number=266712](http://www.mathertel.de/AjaxEngine/S01_AsyncSamples/CalcFactorsRequest.aspx?number=266712)

The source code of this server processing:

```
<%@ Page Language="C#" Debug="true" %>
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Xml.Xsl" %>

<script runat="server">

    private string CalcPrimeFactors(string inputText) {
        string outputText = String.Empty;
        UInt64 prime; // try this factor (only primes will match!)
        UInt64 number; // product of the remaining factors

        if ((inputText == null) || (inputText.Length == 0) || (inputText == "0"))
            return("no number given.");

        prime = 2; // start with 2
        number = UInt64.Parse(inputText);

        while ((number > 1) && (prime * prime <= number)) {
            if (number % prime != 0) {
                // try the next factor (slowly)
                prime += 1;
            } else {
                // found a factor !
                outputText = outputText + " " + prime;
                number = number / prime;
            } // if
        }
    }
}
```

```

    } // while

    if (number > 1) {
        // the last factor (a prime) is here.
        outputText = outputText + " " + number;
    }

    return(outputText.Trim());

} // CalcPrimeFactors

</script>
<%
    Response.Clear();
    string ret = CalcPrimeFactors(Request.QueryString["number"]);
    Response.Write(ret);
%>

```

That is the simple version of the function for the computation of the prime factors. The more complex variant of the code from CalcFactorsAsync2.htm is not needed. Thus programming is simplified for the server-side execution again and massive multithreading is part of http servers from the ground up.

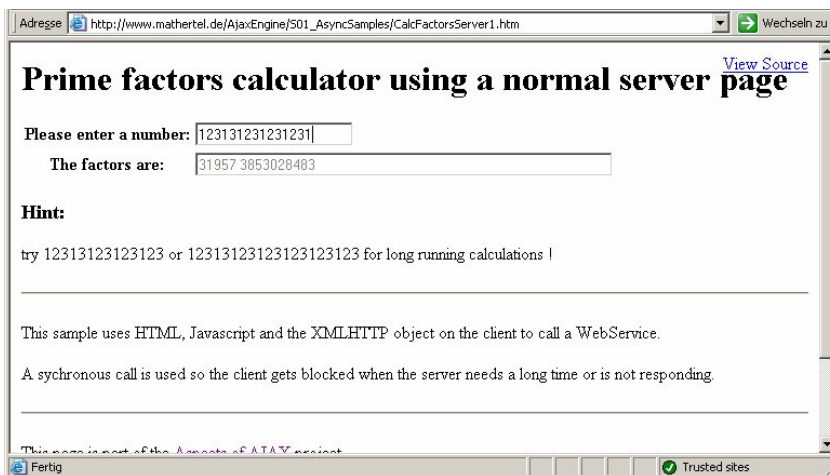
To start such a request from the client a quite simple code is sufficient:

```

var xmlObj = null;
if (window.XMLHttpRequest){
    // If IE7, Mozilla, Safari, etc: Use native object
    xmlObj = New XMLHttpRequest()
} else if (window.ActiveXObject) {
    // ...otherwise, use the ActiveX control for IE5.x and IE6
    xmlObj = new ActiveXObject("Microsoft.XMLHTTP");
}
xmlObj.open("GET", "http://localhost/CalcFactors/Request.aspx?number=" + inputText,
    false);
xmlObj.send();
outputText = xmlObj.responseText;

```

The first part retrieves one an XMLHttpRequest object through one of the various methods. The last 3 lines establish a connection to the server transfers the parameter and retrieves the result.



[http://www.mathertel.de/AjaxEngine/S01\\_AsyncSamples/CalcFactorsServer1.htm](http://www.mathertel.de/AjaxEngine/S01_AsyncSamples/CalcFactorsServer1.htm)

If you tried the samples using a client side calculation you may notice the faster response for calculation a computation. It's faster because the C# code on the server runs a lot better then any JavaScript code on the client. JavaScript is still interpreted and therefore slower. The response is faster in spite of using a network transfer for computing.

# Native AJAX Programming

## Doing it asynchronous

So we can call the server. The answer of this call can, as we know, take up some long (cpu) time and the send() method waits for the answer of the server. During this time and with the above implementation the client does freeze and user input events do not execute. Again we have a locking synchronous situation.

Sending the parameters and receiving of the answer therefore must be implemented in 2 different methods so that in the meantime the events of the keyboard and the mouse in the browser can execute.

The XMLHttpRequest object (here used through the xmlObj variable) must be defined in a global scope so it will be still available and not garbage collected at the time of the transmission of the result. The result will not be available immediately and a new function RetrievePrimeFactors is needed:

```
xmlObj.open("GET", "CalcFactorsRequest.aspx?number=" + inputText, true);
xmlObj.onreadystatechange = RetrievePrimeFactors;
xmlObj.send();
```

This function will be called when the result is present and the field for the result will be updated:

```
function RetrievePrimeFactors() {
    var outputText;

    if ((xmlObj != null) && (xmlObj.readyState == 4)) { //COMPLETED
        // The result is the whole body of the response
        outputText = xmlObj.responseText;
        xmlObj = null;
        document.getElementById("outputField").value = outputText;
    } // if
} // RetrievePrimeFactors
```

[http://www.mathertel.de/AjaxEngine/S01\\_AsyncSamples/CalcFactorsServer2.htm](http://www.mathertel.de/AjaxEngine/S01_AsyncSamples/CalcFactorsServer2.htm)

With this implementation we reach the kind of the Browser application that can be called an AJAX application. No real XML is used until now but in the end the same result is achieved.

Do we need XML?

Many web application calling themselves as AJAX applications do not use XML for any data transfer between the client and the server.

Some use the kind of request I use up to here that is often called a REST request. Another approach is to use a JSON (JavaScript Object Notation) syntax for the data to be transferred and both approaches can be combined.

An indirect AJAX approach is also very well known. This approach uses specific marked HTML regions inside a classic page and requests the server for updates on parts of the currently loaded page. This reduced the bytes on the network but still transfers design and data together.

My interest is in introducing the standard WebService mechanism and in a clear separation between html code and the data of the application.

# Client-Server Protocols

When searching the web for true AJAX samples (I don't call pure client side scripting an AJAX sample – but some people do) you may see that there are as many protocol definitions as there are samples and frameworks. Even if they mention to use JSON on the network layer the samples differ in many ways. I see no specific standard that is bound to AJAX web applications.

Seeing this situation I asked myself why not to use WebServices because I had a lot of experience with SOAP and already had implemented a SOAP client in JavaScript some years ago.

## Best Practices for a WebService implementation

Using a WebServices client is not so hard to do if you have a layer that takes care of building the valid XML for the network layer so the client should be implemented as you expect or know it from other platforms and languages by providing a stub function on the client that. You can use this local function from JavaScript to call the corresponding function on the server and pass the parameters. This is called a RPC network layer.

When using the (non-multitasking) JavaScript interpreter in the browser there is need for an asynchronous mechanism too. It's because waiting for an answer a call to a server can fail in many ways:

- Packets may get lost.
- The network may be unreliable
- The server may be overloaded
- ... and many more pitfalls.

It's even worse, because when calling a server from the browser using the http protocol there are some limitations you cannot get around:

- You can only call the server that is serving the page that is displayed when using the XMLHttpRequest object.
- Only 2 simultaneous calls can be used as specified in the http protocol definition.
- The timeout the network stack is using by default is too long so you need to cancel long running requests before the network stack does so if you want responsive applications.

Using the SOAP protocol has also very important advantage over using <script> tags or JSON objects together with the eval function because you can specify and check what data types you expect to send and receive using a specific function.

## SOAP was made for AJAX

I ran over the blog of Dave Winer and his post from Sun, Sep 12, 1999 titled "An end to the Über-Operating System".

<http://davenet.scripting.com/1999/09/12/anEndToTheUberoperatingSystem>

*... The purpose of both specs [SOAP and XML-RPC] is to enable scripted web applications to cross operating system boundaries.*

Of course the term AJAX was not known then but I think that "scripted web applications that cross operating system boundaries" is a good definition of AJAX. By using SOAP there even is more XML on the wire than with AJAX applications using a JSON or plain text transport protocol?

Ask yourself: what is or should be the standard protocol for AJAX applications when retrieving data from the server? - By using SOAP many aspects of a transport layer have been discussed (data types, attachments ...) and there are even more upcoming standard extensions for it (security, routing ...).

The only thing that I really miss is a built-in native SOAP client in browsers.

The (deprecated) SOAP spec from 1999:

<http://www.oasis-open.org/cover/draft-box-http-soap-00.txt>

I think that this doc is one of the most influencing documents I've read. It disappeared from the Microsoft site and <http://www.ietf.org>. Does anybody know who feels responsible for achieving historical documents from the internet?

Thanks to Don Box, Dave Winer and all the other founders and protagonists of SOAP.

XML, WebServices, SOAP and WSDL are still used as hype-words and some people still think that these protocols can solve all problems. I personally know they fit the situation where I use them in AJAX web applications.

## Using WebServices in AJAX applications

There are still some disadvantages with many available AJAX frameworks on the web and also with my preceding implementation:

- For every function you want to implement a special page must be realized.
- The URL may exceed a length of more than a few 100 chars and will bring up problems.
- The implementation on the server consists of code fragments used for the communication of parameters and the return value together with the application specific code.
- We need to implement a big part of the framework ourselves, for example parsing the parameters and converting them to the native types. This seems to be an easy job but there have been security leaks in (too simple) AJAX implementations.

Here it is obvious that using the server side communication infrastructure of WebServices brings in some huge advantages:

- The same functionality can also be used from other WebService enabled applications like Excel.
- Multiple methods can be implemented together in one class so it is possible to choose the right granularity for bigger collections of WebServices.
- You can use data types
- There is exception handling (more on this later)
- A proxy can be created for the client that enables an easy implementation of calling a server side method. (more on this soon)
- The existing SOAP and WSDL standards in the current available version (Spring 2005) are perfect usable on http and https connections. The circumstances for transferring html over http are identical and usable without any problems for web services.
- The actual discussions about extending the WebService Infrastructure with security features and routing soap messages are not needed and SOAP is as secure as the web is in this scenario.
- The core implementation of the WebService infrastructure can be reused. This part should be highly stable and security issues (buffer overflows etc.) should be cleared by the big manufacturers like Microsoft, SUN, IBM, BEA, ... and the AJAX Engine just gets the benefits from this.

WebServices can be implemented very easily in ASP.NET so I prefer this platform in my samples but also port the core part if it to the JAVA platform to see how platform independent the Engine is.

The implementation of a WebService for the calculation of the prime factors can be found in CalcService.asmx in this sample website. The core implementation is the same as in Request.aspx.

The implementation of the AJAX engine is done in JavaScript only and has to take care of the different browser platforms but is independent of the server platform because the protocol is standardized.

```
<%@ WebService Language="C#" Class="Service" %>

using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://www.mathertel.de/CalcFactorsService",
    Description="A WebService for the calculation of prime factors.")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService {
```

```

...

[WebMethod(Description="Calculate all prime factors of a given number.")]
public string CalcPrimeFactors(string inputText) {
    string outputText = String.Empty;
    UInt64 prime; // try this factor (only primes will match!)
    UInt64 number; // product of the remaining factors

    if ((inputText == null) || (inputText.Length == 0) || (inputText == "0"))
        return (null);

    prime = 2; // start with 2
    number = UInt64.Parse(inputText);

    while ((number > 1) && (prime * prime <= number)) {
        if (number % prime != 0) {
            // try the next factor (slowly)
            prime += (prime == 2UL ? 1UL : 2UL);
        } else {
            // found a factor !
            outputText = outputText + " " + prime;
            number = number / prime;
        } // if
    } // while

    if (number > 1) {
        // the last factor (a prime) is here.
        outputText = outputText + " " + number;
    }

    return (outputText);
} // CalcPrimeFactors

...
} // class

```

[http://www.mathertel.de/AjaxEngine/S01\\_AsyncSamples/CalcService.asmx](http://www.mathertel.de/AjaxEngine/S01_AsyncSamples/CalcService.asmx).

The special thing here, compared to the previous sample using Request.aspx with an http GET command, is that the functional characteristics of the server side operation gets clear. There is no need for the programmer to care about the details of the communication

Because there no usable SOAP client available in the current browsers there is more overhead in the JavaScript programming in the browser for implementing the SOAP protocol as far as we need it. There is in deed a SOAP interface available in Firefox but this implementation is not working correctly and is only partial implemented.

Fortunately the assembling of the SOAP request and analyzing the result is also possible by building strings that contain the valid XML code and transferring it to the server with the well known XMLHttpRequest Object.

Here is some sample code that directly calls a Webservice:

You should use the JavaScript proxies instead of hand-coding calls to the server every time. It's far more robust and easier to use.

```

// call the server using the SOAP encoding
xmlObj = new XMLHttpRequest("Microsoft.XMLHTTP");
xmlObj.Open("POST", "http://localhost/CalcFactors/Service.asmx", true);
xmlObj.setRequestHeader("SOAPAction",
"http://www.mathertel.de/CalcFactorsService/CalcPrimeFactors");
xmlObj.setRequestHeader("Content-Type", "text/xml; charset=utf-8");
xmlObj.onreadystatechange = RetrievePrimeFactors;
var soapText = "<?xml version='1.0' encoding='utf-8'?>"
+ "<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>"
+ "<soap:Body>"
+ "<CalcPrimeFactors xmlns='http://www.mathertel.de/CalcFactorsService'>"

```

```
+ "<inputText>" + inputText + "</inputText>"
+ "</CalcPrimeFactors>"
+ "</soap:Body>"
+ "</soap:Envelope>";
xmlObj.Send(soapText);

...

// the response is inside the <CalcPrimeFactorsResult> tag
outputText = xmlObj.ResponseText;
p = outputText.indexOf("<CalcPrimeFactorsResult>");
if (p > 0) {
    outputText = outputText.substr(p+24);
    outputText = outputText.substr(0, outputText.indexOf("<"));
} // if
```

This implementation is only used to show how the SOAP protocol works but has no error handling and lacks of being robust.

This implementation can be found in the page CalcFactorsServerSoap.htm on the sample website. It only works for IE.

[http://www.mathertel.de/AjaxEngine/S01\\_AsyncSamples/CalcFactorsServerSoap.htm](http://www.mathertel.de/AjaxEngine/S01_AsyncSamples/CalcFactorsServerSoap.htm)

The server side code gets dramatically simplified when using WebServices. For the client there also exists a good solution and another level of abstraction that makes implementing AJAX calls really easy.

BUT

Implementing AJAX can even be simpler than in this sample. We don't have to care about SOAP and directly using the XMLHttpRequest object. Have a look at the AJAX Engine.



## A SOAP client for JavaScript

Calling a server from JavaScript is a fundamental part of AJAX applications. Using WebServices with SOAP and WSDL is easy if proxy objects and methods are available in the browser.

### Introduction

From the languages and programming environments like C, the .NET CLR and Java we are know proxy generation mechanisms based on IDL and RPC for a long time. These generated classes and files enable the programmer to call a server-side method by calling a local method with the same name. The implementation of the network transfer is taken off your application code.

If you want to implement a communication from JavaScript to WebServices using SOAP it is very important to use an approach that needs only a small amount of code. Complex and long scripts tend to be buggy.

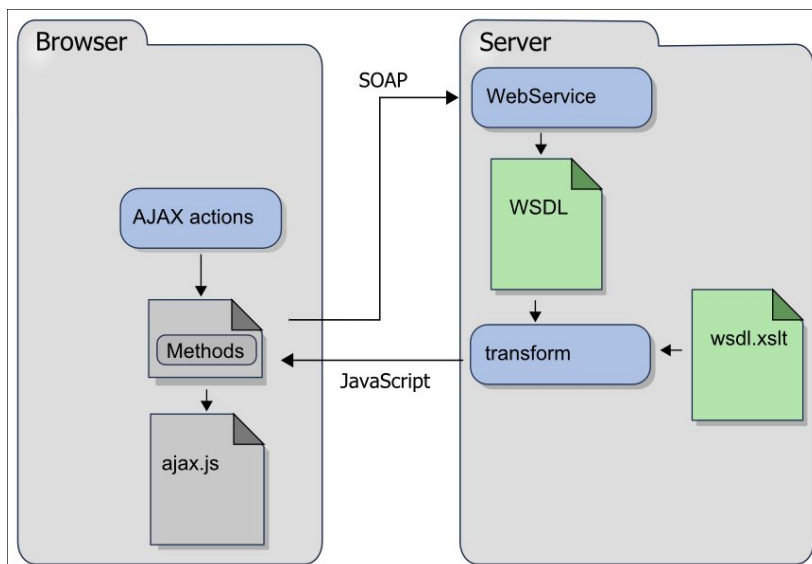
This Proxy generator can be used on its own but is also part of the AJAXEngine framework.

Some AJAX implementations use their own way to transport the information between the client and the server. This implementation uses the standard SOAP protocol and works on Internet Explorer and the Firefox browser.

### How it works - in short

WebServices can be described by using the formal description standard for WebServices called WSDL (WebService Description Language). Everything we need to know for calling a WebService is available in this XML formatted information.

Based on this service description it is possible to generate JavaScript source code that contains methods and some descriptive information. Because WSDL is formatted in XML the transformation can be done by using the XSLT technology.



The transformation itself is really simple. The real implementation is inside the wsdl.xslt file.

### Using the proxy

To make these proxy functions work a common JavaScript include (ajax.js) file and a file that generates the WebService specific code must be included.

```
<script type="text/javascript" src="ajax.js"></script>
<script type="text/javascript" src="getjavascriptproxy.aspx?
service=../S02_AJAXCoreSamples/CalcService.asmx"></script>
```

The implementation of the real communication details are implemented in the ajax.js file. A variable named "proxies" is created as an empty JavaScript Object and this is the only one global variable that we need. The individual proxies are then attached to this Object to minimize the naming conflicts that may occur.

The second script include now retrieves the WSDL description of the WebService and generates the specific JavaScript for this service containing local proxy methods that just can be called to execute the corresponding method on the server.

### Asynchronous calls

Calling a server-side method may look like this:

```
// hook up a method that gets the response
proxies.CalcService.CalcPrimeFactors.func = displayFactors;

// now call the server
proxies.CalcService.CalcPrimeFactors(12);

// The return value is passed to this function as a parameter
function displayFactors (retVal) {
    document.getElementById("outputField").value = retVal;
} // displayFactors
```

Here you see an asynchronous call. The function CalcPrimeFactors() returns immediately and the client side scripting continues. After some milliseconds (or longer) the server will send back the result of the called method of the WebService and the value will be passed to the hooked up method as a parameter.

### Synchronous calls

There is also a synchronous version that can be used. In this case the func attribute must remain unset or null and the result of the server-side method is directly returned from the client side method call. This kind of calling the server may block for some milliseconds because no user-events like typing or clicking are processed during the call.

```
proxies.CalcService.func = null; // no hook up function !

// call the server and return the result.
var f = proxies.CalcService.CalcPrimeFactors(12);
```

### Implementation details

Here is a sample extract of the code that is generated for the client to show how the mechanism works.

The include file ajax.js generates the global object named "ajax":

```
var proxies = new Object();
```

Per WebService an object named like the WebService is attached to the ajax object to hold the service specific information like the url and the namespace of the WebService:

```
// JavaScript proxy for webservices
```

```
// A WebService for the calculation of prime factors.
proxies.CalcService = {
  url: "http://localhost:1049/CalcFactors/CalcService.asmx",
  ns: "http://www.mathertel.de/CalcFactorsService/"
} // proxies.CalcService
```

For each WebService method a function on the client is created that mirrors the method on the server. The information we need to build up the full SOAP message is attached to the function object as attributes.

```
// Add 2 numbers.
proxies.CalcService.AddInteger = function () {
  return(proxies.callSoap(arguments)); }
proxies.CalcService.AddInteger.fname = "AddInteger";
proxies.CalcService.AddInteger.service = proxies.CalcService;
proxies.CalcService.AddInteger.action = "http://www.mathertel.de/CalcFactors/AddInteger";
proxies.CalcService.AddInteger.params = ["number1:int", "number2:int"];
proxies.CalcService.AddInteger.rtype = ["AddIntegerResult:int"];
```

### Simple Caching

The proxy implementation also offers a client-side caching feature. An approach that leads to less traffic on the net because repeating the same calls can be prevented.

The Http caching features, instrumented by using HTTP headers do not help in these situations because the request is not an http-get request and there is always a payload in the http body. Caching must therefore be realized by some scripting on the client.

The caching feature in the JavaScript WebService proxy implementation can be enabled by calling the method `proxies.EnableCache` and passing the function that should further use caching. There is a button in the `CalcFactorsAJAX.htm` sample to show how to enable this:

```
proxies.EnableCache(proxies.CalcService.CalcPrimeFactors)
```

By calling this method a JavaScript object is added that stores all results and is used to prevent a call to the server if an entry for the parameter already exists inside this object. This is not a perfect solution, but it works under the following circumstances:

- The parameter must be a string or number that can be used for indexing the properties of a JavaScript object.
- The cache doesn't clear itself. It can be cleared by calling `EnableCache` once again.
- Only methods with a single parameter are supported.

### Analysing problems

With `proxies.service.function.corefunc` is an entry point in the core implementation of the proxies object available that may be helpful when analyzing problems.

To set up a debug output in an alert box that displays the response of a WebService call use:

```
proxies.CalcService.CalcPrimeFactors.corefunc = proxies.alertResult;
```

If the full response text of the received SOAP message is needed you can use:

```
proxies.CalcService.CalcPrimeFactors.corefunc = proxies.alertResponseText;
```

Instead of attaching a special function for the further processing of the response of a WebService Call it is possible to just hook up the `window.alert` function to display the result.

```
proxies.CalcService.CalcPrimeFactors.func = window.alert;
```

I recommend always attaching a function for handling possible exceptions, at least while developing. When calling a Webservice asynchronously there will be no executing code that can catch an exception so you must provide a method in this case.

```
proxies.CalcService.CalcPrimeFactors.onException = proxies.alertException;
```

## Good tools for analyzing network problems

If there are still problems with the communication of the SOAP messages I recommend using an http monitor tool like Fiddler:

<http://www.fiddlertool.com>

If you prefer to implement by using the Firefox browser I recommend the debugger named “firebug” that has a good network tracing functionality built in. There you can see most of the communication details and it has a fantastic feature for analyzing the timing situations when the page gets loaded. You can install it from

<http://www.getfirebug.com/>

## Generating JavaScript Proxies in ASP.NET

Retrieving a WSDL description is very easy when implementing in ASP.NET. The URL of the WebService can be used with an attached WSDL Parameter:

[http://www.mathertel.de/AJAXEngine/S02\\_AJAXCoreSamples/CalcService.asmx?WSDL](http://www.mathertel.de/AJAXEngine/S02_AJAXCoreSamples/CalcService.asmx?WSDL)

The proxy generator can retrieve this XML document by using an `HttpRequest`. By using a XSLT transformation it is now very simple way to implementing a WSDL to JavaScript compiler.

```
// GetJavaScriptProxy.aspx

private string CreateClientProxies (string url) {
    if ((url != null) && (url.StartsWith("~/")))
        url = Request.ApplicationPath + url.Substring(1);
    if (url.EndsWith(".asmx", StringComparison.InvariantCultureIgnoreCase))
        url = url + "?WSDL";
    Uri uri = new Uri(Request.Url, url);

    HttpRequest req = (HttpRequest)WebRequest.Create(uri);
    req.Credentials = CredentialCache.DefaultCredentials;
    // req.Proxy = WebRequest.DefaultWebProxy; // running on the same server !
    req.Timeout = 6 * 1000; // 6 seconds

    WebResponse res = req.GetResponse();
    XmlReader data = XmlReader.Create(res.GetResponseStream());

    XslCompiledTransform xsl = new XslCompiledTransform();
    xsl.Load(Server.MapPath("~/ajaxcore/wsdsl.xslt"));

    System.IO.StringWriter sOut = new System.IO.StringWriter();
    xsl.Transform(data, null, sOut);
    return (sOut.ToString());
} // CreateClientProxies
```

The complex part lies in writing the right transformations. Inside the `wsdsl.xslt` file you can find the templates of the JavaScript code that define these proxy objects. Instead of generating another XML document this transformation produces plain text that is valid JavaScript code.

You can find the source of the `wsdsl.xslt` file in the listings at the end of this book or online.

The complete `GetJavaScriptProxy.aspx` source code

<http://www.mathertel.de/AjaxEngine/ViewSrc.aspx?file=ajaxcore/GetJavaScriptProxy.aspx>

`wsdsl.xslt` source code

<http://www.mathertel.de/AjaxEngine/ViewSrc.aspx?file=ajaxcore/wsdsl.xslt>

JavaScript Proxy sample code

You can see the actual generated JavaScript proxy by loading the url:

[http://www.mathertel.de/AjaxEngine/ajaxcore/GetJavaScriptProxy.aspx?service=../S02\\_AJAXCoreSamples/CalcService.asmx&html=true](http://www.mathertel.de/AjaxEngine/ajaxcore/GetJavaScriptProxy.aspx?service=../S02_AJAXCoreSamples/CalcService.asmx&html=true)

## Generating JavaScript Proxies in JAVA

This AJAX Engine was originating implemented for the ASP.NET 2.0 Platform. When porting the code to Java the only real re-implementation needs to be done in GetJavaScriptProxy:

Here is the core part of the code:

```
String url = request.getScheme() + "://" + request.getServerName()
    + ":" + request.getServerPort() + request.getContextPath();

String service = request.getParameter("service");
if ((service != null) && (!service.startsWith("/")))
    url += "/";
url += service;

ServletContext ctx = pageContext.getServletContext();

String xsdlFile = ctx.getRealPath("ajaxcore/wsdl.xslt");

TransformerFactory tFactory = TransformerFactory.newInstance();

Transformer transformer = tFactory.newTransformer(new StreamSource(xsdlFile));
transformer.transform(new StreamSource(url), new StreamResult(out));
```

You can see that most of the work is done inside the XSLT transformation using wsdl.xslt and wsdl.xslt can be used without changes.

## Proxy Generator supported datatypes

### Simple datatypes

Up to now only those methods were supported that where converting of the parameters and result values was not necessary. This applies to strings and numbers.

With this version the datatypes defined on the server and the WSDL are passed to the client so that the datatypes can be converted using JavaScript at runtime. In the generated proxy code, the listing of the names of the parameters is now extended by an optional specification of the datatype. Without this the values are treated as strings.

In the HTML object model, the JavaScript datatypes are not well supported. The value that is displayed inside an HTML input field is always a string, even if it's containing only digits. So when calling the proxy functions all the parameters are also accepted as JavaScript strings and converted (if possible) to the right types.

### XML data

Passing XML documents was implemented to make it possible to pass complex data. In the supported browser clients the XmlDocument Object from Microsoft or Firefox and on the server the .NET XmlDocument class can be used.

A method has to be is declared in C# like this:

```
[WebMethod()]
public XmlDocument Calc(XmlDocument xDoc) {
    ...
    return (xDoc);
} // Calc
```

The proxy functions also accept the XML document as a string type. In this case, the contents of the passed string is passed directly to the server any must for this reason contain a valid XML document without the declarations any without any "XML processing Instructions" like <? ... ?>.

With this datatype it is possible to pass complex data directly to the server an there is no need to define a method with many parameters if using this datatype. If the data scheme is extended with new fields it will not be necessary to give a new signature to the WebService.

The disadvantage of this approach is that the content of the XML document cannot be validated by the WebService infrastructure because there is no schema for this part of the conversation available.

### The implementation of the call

The transmission of the SOAP/XML Messages can be implemented using the appropriate XMLHttpRequest object that is available in many state-of-the-art browsers today. This implementation was (until now) tested with Internet Explorer and Firefox.

```
/// <summary>Get a browser specific implementation of the XMLHttpRequest object.</summary>
function getXMLHTTP() {
    var obj = null;

    // from http://blogs.msdn.com/ie/archive/2006/01/23/516393.aspx
    if (window.XMLHttpRequest) {
        // if IE7, Mozilla, Safari, etc: Use native object
        obj = new XMLHttpRequest();

    } else if (window.ActiveXObject) {
```

```
// ...otherwise, use the ActiveX control for IE5.x and IE6
try { objx = new ActiveXObject("Msxml2.XMLHTTP"); } catch (e) { }
if (objx == null)
    try { objx = new ActiveXObject("Microsoft.XMLHTTP"); } catch (e) { }
} // if

return(obj);
} // getXMLHTTP
```

<http://blogs.msdn.com/ie/archive/2006/01/23/516393.aspx>

This object is implemented in different technologies, depending on the available technologies in the browsers. It was first developed by Microsoft in the Internet Explorer as an ActiveX control and the Mozilla developers re-implemented it by providing the same methods and properties. A call can be done using the following sequence of methods:

```
x.open("POST", p.service.url, true); // async call
x.setRequestHeader("SOAPAction", p.action);
x.setRequestHeader("Content-Type", "text/xml; charset=utf-8");
x.onreadystatechange = p.corefunc; // hook up a method for the result processing
x.send(soap); // send a soap request
```

More details and some more internal description can be found in `ajax.js` include file.

#### Datatype mappings

| XML datatypes                                                | Alias in the proxy attributes | JavaScript Datatype                                                                                                                       |
|--------------------------------------------------------------|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| string                                                       | string / null                 | String                                                                                                                                    |
| int, unsignedInt, short, unsignedShort, unsignedLong, s:long | int                           | Number (parseInt)                                                                                                                         |
| double, float                                                | float                         | Number (parseFloat)                                                                                                                       |
| dateTime                                                     | date                          | Date                                                                                                                                      |
| boolean                                                      | bool                          | Boolean                                                                                                                                   |
| System.Xml.XmlDocument                                       | x                             | In Mozilla / Firefox:<br>XMLDocument<br>In Internet Explorer:<br>ActiveXObject("Microsoft.XMLDOM")<br>ActiveXObject("MSXML2.DOMDocument") |



# The AJAX Engine

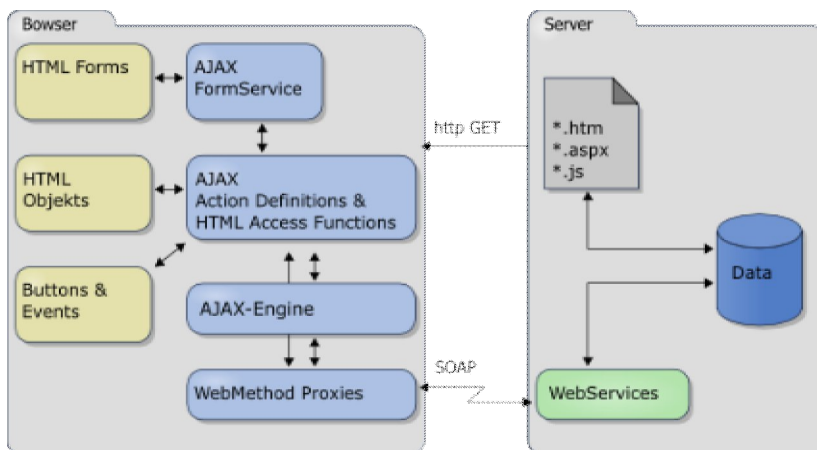
## Overview

The basic elements for an application using the AJAX technologies JavaScript and XMLHttpRequest are not difficult to realize. However, the selection from the available technologies as well as a suitable abstraction in using these elements is important for the success of the realization of an application. One main goal behind the architecture of this AJAX engine was to build an AJAX framework that you can reuse every time you want some asynchronous processing or when you need a smart way to refresh information on the current web page will help a lot.

When targeting applications with some hundred sides and WebServices and with in sum about a thousand methods the developer must have a clear and simple kind and pattern for coding the JavaScript code on the client to avoid errors and to not think about the implementation details. Only by using a simple approach a good quality and maintenance can be achieved.

So the main idea of this AJAX engine on the client is the simplification of the implementation of the code, which is needed for a specific function on the client. Like with the WebService framework of ASP.NET on the server the details of communication over SOAP on the client are completely hidden from the developer and also the recurring code portion is only realized once in a central place. Also all details about the different implementations of the XMLHttpRequest object in Internet Explorer or the Firefox browsers are hidden from your code.

The blue print of the AJAX engine has the following connected components:



### HTML form and elements

The static or dynamically provided HTML of objects e.g. `<input>` elements become for the interaction with the user used.

### Buttons & Events

The buttons and events that start the AJAX functionality must only call a simple JavaScript function. This can be used for example using inline code in an onclick attribute. This starts then the processing of the steps that should be executed.

### AJAX actions

For the call of the server-side functionality parameters must be retrieved and the result must be processed on the Client. These elements together form a AJAX action.

### AJAX functions

AJAX functions are the basic steps that together form a AJAX functionality which is supported by a slim and efficient JavaScript framework that hides the browser specific details from the implementation.

### Web methods and proxies

The proxy framework is used on the client to call the server-side methods. The individual methods to call a WebService is generated by a WSDL to JavaScript Compiler and the core functionality for building SOAP messages is available in `ajax.js`.

### The AJAX engine

The execution of the individual steps of an AJAX functionality using timer and XMLHttpRequest objects is coordinated and supervised by the AJAX engine. The methods for the implementation of the AJAX engines are available in `ajax.js`.

Also the AJAX engine needs only one variable with the name `ajax` to be declared in the global namespace to minimize name conflicts.

In many cases it is importantly for the implementation of actions that they occur one after the other by using a queue mechanism. This is also necessary from a network perspective because with HTTP connections only 2 simultaneous requests per server should be made at a time.

Therefore the AJAX engine implements a queue. Using some attributes on the AJAX actions it is possible to control the way how the existing entries are treated when starting of a new action.

### WebServices

The communication between the client and the server is realized by using the standard WebServices infrastructure with SOAP messages and WSDL services descriptions instead of building a new server-side mechanism for good reasons.

- Writing a server side framework must not be done. There are already a lot of them but the WebService based on SOAP and WSDL is widely accepted as a standard.
- Before implementing a new proprietary core feature with a high complexity I think it makes sense to search for existing technology in the common frameworks that I can rely on.
- I use ASP.NET in my engine to build the server side part. Because I expose only WebServices to the client-side AJAX engine it should be possible to port this part of the overall application architecture to another server platform if needed.
- Writing server-side ports can be a nightmare regarding security. Many security risks of the past years came through open ports that do not work as expected and could be used for different purpose.
- By just NOT implementing a server side communication framework I leave this task to Microsoft. – Of course it is still necessary to write good and secure code inside the methods.
- I haven't found a situation until now where WebServices do not fit into the architecture.

|                                                                                                                                                                                                                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>You cannot avoid using some global variables when implementing JavaScript in the browser but using as few as possible and structuring the storage by using more complex objects helps getting compatible.</p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

- The technologies around SOAP and WSDL are at a solid usable level but also still evolving. I expect to see a native universal service client in browser type applications and I hope we will not have to use the basic XMLHttpRequest Object in the future and will participate in this evolving when building web applications.  
(Mozilla/Firefox has already a built-in but very buggy object to do this and Microsoft has as COM object part of the Office suite).

The pages and include files

The initial load of a web application using AJAX is a regular HTML GET operation. In the samples you see plain \*.htm or \*.aspx files that have a constant html output.

ASP.NET Web Forms and ASP.NET Web Controls are not used to reflect any POSTed information or the state of the client. Also no session variables are use.

The common JavaScript functions, the JavaScript proxy code we see next and the control specific functions we use later are downloaded by fetching static JavaScript include files or dynamically generated static JavaScript code.

Because the initial content is static the server and the browser can be enabled to cache these files and give the user a fast response.

## AJAX Actions

Using the web service proxy functions directly from JavaScript code is one option for building AJAX enabled web pages. With this approach you have all the flexibility and power with data types and multiple parameters that you might need.

On the other side you will have to write a lot of JavaScript code for each scenario that you have to implement. AJAX actions will bring you another layer of abstraction that reduces a lot of the complexity of AJAX by reducing the amount of code.

Using the AJAX Engine is done by defining AJAX Actions that are used on the loaded page. Here is a simple sample:

```
// declare an AJAX action
var action1 = {
  delay: 200,

  prepare: function() {
    return (document.getElementById("inputField").value);
  }, // prepare

  call: proxies.CalcService.CalcPrimeFactors,

  finish: function (p) {
    document.getElementById("outputField").value = p;
  }, // finish

  onException: alertException
} // action1
```

You can see that the logical steps of an action are written in the source code step by step as they will be executed even if there are timers and callback methods used to implement it behind the scene. This makes it easy to follow the idea of each action. It is possible to write inline JavaScript code as well as linking to existing available functions.

Declaring this object is easy when using the JSON syntax extended with some functions for describing complex JavaScript objects.

JSON stays for JavaScript Object Notation and can be used to build complex objects at runtime. It's not a class definition mechanism!

## AJAX Action Reference

Every action on the page is described with the assistance of an object that holds all information together. The properties of this object are:

`action.prepare(option)`

This property defines a function that is used directly before sending the SOAP of package to the server. This function has to return the data that is used as the argument of the call to the server. If this function is not defined then a server-side method without parameters is called.

The option that was passed to the `ajax.Start` function is also available in the prepare function as a parameter.

`action.call()`

This property refers to a function that was generated by the proxy generator to a `WebService`.

`action.finish(option)`

This property defines a function that will retrieve the result of the call. This property may be zero.

The option that was passed to the `ajax.Start` function is also available in the `prepare` function as a parameter.

`action.onException(ex, option)`

This property defines a function that is called in the case of an exception. This property may be zero.

The option that was passed to the `ajax.Start` function is also available in the `prepare` function as a parameter.

`action.delay : [msec]`

This property defines the time in milliseconds how long the call should be delayed. This is useful together with events that are triggered very often and in a fast order. The default value of 0 deactivates this functionality and the action starts immediately.

`action.timeout : [sec]`

This property defines the time in seconds how long the call to the server can last before it is canceled on the client. When this time runs off without an answer from the server the http connection is closed. The default value of 0 deactivates this functionality.

A timeout of more than approx. 60 seconds does not work well because http requests can be terminated by the network layer before this time.

With some additional flags you can specify how pending actions are handled.

`action.queueClear : [bool]`

This property can be set to true to specify that the queue with the pending actions is cleared before the new action is entered. An already running action will not be stopped. The default value is false.

`action.queueTop : [bool]`

This property can be set to true to specify that the action is entered into the queue in front of all other pending actions. The default value is false.

`action.queueMultiple : [bool]`

This property can be set to true to specify that an action can be queue more than once. The default value is false.

In the samples you can see that it is not necessary to write functions before assembling them into the action object. The implementation of a function is also possible using inline code.

**Important Note:** The last property of a definition of a JavaScript object MUST NOT have a trailing comma. The otherwise quite tolerant Microsoft Internet Explorer complains of this error during the Firefox tolerates this. I lost some hours not seeing a comma like this.

## Starting an AJAX Action

`ajax.Start(action, option)`

By calling the `Start` function of the global `ajax` object a defined action will be started.

With the option parameter it is possible to define an option for a specific execution that will be passed to various functions that are defined through the action definition. This value is stored into the queue of the AJAX engine together with the first parameter specifying the action.

When the prepare and finish methods are executed as well on handling an exception, this value is passed as an additional parameter. By using this value it is possible to store a context of an action. That may be for example a HTML object that started the action. Because all methods have access to it, it is now possible to use the same action definition for multiple fields, forms or other situations. Without this parameter it was necessary to implement the direct access to the HTML objects inside the prepare and finish methods.

```
ajax.Cancel()
```

This method cancels the execution of the currently running action. If a timeout value is specified in the action options this method is called automatically.

```
ajax.CancelAll()
```

This method cancels the execution of the currently running and all the pending actions.

## Handling Exceptions

Also in AJAX architectures failures in the communication and in the execution may raise errors. Fortunately when using a WebServices based framework these cases are also defined in the SOAP communication protocol.

The answer to a call to a Webservice in the SOAP format will under normal conditions return the result of the method:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <CalcPrimeFactorsResponse xmlns="http://www.mathertel.de/CalcFactorsService/">
      <CalcPrimeFactorsResult>2 2</CalcPrimeFactorsResult>
    </CalcPrimeFactorsResponse>
  </soap:Body>
</soap:Envelope>
```

In the case of an error the information about the failure of the server-side execution will be passed back to the client.

In the simplest case this information is shown to the user but also other reactions may be appropriate like reloading the whole page.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>Server was unable to process request. ---&gt;
        Input string was not in a correct format.</faultstring>
      <detail />
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

The client-side implementation allows the handling of the exceptions that occur at the level of the Webservice proxies and on the level of the AJAX Actions.

In both cases there is an `onException` event available and a specialized method handling these exceptions can be plugged in. This method gets passed the thrown exception as a parameter and can organize the further functionality of the page.

There is usable method defined in `ajax.js` that can be used for showing exceptions:

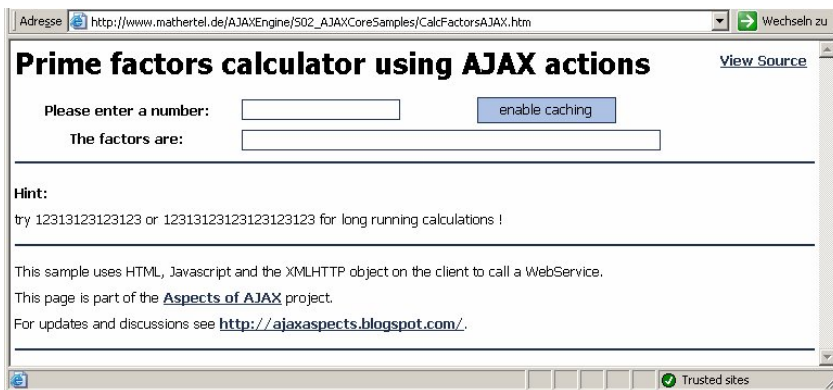
**`proxies.alertException(ex):`**

This method shows the exception object in a readable format using an alert box. This method can be used in `proxies.service.method.onException` as well as in `action.onException`.

## Examples that use the AJAX Engine directly

### The AJAX prime factors sample

Here is the prime factors calculation sample, but now using the AJAXEngine. You will see that the amount of scripting on the client is again reduced to effectively 2 lines of coding for the prepare and finish methods and some lines of parameters and the include for the Webservice proxy.



[http://www.mathertel.de/AJAXEngine/S02\\_AJAXCoreSamples/CalcFactorsAJAX.htm](http://www.mathertel.de/AJAXEngine/S02_AJAXCoreSamples/CalcFactorsAJAX.htm)

The first sample that shows how to use the AJAX engine can be found in the files [CalcFactorsAJAX.htm](#) (Client) and [CalcService.asmx](#) (Server). With this sample the principle steps can be analyzed easily.

The connection to the already known Webservice is done by using the proxy generator:

```
<script type="text/javascript" src="GetJavaScriptProxy.aspx?service=CalcService.asmx"></script>
```

The HTML code contains 2 input elements that are used for the communication with the user.

The import element with the id inputField is editable and used to enter a number that should be split into the prime factors. The event onkeyup was chosen for triggering the AJAX action.

This event is not only triggered by entering characters but by the special keys like backspace, delete or Ctrl+C for pasting a new value.

With IE, there is also the event onpropertychange available that suits better our needs here but the Firefox browser doesn't implement this and there is no standard event defined for being triggered immediately on any value changes.

The field with id outputField is deactivated and is used to show the calculated prime factors.

```
...
<td><input id="inputField" onkeyup="ajax.Start(action1)"></td>
...
<td><input id="outputField" size="60" disabled="disabled"></td>
...
```

The AJAX action is declared inside the JavaScript block and there is no other programming necessary except this.



You can see within the methods prepare and finish how to exchange the values with HTML Elements.

The call method is setup by linking to the local method of the generated proxy and exceptions are displayed by using a simple alert box.

The delay parameter is set to 200 msec. This time was chosen after some empirical timing measurements (using myself) and was found being appropriate. When one of the testing people entered a number this time wasn't exceeded between entering the digits and the action was indeed executed at the end of entering the whole number.

If the action was started already by entering a digit and when another digit was entered before the delay time was over then the first action was removed from the queue, the second identical action was entered into the queue and the timer was started again.

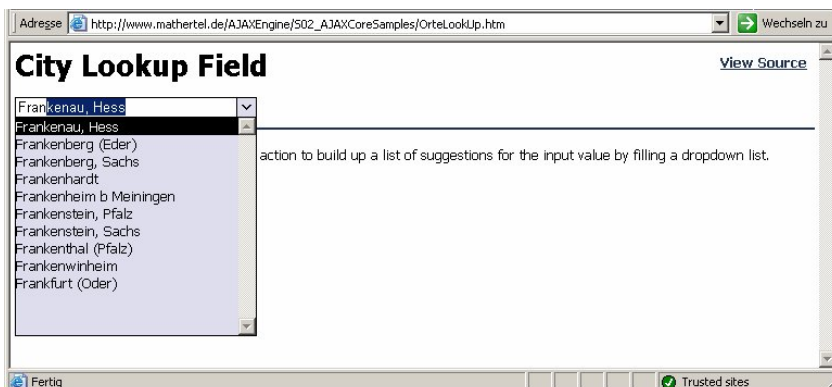
```
// declare an AJAX action
var action1 = {
  delay: 200,
  prepare: function() { return (document.getElementById("inputField").value); },
  call: proxies.CalcService.CalcPrimeFactors,
  finish: function (p) { document.getElementById("outputField").value = p; },
  onException: proxies.alertException
} // action1
```

Looking at this script you can imagine the actual flow of the action. This is a big advantage for everyone that needs to analyze, extend or fix a broken implementation, because all the functionality is not spread over the source code into several methods, callback functions and timer events but kept together in this JavaScript object.

## A auto-completion textbox and lookup for city names

This is another sample using the AJAX Engine to lookup names of cities from a huge server side list and proposing them for completion. You can see the AJAX related sources here.

The HTML objects and the JavaScript code we need to display the popping up field and handling the various keyboard events is lengthier and you can find it in the sources if you like to analyze it. Later we will build a ASP.NET web control that makes reuse of this code quite easier.



[http://www.mathertel.de/AJAXEngine/S02\\_AJAXCoreSamples/OrteLookUp.htm](http://www.mathertel.de/AJAXEngine/S02_AJAXCoreSamples/OrteLookUp.htm)

In this implementation you can see how to load and display data fragments from a huge dataset on the server by using an AJAX action.

On the server, there is a huge list of German towns and cities in the file orte.txt. If you would include this information into the page as a list of

OPTION elements for a SELECT element you would have to download about 400.000 bytes of additional HTML code – too much. Instead a simple INPUT field is combined with the possibility to search in this dataset.

The WebService that implements this lookup functionality can be found in OrteLookup.asmx and, again, has no AJAX specific implementations. The method gets one parameter passed that contains the first characters of a possible name of a city and returns up to 12 found entries of the list back to the client.

The connection to this WebService is done by using the proxy generator:

```
<script type="text/javascript" src="GetJavaScriptProxy.aspx?service=OrteLookup.asmx"></script>
```

This lookup method must be called in several situations: The code on the client that display the list and the completion of the current entered text is somewhat complex and most of the JavaScript of this page is about that. The call `ajax.Start(LookupAction)` that can be found several times starts the action.

The action itself is implemented very compact in one place and you can see that a not a lot of coding is necessary because the AJAX engine handles a lot.

```
var LookupAction = {
  delay: 100,
  prepare: function() { return (document.getElementById("inputField").value); },
  call: proxies.OrteLookup.OrteStartWith,
  finish: function (val) {
    var fld = document.getElementById("inputField");
    var dd = createDropdown(fld);
    FillDropdown(dd, val);
    if (isIE)
      NextDropdownEntry(fld);
  },
  onException: proxies.alertException
} // LookupAction
```

Looking at this script you can again see the actual flow of the action and there is no need to implement several JavaScript methods outside of the definition.

## An AJAX Engine for Java

This AJAX Engine was originating published in 2005 for the ASP.NET 2.0 Platform and is now partly also available in JAVA. This engine can be ported very easily because it relies on web standards that are available on many platforms and in many languages. For porting the AJAX framework to the Java platform the Proxy generator that is executed on the server must be re-implemented. This might look complicated but it isn't because the most part of the generation is done by using an XSLT transformation that can be reused without any changes.

### JavaScript & ajax.js

A huge part of the coding is done using JavaScript on the client. This code needs shared by both projects and is always exchangeable.

### WebServices

The asynchronous calls from the page in the browser to the server are implemented using the SOAP standard. On the server we only need Standard WebServices. Both platforms, ASP.NET and Java support the bottom up implementation of WebServices and we just need to write simple methods that get called from the client.

### XSLT / WebService Proxies

To enable a communication from the client side, the server uses the WSDL definitions that can be generated from WebServices on both platforms and transforms them to proxy objects using JavaScript.

The retrieval of the WSDL and for the transformation is ported and needs only about 30 lines of code that you can find in  
~/ajaxcore/GetJavaScriptProxy.aspx or ~/ajaxcore/GetJavaScriptProxy.jsp.

The definition of the translation itself is coded using an XSLT definition in the file ~/ajaxcore/wsd.xslt that is also identical on both platforms.

The core AJAX Engine needs nothing else than these 3 building blocks.

### Download

The download is available on my web side:  
<http://www.mathertel.de/AJAXEngine/> in the download section containing an eclipse project. I use right now: Eclipse 3.1.2 with the Webtools plug-in 1.0, Java 1.4.2.x, Tomcat 5.5 incl. the compatibility jar and Axis 1.2.1

## AJAX and Forms

The good old HTML form element is used by many web pages for the communication of the data from forms. This mechanism was developed to work without any JavaScript or XML and long before AJAX.

It's a typical scenario in web applications that they have some more or less static forms with input fields for displaying and editing data values. Therefore a concept to handle form situations is also needed in a AJAX engine.

This part is about the programming we need on the client. It's implemented in JavaScript and uses intensively XML documents. Some words about the extensions in the Proxy Generator as well as the implementation on the server will follow. Some samples can already be found on the samples WebSite.

### Classical approach and drawbacks

The mechanism behind form element works since the beginnings of HTML to send the data that is entered or changed by the user back to the WebServer. The format used by these postings has nothing in common with XML but uses for in many cases a URL like format defined by the mime type application/x-www.form-urlencoded. With the upcoming XHTML 2.0 definition also a XML Format and some more advanced form functionalities will be available. But all these good news are not supported by the wide spread browsers available today.

A direct access to the data inside a HTML Form in a whole does not exist in the browser nor in JavaScript. But of course the individual elements can be accessed one by one. The values that are shown to the user are normally included inside the HTML code that is sent from the server to the client. Collecting the values and sending them back to the server is hard coded to the submit action of the form that normally downloads new HTML Code and displays them in the window,

It's obvious that this approach doesn't fit into the processing of AJAX. The graphical elements of the form and the data is not separated properly.

### Using Form Data

One way to realize an access to the data in a HTML form for AJAX applications is to use JavaScript. The include file **ajaxForms.js** was written to enable the developer to handle this effectively and without using long scripts in every form but to help with a collection of methods.

The goal of this approach is to distinguish between the data of the form represented as a XML document and the HTML elements that are used to display and edit the data.

To define a form on a HTML page you need a HTML Element that contains all the form elements. This encapsulating element can be a DIV tag.

All the input fields, checkboxes, textareas and select objects with a name that are inside this HTML element are considered holding data in the appropriate object properties. This data is assembled into a XML document with a data root element by using the names of the HTML elements as the names of the inner tags.

This HTML code ...

```
<div id="frm" style="margin: 2px; padding: 2px; border: solid 1px green">
  <p>F1: <input name="F1" value="one by one"></p>
  <p>F2: <input type="checkbox" name="F2"></p>
  <p>T1: <textarea name="T1" style="width: 400px; height: 80px">some
```

```

text
lines</textarea>
  <p>S1: <SELECT name="S1">
    <option value="10">ten</option>
    <option value="11">eleven</option>
    <option value="12" selected="selected">twelve</option>
    <option value="13">thirteen</option>
  </SELECT>
</div>

```

... corresponds to this XML document:

```

<data>
  <F1>one by one</F1>
  <F2>>false</F2>
  <T1>some
text
lines</T1>
  <S1>12</S1>
</data>

```

The data of a form is now available as a single value and therefore fits well into the existing functionalities of the AJAX engine.

In AJAX applications it is typical that the same HTML Form is reused several times before getting and displaying new HTML code from the server so we also need methods for the reverse data flow that puts the data of an XML document back into the HTML form.

## AJAX Form Services

The implementation of the AJAX Form Services, that allows an efficient implementation of forms can be found in the JavaScript include file `ajaxForms.js`.

Again, only one global object named `ajaxForms` is defined to minimize possible name conflicts. Attached to this object the following methods are available.

**`data = ajaxForms.getData(obj)`**

This method searches all `INPUT`, `TEXTAREA` and `SELECT` elements that are contained inside the passed object and uses their values to build up the elements of a XML document.

The *obj* parameter can be passed using the id of a formula or as a reference to a HTML Object.

This result of this method is a `XmlDocument` object.

**`ajaxForms.setData(obj, data)`**

This method transfers all values from data into the corresponding `INPUT`, `TEXTAREA` and `SELECT` elements of the HTML form. All HTML elements that are part of the form but have no value in data will be cleared or unchecked.

The *obj* parameter can be passed using the id of a formula or as a reference to a HTML Object.

The data parameter can be passed as a XML text of a `XmlDocument` object.

This method also calls `clearErrors` at the end (see below).

**`ajaxForms.clearData(obj)`**

All HTML `INPUT`, `TEXTAREA` and `SELECT` elements that are part of the form are cleared or unchecked.

The data parameter can be passed as a XML text of a XmlDocument object.

#### **ajaxForms.resetData(obj)**

All HTML INPUT, TEXTAREA and SELECT elements that are part of the form get their initial value that is coded inside the

The obj parameter can be passed using the id of a formula or as a reference to a HTML Object.

#### **ajaxForms.processException(ex)**

AjaxForms implements a mechanism to display exceptions that are thrown by the server nearby the fields that causes them. If exceptions are handled by **ajaxForms.processException()** all ArgumentException are detected.

To enable the developer how the exception text is shown SPAN elements tagged with the css-class AJAXFORMEXCEPTION can be defined inside the form. The text of the exception is then inserted into this Element and can be formatted by using a CSS rule.

```
<span class="AJAXFORMEXCEPTION" name="LOANVALUE"></span>
```

#### **ajaxForms.clearErrors()**

This method clears all exception texts inside the form.

## Sample for AJAX Forms

The screenshot shows a web browser window with the address [http://www.mathertel.de/AJAXEngine/S02\\_AJAXCoreSamples/KreditCalc.htm](http://www.mathertel.de/AJAXEngine/S02_AJAXCoreSamples/KreditCalc.htm). The page title is "Credit calculation Form Sample". The form contains the following fields and values:

|                                                                  |                                     |            |                            |                                        |   |
|------------------------------------------------------------------|-------------------------------------|------------|----------------------------|----------------------------------------|---|
| Loan amount:                                                     | <input type="text" value="100000"/> | €          | Estimated Monthly Payment: | <input type="text" value="415.83"/>    | € |
| Yearly interest rate:                                            | <input type="text" value="3.99"/>   | % per year | Years to Repay Loan:       | <input type="text" value="40"/>        | € |
| Repayment:                                                       | <input type="text" value="1.00"/>   | % per year | Total payment:             | <input type="text" value="201366.41"/> | € |
| <b>Warning! Do not rely on these numbers, they may be false!</b> |                                     |            | Total interest:            | <input type="text" value="101366.41"/> | € |

Below the form is a section titled "KreditData" with a graph showing a blue area above a red area, representing loan data over time.

[http://www.mathertel.de/AJAXEngine/S02\\_AJAXCoreSamples/KreditCalc.htm](http://www.mathertel.de/AJAXEngine/S02_AJAXCoreSamples/KreditCalc.htm)

This sample implements the processing of data of a form using the AJAX engine and the AJAX Forms.

AJAX also brings some advantages to this kind of application on the validation and calculation of formula data. Instead of "sending" the data and retrieving new to be displayed HTML an asynchronous communication only transfers the data part of the form using XML documents.

The supplementing functions necessary for the handling of AJAX Forms it is necessary to include the ajaxforms.js scripts in the HEADER of the page.

```
<script type="text/javascript" src="ajaxForms.js"></script>
```

The connection to the corresponding WebService is done by using the proxy generator:

```
<script type="text/javascript" src="GetJavaScriptProxy.aspx?service=_Kreditcalc.asmx"></script>
```

In this example the validation of the input values and the calculation of the new values are together implemented in the same method. The communication of the values between the client and the server uses the XML format defined by ajaxForms.js module. The two functions ajaxForms.getData and ajaxForms.setData that implement the transfer of the data from the XML format to the HTML of objects therefore can be used directly in the setup of the actions.

### Exceptions

The function ajaxForms.processException can be used to handle and display the exceptions that occur within validating the passed data.

If a wrong or missing value is detected inside the passed form data the server should throw an ArgumentException using the name of the element as the name of the wrong parameter.

This name is then passed to the client using an exception return package.

ProcessException then recognizes that an ArgumentException was thrown and shows the error text beside the input field. To show these texts, a SPAN element is used for each field inside the form:

```
<span class="AJAXFORMEXCEPTION" name="LOAN"></span>
```

### The Form

The layout of the form and the data together with the format of the XML document is specified by the HTML code of the page. The DIV element with id frm (having a green border) is used as a container for all fields of the form. Inside this object you can find the INPUT elements like:

```
<input id="loan" name="LOAN" value="100000">
...
<input name="MONTHLYREPAY" disabled="disabled">
```

### The Actions

The first functionality on the server implements the calculation a simple credit case. On the left side of the form some values like the loan amount and the rates can be modified by the user. These values are used to calculate the monthly payment, the duration of the credit and the totals of payment that are displayed on the right side of the form.

For that purpose all the data of the form is send over the network to the server as a XML document as well as the result of the calculation.

The first action uses the functions getData und setData from ajaxForms.js and needs no further programming.

```
// declare an AJAX action
var action1 = {
  delay: 200,
  prepare: ajaxForms.getData,
  call: proxies.KreditCalc.Calc,
  finish: ajaxForms.setData,
  onException: ajaxForms.processException
} // action1
```

The second implemented method on the server gets the same XML document information as the first method but returns a XML Document containing the monthly balance of the credit.

The code on the client is the responsible for displaying the chart by using the special local implemented method `displayAmortization`. Here the XML document passed back from the server is transformed into a HTML document fragment by using a XSLT transformation on the client. This transformation uses a lot of small blue and red images that are sized and positioned as needed to make the chart look like it should.

Because the transformation is done on the client only about 23% of the necessary HTML code is transferred over the network. The XSTL is retrieved only once and is cached on the client.

```
var action2 = {
  name: "Amortization",
  delay: 800,
  prepare: ajaxForms.getData,
  call: proxies.KreditCalc.Amortization,
  finish: displayAmortization,
  onException: proxies.alertException
} // action2
```

### Starting the actions

To start the asynchronous actions again the `onkey` event is used. In this sample 2 actions are started by one event. Because both actions have a delay time specified they will stay in the queue and will be removed and started again when another key is pressed before the delay time is over.

The second parameter on starting the actions is used to reference the AJAX form.

```
onkeyup="ajax.Start(action1, 'frm');ajax.Start(action2, 'frm')"
```

## Use web services with multiple parameters in the AJAX Engine

While it is possible to use web methods with multiple parameters when using the web service proxies layer directly, the AJAX engine level is only supporting one parameter - but there is a small trick that help you out of this situation. Because the AJAX engine knows about all the asynchronous steps that have to be done to complete the whole functionality of a complete AJAX functionality it might make sense to use a `WebService` with more than a sinle parameter.

You can find a sample that simply adds 2 integers at

[http://www.mathertel.de/AJAXEngine/S02\\_AJAXCoreSamples/CalcAJAX.aspx](http://www.mathertel.de/AJAXEngine/S02_AJAXCoreSamples/CalcAJAX.aspx)

The `prepare` function is normally used for retrieving the client-side parameter from the html objects. Because we need more than one parameter we just do nothing here but returning the context object.

The `call` function, that is called next is not just pointing to the proxy function of the method of the web service but needs some more code. It retrieves all the parameters itself and then calls the proxy method `WebService`. The trick is that the mechanism that handles asynchronous result messages is not hooked up correctly and must be patched by using 2 lines of code for the regular and the exception case:

```
proxies.CalcService.AddInteger.func = this.call.func;
proxies.CalcService.AddInteger.onException = this.call.onException;
```



Now the call will be processed asynchronously and the result value will be passed to the *finish* function as usual.

Here the complete code that declares the AJAX action:

```
// declare an AJAX action
var action1 = {
  delay: 200, // wait for multiple keystrokes from fast typing people

  // the prepare function just returns the context object (if any)
  // and makes it available to the call function.
  prepare: function(obj) { return (obj); },

  // the call is not pointing to the webservice proxy directly but calls it using several
  // parameters.
  call: function (obj) {
    var n1 = document.getElementById("n1").value;
    var n2 = document.getElementById("n2").value;
    proxies.CalcService.AddInteger.func = this.call.func; // patch
    proxies.CalcService.AddInteger.onException = this.call.onException; // patch
    proxies.CalcService.AddInteger(n1, n2); // and call
  },

  // the result will now be processed as usual.
  finish: function (p) { document.getElementById("outputField").value = p; },

  onException: proxies.alertException
} // action1
```

## Application Aspects

The examples on AJAX I've explained up to now had to point out special aspects of the technical design and the implementation. They correspond to the "Bottom UP" method and are about the base components JavaScript, XMLHttpRequest, XML and the asynchronous processing - the aspects that give AJAX its name.

However, it is not the goal to play some technological games but to realize, improved or modernized a complete application with the assistance of the AJAX engine. The use of AJAX technologies fortunately does not need any principle proceeding at the level of the application but starts at the level of components.

There are 2 major directions of thinking I found useful to get "the big picture" of an AJAX enabled or an AJAX using web application.

The first direction is identifying useful patterns in the architecture. This helps to sort the responsibilities of the individual parts of the architecture.

The other direction is to have a look at the "developer's productivity" we need to enable a team of developers to build web applications in a give and acceptable time frame.

## Model View Controller (MVC) Pattern - Thinking in patterns

Thinking about the structure and the layering of the components of solutions is important for software architects. It's about getting the right distance to a concrete implementation, detecting the principles and unnecessary deviations.

Sometimes those thoughts exceed this range and lead into almost religious discussions about holy principles.

Discussing the Model-View Controller (MVC) is one of those topics that describes the distribution of objects and functionalities of a application with a user interface on a high level. There are books about I and a also the material on the web is very detailed.

Because AJAX style applications is for many developers a new kind of modeling web applications I have collected some of my thought on this topic.

Here I want to look and analyze the existing AJAX Engine and not build a ideal optimal AJAX architecture.

### Definition of MVC

There are many different definitions for the Model-View-Controller Pattern (google it). I stay with the short one that can actually be found at Wikipedia:

<http://en.wikipedia.org/wiki/Model-view-controller>

- Model: This is the domain-specific representation of the information (data and methods) on which the application operates.
- View: This renders the model into a form suitable for interaction, typically a user interface element.
- Controller: This responds to events, typically user actions, and invokes changes on the model or view as appropriate.

### Model

The Model part of the architecture all is relatively easy to discover in the WebServices. Here all functional aspects should be discoverable. There are of course also functional aspects in the User Interface but this code is

implementing a more comfortable way of interaction and handles topics of the presentation.

Based on the role this code takes over in the whole architecture some typical aspects very similar to the SOA architecture of the model can be identified.

Here some keywords:

- Independent of HTML objects
- No culture or language specific implementations
- Manipulating and storing data
- Stateless
- Checking and securing of the parameters of the webmethods.
- Reliable algorithms and functionalities

#### View

The View part can be localized in the realization of the User Interface by using HTML objects and some supplement JavaScript methods.

The base functionality of the HTML objects like editing values in input fields is covering a lot but also a lot of the JavaScript implementation in the pages of my samples for example the LookUp List and displaying graphics is part of the view.

- UI Events
- Active elements like input fields
- Layout elements and decorative additions
- Converting of numbers and date formats to strings
- Methods to support the working

#### Controller

But where is the Controller part in a AJAX application and how does it work ?

A simple answer is obvious: The AJAX Engine. Beside the WebService and the HTML there are only script include files left to implement this part. The proxy methods for the WebServices can be defined as part of the network infrastructure.

#### Page Controller

When comparing the traditional ASP.NET Web Forms with the MVC pattern the page-controller term is used very often. When comparing the 2 most used approaches ASP.NET and Struts you can easily see that the server side implementation only covers the collaboration of the objects of a single page.

The transitions between the pages are not really supported with a structure by ASP.NET 1.1. Hyperlinks or Redirect statements are used on the server and some problems have to be solved for transferring information from one page to the next one. With ASP.NET 2.0 it is possible to have 2 pages in memory at once and transferring data is easier to implement but this is no solution for controlling the global flow of a web application.

Struts mixes up both aspects what may end in complex struts-config.xml files.

Both approaches for realizing web applications are described and compared regarding the MVC pattern in

<http://msdn.microsoft.com/library/en-us/dnasp/html/ASPNet-ASPNet-J2EE-Struts.asp>

as well as in

[http://struts.apache.org/userGuide/building\\_controller.html](http://struts.apache.org/userGuide/building_controller.html).

When realizing an AJAX application the controller is shifted to the client and is implemented in JavaScript. Like with ASP.NET it can control only the activities of the one loaded page because all local information is lost when navigation to another page and a new UI and controller is built up. This is the reason why some AJAX applications have only one active URL using a complex page or framesets.

I have to compare this situation with the 2-tier application architecture of the 90's. Here the controller also was implemented on the (fat) client and the server added data and methods (SQL connections and stored procedures).

## Developer's productivity – Layers of abstraction

If you have to lead a team of a dozen developers that have to build a huge and complex web application you will need a set of specialized knowledge and people with very different skills. You will have database specialists, front-end designers and people that know a lot about the processes that have to be implemented just to name a view. Now you might think about using AJAX technology in your application you will add another very special technology into that team.

There are some ways to handle this in real life, now in year 2005:

- You have an extraordinary and fabulous team, which also understands that extra technology.
- You identify a member of your team who does all the AJAX specific tasks and keeps all the deep secrets of it away from the other developers
- You buy a framework, AJAX engine or components from a company that can give you support and will provide you with updates.

If you can choose the first option – be glad!

You might choose the third option and pay some money to the people that can implement all the AJAX code you need. Not a bad option! There are already some commercial frameworks available and sure all others will follow and put will adopt some “AJAX” or “Web 2.0” functionality.

Microsoft will also ship a framework (no license fee, but still learning and support) in a few months, called “atlas”.

<http://atlas.asp.net>.

If you choose the second option you can get a lot of good work and a quick start from some open source frameworks (and I hope also with parts from my AJAX engine). In this scenario it is important to have a clear separation of the responsibilities and building components that encapsulate the specific AJAX details is the best approach I found for that.

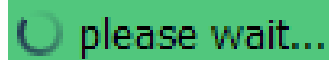
The JavaScript proxies for the WebServices are a first step to get away from programming with the XMLHttpRequest object directly and to deal with xml nodes that build the SOAP protocol. You just use a client side method to call a server side method.

The AJAX Engine again brings a layer of abstraction that enables your developers to write down the AJAX  $\mu$ -process in an almost linear coding style even if it gets executed by using several events, callback functions, timers, a queue and caching mechanism. The shown linear programming style by defining the AJAX action object is quite easier to understand.

Both elements of the AJAX engine are a layer of an abstraction that make it easier to build AJAX applications and makes your team faster.

## Less waiting on AJAX

It's a psychological phenomenon that waiting takes less time as long as something happens.



I'm sure you have seen all these nice rotating arrows, bouncing points or bars that are commonly used for situations where a progress bar should appear to tell the user that it's time for waiting like spinning a globe in IE or the rotating points in Firefox, on the windows start up screen on pageflakes.com or even in windows media edition. It can also be as simple as the [Loading...] text that is used by the Google-Mail user interface.

<http://www.pageflakes.com/>

On [www.ajaxload.info](http://www.ajaxload.info) you can easily generate "AJAX-" animated gif files I thought it is time to implement a few lines into the AJAX Engine.

<http://www.ajaxload.info>

The right place to start displaying a progress indicator is just before starting the webservice call to the server. Here I call `StartProgress()`.

But it also has to be hidden after anything happens that ends the action like when the return value is there, an exception happens or when the timeout timer strikes. To identify these places I searched the code for the `ajax.current` gets cleared. There I call `EndProgress()`;

The first implementation was straight forward creating and removing a html element. After some testing I found that this costs me more time than the real call over the internet and in many situations immediate responses got slower and that's definitively not that what I wanted to achieve.

In the end I came to the following solution:

- The `StartProgress` function only sets a flag (`ajax.progress`) to true and starts a timer (`ajax.progressTimer`) with a timeout of 220 msec.
- This time was chosen by some testing and many server calls do not last so long and therefore need no progress indicator.
- When the timer strikes it calls the `ajax.ShowProgress` function. Here I implement the real code that creates the HTML element or just shows an existing one again.
- The `EndProgress` function clears the flag and also starts the timer but with some less waiting.
- When the timer strikes after a call has finished the existing object is just hidden.

This architecture has some advantages. First the progress indicator is not shown when short calls are made and when multiple calls are made one after the other it is not hidden. This can save a lot of flickering.

Here are the specific new functions:

```
// ----- show or hide a progress indicator -----
// show a progress indicator if it takes longer...
ajax.StartProgress = function() {
  ajax.progress = true;
  if (ajax.progressTimer != null)
    window.clearTimeout(ajax.progressTimer);
  ajax.progressTimer = window.setTimeout(ajax.ShowProgress, 220);
} // ajax.StartProgress

// hide any progress indicator soon.
```

```
ajax.EndProgress = function () {
  ajax.progress = false;
  if (ajax.progressTimer != null)
    window.clearTimeout(ajax.progressTimer);
  ajax.progressTimer = window.setTimeout(ajax.ShowProgress, 20);
} // ajax.EndProgress

// this function is called by a timer to show or hide a progress indicator
ajax.ShowProgress = function() {
  ajax.progressTimer = null;
  var a = document.getElementById("AjaxProgressIndicator");

  if (ajax.progress && (a != null)) {
    // just display the existing object
    a.style.top = document.documentElement.scrollTop + 2 + "px";
    a.style.display = "";
  } else if (ajax.progress) {
    // find a relative link to the ajaxcore folder containing ajax.js
    var path = "../ajaxcore/"
    for (var n in document.scripts) {
      s = document.scripts[n].src;
      if ((s != null) && (s.length >= 7) && (s.substr(s.length -7).toLowerCase() == "ajax.js"))
        path = s.substr(0,s.length -7);
    } // for

    // create new standard progress object
    a = document.createElement("div");
    a.id = "AjaxProgressIndicator";
    a.style.position = "absolute";
    a.style.right = "2px";
    a.style.top = document.documentElement.scrollTop + 2 + "px";
    a.style.width = "98px";
    a.style.height = "16px";
    a.style.padding = "2px";
    a.style.verticalAlign = "bottom";
    a.style.backgroundColor="#51c77d";

    a.innerHTML = "<img style='VERTICAL-ALIGN:bottom' src='" + path + "ajax-loader.gif'> please
wait...";
    document.body.appendChild(a);
  } else if (a) {
    a.style.display = "none";
  } // if
} // ajax.ShowProgress
```

If you want to see how it looks like you can use the old prime factor sample. Try some long the numbers like: 98798798789878987. You might see it only if someone else is stressing the server too - It seems to be a powerful machine :-) and prime factors get calculated fast even with my stupid algorithm.

## Thinking in components

The next level of abstraction brings you again a huge amount of productivity.

Now, with components, web controls or tag libraries that again encapsulate the AJAX actions you can build complex AJAX enabled web sites by combining some pre-coded controls. If you have to code many web pages that share common functionality you can identify these needs and pay for building your special controls. You will find the productivity by those developers that use these components.

## Building AJAX Controls

In many cases we can wrap a special AJAX functionality into a control instead of coding the JavaScript code into the page itself. To reach a high percentage of reusability those components are realized outside of the individual pages and are parameterized for their concrete usage.

There are some advantages by doing that.

When implementing conventional (old style) web forms you are familiar with using controls together with html elements to build up the whole page. If we build AJAX into new or existing controls the web programmer can continue to build the pages using the well known techniques. Also the form editors with the rich UI experience can be used again.

AJAX functionalities can be used together with the standard executing mechanism of posting form data to the server. In a web form an AJAX control can validate the current content of a HTML form in the background and guide the user before a regular posting of the form data starts the transaction or processing.

The LookUp example that was introduced at [OrteLookup.htm](#) in [the second parts of the samples](#) is using the AJAX engine directly. You can also find a [LookUp Control sample page](#) in the collection of the AJAX Controls in [the third part of the samples](#) that does the same thing but is reusable on other pages.

The amount of JavaScript code that has to be loaded on the client for the LookUp control is only about 18 kByte for the AJAX engine and the additional code for handling the selection list not very large.

It helps however to save a multiple of this size on the network transfer because the data portion does not have to be sent completely to the client. The JavaScript code that is extracted into a separate file is also transferred only once when reloading of the pages saves several times.

# JavaScript Behaviors

## The Behavior mechanism

The common idea behind the behavior implementations is using HTML (or XUL) to implement an object model that defines a client-side component and to add the functionality for this component by using a separate and reusable JavaScript definition file.

CSS attributes can be used in conjunction with behaviors to specify how the objects are displayed.

With this separation of concerns we get a well organized infrastructure for the scripting on the client without the need for maintaining JavaScript code fragments in many places.

The word "Behavior" was introduced by the DHTML Behavior technology developed by Microsoft for IE 5.x and later and the XBL technology implemented in the Mozilla browser project.

## Building AJAX Controls

JavaScript Behaviors are a key technology for building reusable html components or controls with a rich functionality attached to them without relying on any vendor specific proprietary implementations. Beside implementing the JavaScript that builds the implementation of the functionality some more problems should be solved.

When using AJAX functionality or if you want to enrich your web pages with client side functionality you also have to deal with the complexity of standardizing and reusing JavaScript code on the client. The JavaScript Behaviors mechanism is offering a solution for this client side architectural problem.

To extend the reusability of components on a server side platform almost every web server framework offers a way of declaring server side tags or server side elements that will produce some valid html for the client. In this chapter I will use ASP.NET user controls and C# as the server side framework. These kinds of components are easy to implement and adapt. They offer already enough possibilities compared with ASP.NET Web Controls that must be implemented as classes but we also have to implement real web controls in some cases.

Because the core mechanism of JavaScript Behaviors is implemented in JavaScript and is running on the client platform it can easily be ported to other server side platforms and languages.

By using some naming conventions the implementation and the organization of the files on the server will become transparent and easy to understand.

## Delivering JavaScript functionality

HTML pages without any more than HTML offer only a limited functionality and allow only displaying information and filling out forms. As a result HTML pages are often extended by using JavaScript for handling input validation, reacting on clicks or bringing optical effects.

But when it comes to reuse once-written functionalities there is no built-in concept to HTML and JavaScript except the flat programming model of include files. Copying the source code around is the often used response to this weakness but will lead into unsupportable huge web sites.



When tailoring functionality into components the JavaScript methods that make up these components can be placed in separate files, one file for each component type. When using multiple different JavaScript Behaviors then multiple JavaScript include files are downloaded to the client; one for each type but only one when using the same behavior multiple times. These files have static content only and can be marked as cacheable for the browser so reloading the same file will result in cache lookups and not in reloading the from the server. This brings down network traffic and can efficiently speed up the initial loading of a page.

This approach is preferred over putting all the behaviors into a single file or sending the JavaScript code together with the page source code to the client.

This technique has nothing (not yet) to do with AJAX because it is a general usable implementation for reusing JavaScript code for components. Also, this technique has also nothing to do with ASP.NET and can easily be ported to Java based server because it only uses the client-side (browser) available HTML objects and JavaScript.

It will be the client side functionality for the upcoming AJAX controls and it fits perfectly into a server-side HTML controls framework that enables also the reusing of HTML code fragments.

## Browser specific proprietary behaviors

Both most used browsers, Internet Explorer from Microsoft and the Mozilla/Firefox from the Mozilla foundation, have both their own and incompatible way to allow reusing JavaScript functions and HTML on a component level

### Internet Explorer

The Microsoft implementation was introduced with the Internet Explorer 5.5 to encapsulate functionality that extends the standard implementation of HTML elements. This can be done by implementing COM interfaces (in C++)

<http://msdn.microsoft.com/workshop/browser/behaviors/howto/creating.asp>

or by writing script in \*.hta files.

<http://msdn.microsoft.com/workshop/author/behaviors/overview.asp>

### Mozilla

The Mozilla project introduces another technology called XUL that is used to define the User Interface for the applications of this suite.

<http://developer.mozilla.org/en/docs/XUL>

XBL (Extensible Binding Language) is part of this technology and is used to define the bound behavior.

<http://developer.mozilla.org/en/docs/XBL>

Common to both solutions is the big advantage over global script include files that the functionality is bound and accessible through a specific HTML object, can be bound to multiple objects and support specific methods, attributes and event handlers.

Will a standard become true?

There are many ideas about a standardization of a custom behavior of existing or new html tags. Here is a list of w3c drafts that show how many different approaches where submitted:

HTML Components (1998)

<http://www.w3.org/TR/1998/NOTE-HTMLComponents-19981023>

Behavioral Extensions to CSS (1999)

<http://www.w3.org/TR/becss>

XBL - XML Binding Language (2001)

<http://www.w3.org/TR/2001/NOTE-xbl-20010223/>

XML Binding Language (XBL) 2.0 (2006)

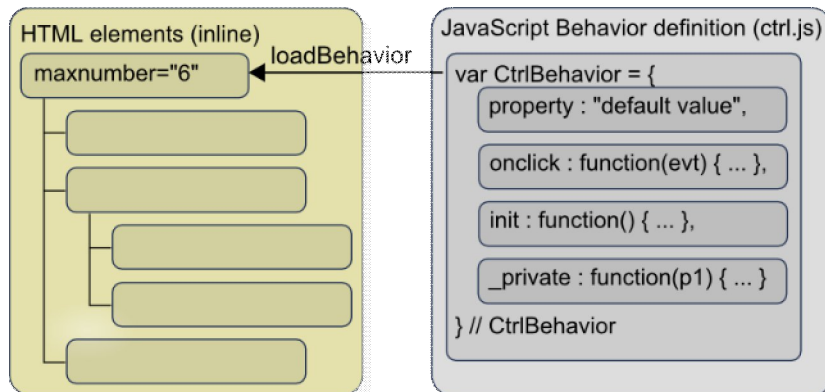
<http://www.w3.org/TR/2006/WD-xbl-20060907/>

Common to all these approaches is that the specific details are implemented by using JavaScript and the HTML Object model (DOM).

Before one (or another) behavior mechanism will become a standard at least some years will pass so we have to look for a cross browser way of implementing it.

## The JavaScript behavior mechanism

As mentioned before, the basic idea behind the JavaScript Behavior mechanism is using HTML elements for defining an object model of the component and using JavaScript to implement the functionality. Here is a small picture of the situation:



You can see that the outer HTML element that can have inner HTML elements will be bound to the Behavior that is specified in JavaScript.

There are properties, events, public and private methods that make up the Behavior object that is specified by using the JSON syntax.

### A step by step instruction

The easiest way to understand building a new client side component by using the JavaScript behavior mechanism is to do follow a step by step sample. It is first implemented without using any server side help by only implementing JavaScript in a plain html file. Then the JavaScript Behavior is modeled and at last it is migrated into an ASP.NET user control.

There are 2 advantages when working this way. First you can focus on the JavaScript implementation from the beginning and can avoid the typical cache issues when using included JavaScript files and second you can see some of the reasons where Behaviors take advantage over plain JavaScript implementation.

The sample functionality I use to show this is a simple dice (German: Würfel).

It basically consists of defining new properties, specific methods and event handlers by building a JavaScript prototype object for each behavior. Also a common binding functionality is needed that attaches these definitions to a HTML object after the page is loaded. Script include files can be used to bring the prototype objects into the pages.

You can download all files from

[http://www.mathertel.de/Downloads/Start\\_JSBTutorial.aspx](http://www.mathertel.de/Downloads/Start_JSBTutorial.aspx).

## Building the JavaScript Behavior Basics

### 1. Coding it all in one place

The best place for writing a new control is inside a single HTML file that contains all the fragments that you will separate later into different locations:

- A JavaScript tag that includes the common behavior loading mechanism `<script type="text/javascript" src="../controls/jcl.js"></script>` in the `<head>` element.
- A CSS section inside the `<head>` element that will hold all the style rules that we will later move out into the common css file. You can code all the css rules into the html elements first if you like. Later you should not include any CSS code inside the rendered html elements to make some personalization and style adoption easier.
- The `<script type="text/javascript">` element that will contain the JavaScript behavior definition using a object notation in the JSON coding style and the statement for binding the JavaScript behavior object to the HTML element.
- A HTML object structure that will be used for rendering the new control. This should be a single outer element that may contain complex inner HTML elements. Give it a unique id that can be used to identify the first prototype.
- And a small `<script type="text/javascript">` element that will cause to bind the defined Behavior to the HTML element.

The advantage of using this intermediate development state is that you can hit F5 in the browser and can be sure that all your code will reload as expected. You also will not have any timing problems that may happen when JavaScript or CSS files are cached locally. You need no server side functionality so a \*.htm file is fine for now.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Ein Wuerfel</title>
  <script type="text/javascript" src="jcl.js"></script>

  <style type="text/css">
    .Wuerfel {
      border: solid 2px green; width:40px; height:40px; overflow:hidden;
      cursor: pointer; background-color:#EEFFEE;
      font-size: 30px; padding:20px; text-align: center;
    }
  </style>

  <script type="text/javascript">
    var WuerfelBehaviour = {
      onclick: function(evt) {
        Wuerfell.innerHTML = Math.floor(Math.random()*6)+1;
      }
    } // WuerfelBehaviour
  </script>
</head>

<body>
  <div id="Wuerfell" class="Wuerfel" unselectable="on">click</div>

  <script defer="defer" type="text/javascript">
    jcl.LoadBehaviour("Wuerfell", WuerfelBehaviour);
  </script>
</body>
</html>

```

The file [wuerfel\\_01.htm](#) contains an implementation in this state.

## 2. replacing all hard-coded references

If you want to make it possible to use the same control multiple times on the same page then you must avoid using hard coded ids or names. The only place where you should find the id of the outer HTML element is inside the first parameter of the `jcl.LoadBehaviour` function call.

All the other references should be replaced by using the "this" reference.

The other thing you should take care too are the parameters / attributes that you want to use together with the new control. You should define the as attributes in the outer HTML element and as properties of the JavaScript behavior definition. There should not be any constants inside the JSON object.

If everything is well done you can make a second copy of the outer HTML element with a new id and can bind the same behavior definition to it. Both elements should now work as expected independently. Check also if the parameters work as expected.

The file [wuerfel\\_02.htm](#) contains an implementation in this state.

### 3. separating the behavior code

The next step is to extract the core of the behavior into a new \*.js file and reference this file by using a new `<script type="text/javascript" src="wuerfel.js"></script>` in the `<head>` element.

The advantage of a separate file for the behavior definition is that the implementation can be cached by the browser independently from the individual use and If the control is reused in different pages you can see dramatic performance improvements.

The file [wuerfel\\_03.htm](#) and [wuerfel.js](#) file contain an implementation in this state and [wuerfel.js](#) has also got some more functionality.

### 4. separating the CSS style definitions

The style of the new control should not be coded inline into the html code but should be separated into some css statements. So I use a classname for the top element of the control by using the name of the behavior. If you have special inner elements they can be prefixed by the same name or you might use css selectors by specifying the outer and inner class names. Sample:

```
div.TreeView .do { ... }
div.TreeView .dc { ... }
```

Because the css statements are usually much smaller then the JavaScript code for a control I do not extract the css statements into separate files but include them all in a single css file for all the controls I've done. The \*.css files are cached by the browser so loading them from the server doesn't occur too often.

## Integration into the ASP.NET framework

Now it's time to reduce the complexity of USING the new control for developers by using the rich features of a server framework.

### 1. converting to a ASP.NET User Control (\*.ascx)

Now it's time to switch from a \*.htm file to a \*.aspx file because you will need some server side functionality now.

Rename the file and add a `<%@ Page Language="C#" %>` statement at the top of the page. In Visual Studio you will have to close the file and reopen it to get the full editor support for the right server side languages.

The html code and the javascript statement that binds the JavaScript Behavior of the new control is copied into the new User Control file [wuerfel.ascx](#).

The id attribute that is rendered for the client should not be hardcoded to a static value. The UniqueID can be used and will produce the given id if one is specified in the \*.aspx page.

```
<div id="<%=this.ClientID
    %>" class="Wuerfel" unselectable="on">click</div>
<script defer="defer" type="text/javascript">
    jcl.LoadBehaviour("<%=this.ClientID %>", WuerfelBehaviour);
</script>
```

Now it is easy to include the new control into the page by dragging the wuerfel.ascx file into a blank page while using the Design mode. The code will look like this:

```
<ucl:Wuerfel ID="Wuerfel1" runat="server" />
```

and a reference to the used control will also be generated:

```
<%@ Register Src="Wuerfel.ascx" TagName="Wuerfel" TagPrefix="ucl" %>
```

Open this file by using the browser and have a look to the source code that is delivered to the client - it will look very similar to what you had before. Again you can check whether everything is fine by pasting the <ucl:Wuerfel...> element several times. Visual Studio will automatically generate different ids so all elements work independent.

## 2. using the script including mechanism

You still have to take care of including the right JavaScript include in the <head> of your page. Now we also get this work done automatically. The advantage is that you do not have to take care of using the right include files and you will never forget to remove them when a control is removed from the page.

In the \*.aspx page the <head> element must be marked with runat="server"

In the \*.ascx file some server side programming is needed inside a <script runat="server"> tag:

```
protected override void OnPreRender(EventArgs e) {
    base.OnPreRender(e);

    if (Page.Header == null)
        throw new Exception("The <head> element of this page is not marked with
            runat='server'.");

    // register the JavaScripts includes without need for
    a Form.
    if (!Page.ClientScript.IsClientScriptBlockRegistered(Page.GetType(), "CommonBehaviour")) {
        Page.ClientScript.RegisterClientScriptBlock(Page.GetType(), "CommonBehaviour", String.Empty);
        ((HtmlHead)Page.Header).Controls.Add(new LiteralControl("<script type='text/javascript'
src="
        + Page.ResolveUrl("jcl.js")
        + "'><" + "/script>\n"));
    } // if

    if (!Page.ClientScript.IsClientScriptBlockRegistered(this.GetType(), "MyBehaviour")) {
        Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "MyBehaviour", String.Empty);
        ((HtmlHead)Page.Header).Controls.Add(new LiteralControl("<script type='text/javascript'
src="
        + Page.ResolveUrl("Wuerfel.js")
        + "'><" + "/script>\n"));
    } // if
} // OnPreRender
```

Have a look at the files [wuerfel\\_04.aspx](#) and [wuerfel.ascx](#).

### 3. using a global registration for the control

When dragging a User Control onto a page the UserControl is registered for this page by using a server site Register tag.

```
<%@ Register Src="Wuerfel.ascx" TagName="Wuerfel" TagPrefix="uc1" %>
```

There is no real problem with that automatic stuff but if you copy HTML code around from one page to another you always have to take care of copying these Register tags as well.

Fortunately there is another solution that registers User Controls globally in the web.config file and needs no Register tags.

Open the web.config file you can find in the root of your web application and locate the <configuration><system.web><pages><controls> region. Here you can add a add element:

```
<add src="~/controls/LightBox.ascx" tagName="LightBox" tagPrefix="ve"/>
```

You can find many samples in the web.config file of the AJAXEngine demo web site project and there is a good post on this topic in Scott Guthrie's blog too:

<http://weblogs.asp.net/scottgu/archive/2006/11/26/tip-trick-how-to-register-user-controls-and-custom-controls-in-web-config.aspx>

### 4. converting to a ASP.NET Web Control (\*.cs) implementation

When writing simple controls without nested other controls there is no need to convert a UserControl into a WebControl. You need this step only when the control will be used as a wrapper to more HTML code that is declared on the web page and not within the control itself. If you download the complete source code of the AJAX Engine project you can find some advanced implementations using ASP.NET Web Controls in the APP\_Code folder. Writing WebControls and designers for Web Controls is not covered here.

## Building JavaScript Behaviors - Properties, Attributes and Parameters

The way parameters are passed to the behavior implementation is some kind of tricky:

1. you write down a parameter into the \*.aspx source file when using an instance of the control on a page.
2. when the page is called from the server the parameter is passed to the server side control.
3. the parameter is then written out into the response (html) stream and send to the client.
4. when the behavior is attached to the html element it is made available to javascript.
5. the behavior implementation is using the parameter by using *this.parameter*.

There are some traps and tricks on the way.

Take care of uppercase characters in the parameter name. Parameters with uppercase characters work fine on the server but using them on the client breaks the xhtml format spec. You can use lowercase parameters on the server and the client and you don't get confused when writing code for the server platform and the client platform the same time.

If you want to make a parameter available to the server control you have to add a public property or a public field to the class.

Using public fields is working fine but Visual Studio will not help with intellisense then so I prefer using an encapsulated private field using a public property:

```
private string _datatype = String.Empty;

public string datatype {
    get { return _datatype; }
    set { _datatype = value; }
} // datatype

public string working_but_no_intellisense = String.Empty;
```

Passing null through a parameter just doesn't work because you cannot specify an attribute for a xml or html tag with a null value. I am using empty strings instead.

If no attribute value is specified in the source code you need to define a default value. The easiest is to assign the default value to the private server field declaration and always render the attribute into the response stream.

The Firefox browser makes a big difference between attributes of a html element and a property of an object so when attaching a behavior to a html element all the attributes are copied into object properties.

Don't use any reserved words you know from C#, VB.NET, JAVA, JavaScript, HTML or the DOM specification as a name and don't start a name with "on" because this naming convention is used for identifying event handlers.

### 1. Adding a parameter to the dice sample server control

The sample up to now is only showing random numbers from 1 to 6. A new parameter named "maxnumber" should make it possible to get random numbers between 1 and any positive number greater than 2.:

```
private int _maxnumber = 6;

public int maxnumber {
    get { return _maxnumber; }
    set { _maxnumber = value; }
} // maxnumber
```

This parameter is not needed on the server side and we just pass it to the client through a html attribute:

```
<div id="<%=this.ClientID %>" class="Wuerfel" maxnumber="<%=this.maxnumber %>"
    unselectable="on">click</div>
```

You can find this implementation in [wuerfel2.ascx](#).

### 2. Adding a parameter to the behavior

On the client side we need to declare that parameter as well. The given assignment will always be overwritten by the attribute the server adds to the html element.

And then we must use.

```
// parameter to set the maximum number
maxnumber : 6,

// find a random number
var n = Math.floor(Math.random()*(this.maxnumber-1))+1;
```

You can find this implementation in [wuerfel2.js](#).



### 3. Using the new feature

Now the new parameter can be used on any wuerfel2 tag:

```
<ucl:Wuerfel2 ID="Wuerfel2" maxnumber="42" runat="server" />
```

You can find it in [Wuerfel\\_05.aspx](#).

## Event handling

### Methods for event handling

The methods that are used to handle events from the mouse, keyboard or system are identified by their name prefix "on". When the behavior is bound to the HTML element these methods are not just copied over from the JavaScript behavior declaration to the html element but are wrapped by a special function that looks like

```
function() {
    return method.apply(htmlElement, arguments);
}
```

This wrapper is generated automatically for all methods of the behavior that start with the 2 characters "on" to ensure that the JavaScript "this" pointer is pointing to the htmlElement the method belongs to. This really simplifies writing event code. Keep in mind that this special attachment of methods for events is based on this naming convention so don't name other methods of the control this way.

### Simple Events

The first sample already used an event (onclick) and registered a method to calculate a new random number for the dice.

```
// classical event handler implementation
onclick: function(evt) {
    evt = evt || window.event;
    var src = evt.srcElement;

    src.rolling = 50;
    src.count = 0;
    src.rollNext();
}
```

Because the "this" pointer is adjusted to the htmlElement we can use it instead of finding the right element through the event property srcElement:

```
// simpler event handler implementation
onclick: function(evt) {
    this.rolling = 50;
    this.count = 0;
    this.rollNext();
}
```

### Global Events

Sometimes it is not possible to implement an event code by using this simple on\_\_ naming scheme because the event that the behavior needs is not thrown to the htmlElement of the behavior.

If you are interested in global events you need to attach a method by using the AttachEvent method that is available through the jcl object. Don't use the on\_\_ naming scheme for this method:

```
jcl.AttachEvent(document, "onmousedown", this._onmousemove);
```

Implementing event handlers was simplified with the first 2007 version. There is no more the need for the special coding to get the compatible event object.

If you are not interested in these events all the time the handler can be detached by calling:

```
jcl.DetachEvent(document, "onmousemove", this._onmousemove);
```

### Mouse Events

Mouse events are a little bit special when implementing a drag & drop scenario. The `onmousedown` will always be raised on the element that will be dragged around but the other 2 events `mousemove` (while dragging) and `onmouseup` (when dragging ends and the drop occurs) may be raised on any other element on the page or even on the document object itself. Because the event "bubbles up" we can get all the events by attaching these 2 methods to the document object.

The sample at [VBoxDemo.aspx](#) is using the [VBox.js](#) behavior that allows changing the width of the vertical separation by dragging the line between the left and right content.



## Details of JavaScript Behavior Definitions

### LoadBehaviour

The behavior functionality must be bound to the HTML object by calling one method only:

```
LoadBehaviour("TimerSample", TimerSampleBehaviour);
```

The “magic” work behind the LoadBehaviour() method is to loop through all elements of the given JavaScript object and assigning each of them to the HTML object.:

- Methods. Starting with "on" are assumed to be event handlers and are attached using the browser specific methods for attaching events. Only the bubbling phase of the event is used because it is implemented by both browsers.
- Other methods are just bound to the HTML object as function objects.
- All the other objects are assigned if there is not already an assigned property existing on the HTML object. These objects are shared for all HTML objects that are bound to the same behavior.

### Using complex properties in Behavior definition

When defining complex objects (arrays, structures) inside the JavaScript prototype you have to pay attention to the fact that these objects are shared among all HTML objects using this prototype. If individual complex objects are needed it is better to create and assign them in the "init" method.

The `jQuery.cloneObject` method is available and may help to get a real copy of a complex object that is declared inside the JSON object that builds the Behavior definition.

### The init method

A method named "init" can be defined in the JavaScript prototypes to implement special needs after the methods, event handlers and properties are bound to the HTML object. This method is automatically called by the LoadBehaviour method.

The "init" method is called when the onload event is raised for the window object.

### The term method

Another method named "term" can be defined in the JavaScript prototypes to implement a method that is called just before a page unloads. There is a real need for implementing code just before all HTML and JavaScript objects are thrown away in Microsoft Internet Explorer, because there is a well known problem with the memory management used by the HTML DOM and JavaScript. Both object libraries do have their own garbage collector implementation and in some cases circular references between objects of both worlds are not properly detected and memory will not get freed and useless memory consumption is the unwanted result.

The best solution against this “design weakness” is to set all references of JavaScript variables to HTML elements to null in this method.

There is also a great tool and some more background information available that helps to detect this kind of memory leak named Drip available at <http://outofhanwell.com/>.

You can find a sample using Clone Object inside the LookUp Behavior. Here this method is used to get a copy of an action definition that is declared inside the Behavior definition. This enables the usage of multiple instances of the same behavior on a page.

### Defining event handlers

All defined elements of the Behavior Definition object that start with the 2 chars “on” are treated as event handlers. This special naming convention is needed because most event handlers need a reference to the event object that contains most of the details on the current executed event.

Don't use names for your properties that start with “on” to avoid conflicts with the event attaching mechanism!

These methods get passed the event object as a parameter and will be executed in the context of the behavior.

To access the root object of your component you can use the **this** reference. When you need to access the object that got the event you can find it by using `evt.srcElement`. Here is a typical start sequence of an event handler:

```
onmouseover: function(evt) {
    var obj = evt.srcElement;
    ...
}
```

### Inheritance

A new behavior can inherit all details from an existing behavior. This mechanism is useful for building several JavaScript Behaviors that all share the same basic functionality. Here is a sample:

```
// this is a basic behavior definition for menubars
var EditMenubarBehavior = {

    inheritFrom: MenubarBehavior,

    init: function() {
        MenubarBehavior.init.call(this);
    },
    ...
}
```

By specifying the **inheritFrom** attribute the **LoadBehaviour** method will copy all properties, methods and event handlers from the referenced behavior to the specified html object before copying the new properties, methods and event handlers.

In the **init** method you should call the init method of the referenced behavior before continuing the initialization of the new features.

## The common JavaScript include file

The AJAX Controls are using the JavaScript behaviors and therefore need all the common include file “~/controls/jcl.js ” that implements the loading mechanism and most of them are implemented by using a specific include file containing the specific JavaScript behavior prototype code of one behavior.

```
var isIE = (window.navigator.userAgent.indexOf("MSIE") > 0);
```

The variable *isIE* is defined globally and is set to true if an Internet Explorer is the current browser so it is easy to detect the browser specific JavaScript implementations that we have to make for IE.

```
var jcl = {...}
```

The variable *jcl* is used to hold all functions of the behavior loading mechanism.

Because many controls need to share some local information to get synchronized a mechanism called “Data connections” is also implemented in this file and available through `jcl.DataConnections.*`.

Another reason for this include file is to implement a compatibility layer for FireFox/Mozilla that emulates some useful IE methods and properties.

## A naming convention

When defining properties and methods there is no reserved word in JavaScript that declare them local or global available. I used the naming convention that property and method names that start with an underscore character should not be referenced from outside of the include file.

## Enabling the caching of JavaScript files

Because JavaScript files are static they can be cached by the browser in the local cache directory to speed up the reloading of the same page multiple times.

I saved all these \*.js files and all the other files of the controls in the ~/controls directory. There are many references to these files that are hard-coded to this directory so if you rename it, it will cause some problems.

## Cross Browser JavaScript

### Introduction

If you have to implement a Web application that intensively uses JavaScript for some client side effects or some AJAX like features, and if you have to support more than Microsoft Internet Explorer 6.0, then you might find here a not widely known trick to make your programming easier by implementing a small compatibility layer.

Both browsers, Microsoft Internet Explorer and Mozilla/FireFox, do have a lot of functionalities as defined by the standards HTML, CSS and JavaScript, and some of them are really useful for implementing some functionality on the client. Because there are some extensibility features available on these different platforms, it is possible to extend one browser with methods and properties that are not available in the other one out of the box.

### JavaScript Prototypes

The JavaScript language interpreters included in the browser platforms offer almost identical functionalities. There is no real need to use this feature to bring some missing functionalities to one of the platforms, but it is an interesting feature anyway and it gives you the chance of extending both platforms.

The object types of JavaScript, better called intrinsic objects of JavaScript (*String*, *Number*, *Array*, ...) that are part of the JavaScript language have an extensibility mechanism by using the available *prototype* property.

By attaching new functions to the *prototype*, you make this method available to all objects that derive from that type.

The following sample adds a *trim()* method to all string objects that eliminates all leading and trailing blanks and returns a new string with the remaining characters (if any):

```
String.prototype.trim = function() {  
    return (this.replace(/^\s\xA0+/, "").replace(/\s\xA0+$/, ""));  
}
```

To use this function, you will use the following syntax:

```
var s = "    Hello World    ";  
var txt = s.trim(); // returns only "Hello World"
```

You can use this extensibility feature in Internet Explorer and Mozilla/FireFox to add new methods and (readonly) properties to the intrinsic objects, but there is no cross browser way to capture the access to properties of those objects.

You might think that it is also possible to add a global method *trim(s)* that uses a parameter and the return value to do the same thing, but you might get in conflict with other variables or methods from different libraries using the same name. You can also add a *trim* method to the *Array* object that handles array items, without getting a conflict.

Some general information about the *prototype* property can be found at

<http://msdn.microsoft.com/library/en-us/script56/html/js56jsproprototype.asp>

## JavaScript Prototypes in Mozilla/Firefox

The Mozilla/Firefox platform also offers a proprietary but very useful extension model that allows to also implementing properties on the JavaScript intrinsic objects. Here I added a *weight* property that returns the sum of the char codes of a string:

```
String.prototype.__defineGetter__("weight",
function () {
    var w = 0;
    for (var n = 0; n < this.length; n++)
        w += this.charCodeAt(n);
    return w;
});
```

You can use this property on any string:

```
var s = "ABC";
var n = s.weight; // will return 198
```

Remember:

The *\_\_defineGetter\_\_* and *\_\_defineSetter\_\_* are unique features to Mozilla/FireFox and you can find some (still few) samples on the internet by searching through Google for these keywords.

## Prototypes with HTML objects

JavaScript objects and HTML objects are completely different kinds of objects when using the Microsoft Internet Explorer but with Mozilla/FireFox, these objects share a common implementation. So, the feature we look at now is only available in Mozilla/FireFox.

The Internet Explorer, for example, supports a *innerText* property on many HTML objects like *span*, *div*, *p*, ... that can be used to safely access and set the text of an HTML object.

Mozilla/Firefox, in contrary, uses the *textContent* property that is part of the newer DOM Level 3 standard to do the same thing.

Now you can use the extensibility features of Mozilla/FireFox to emulate an *innerText* property on all HTML objects and you can eliminate all conditional scripting across your JavaScript programs:

```
var isIE = (window.navigator.userAgent.indexOf("MSIE") > 0);

if (! isIE) {
    HTMLElement.prototype.__defineGetter__("innerText",
function () { return(this.textContent); });
    HTMLElement.prototype.__defineSetter__("innerText",
function (txt) { this.textContent = txt; });
}
```

You just have to include this script in one of your common JavaScript include files.

The `~/controls/jcl.js` file implements some usable wrappers to Microsoft specific features and makes them available in Firefox too:

|                    |                                                                             |
|--------------------|-----------------------------------------------------------------------------|
| <i>innerText</i>   | Gets and Sets the inner text of HTML nodes.                                 |
| <i>XMLDocument</i> | Returns a <i>XMLDocument</i> object from the inner text that should be XML. |



|                                |                                                                        |
|--------------------------------|------------------------------------------------------------------------|
| <i>jcl.AttachEvent</i>         | Attach event handlers to HTML objects.                                 |
| <i>jcl.DetachEvent</i>         | Detach event handlers from HTML objects.                               |
| <i>window.event</i>            | Return the actual executing event object.                              |
| <i>event.srcElement</i>        | Return the target node of the executing event.                         |
| <i>event.cancelBubble</i>      | Stop the propagation of the current event.                             |
| <i>event.returnValue</i>       | Set the return value of the current event.                             |
| <i>xmlobj.selectSingleNode</i> | Return a single node from an XML element by using an XPath expression. |
| <i>xmlobj.text</i>             | Return the inner text of an XML element.                               |

## Firefox compatibility

When implementing JavaScript that's working with both browsers (IE and Firefox) you take care of the compatibility issues.

I fixed some minor things:

- The `parentElement` attribute of the HTML DOM should not be used. Use `parentNode` instead.
- The `firstChild` attribute might get a text node in Firefox that is skipped or not existing in IE. Whitespace should not be used between a parent node and the first child node.
- The `onscroll` event it is not documented but works as expected in Firefox. Great !
- Don't forget to test!

## Conclusion

If you ever thought that writing cross browser compatible JavaScript is hard to do, you now have a chance to build your layer of compatibility and make your script more readable and easier to maintain, and get rid of the conditional code fragments spread all over your web application.

This approach helps a lot but is not the definitive solution. A lot of people (including myself) hope that the current and upcoming standards will get implemented in all browsers. It's better to use these features, when available on multiple platforms. There are also good samples of proprietary Microsoft Internet Explorer features that are so useful that Mozilla/Firefox did implement them. The most popular sample to this is the *XMLHttpRequest* object that allows AJAX like cross browser implementations.

Some good reading URLs for browser compatibility hints:

# Building JavaScript enabled web controls

When using ASP.NET on the web server some things become easier because it brings a built-in framework for reusing HTML code fragments, building controls and components and managing some useful things around it. I will use some features of the page model, ASP.NET user controls and Web Controls here to make it easy building AJAX Controls.

Together with the JavaScript Behaviours that are used to build the client side the ASP.NET Controls on the server is a solid basis for building AJAX Controls.

There are good articles on the web that explain how to build this kind of controls and there are a lot of samples too. Most of these examples don't use client side functionality but rely on the usual reload mechanism by using forms. There are some tricky things around when building controls that need the specific JavaScript on the client too.

## User Controls

The very useful functionality of ASP.NET User Controls is the possibility of writing often needed HTML code only once in an \*.ascx file and reuse it as often as needed by writing only referencing tags into the ASP.NET pages.

User Controls can be found in the ~\controls folder and Web Controls can be found in the ~\App\_Code folder.

## Web Controls

Some things are hard to be done when using User Controls. Especially when the Control is only a framing element for more inner HTML objects User Controls cannot be used easily. In this case Web Controls are used to implement the specific tag that have to be implemented as special classes, are easier to be implemented so you can find both technologies in the samples.

When using Web Controls or User Controls in ASP.NET pages it is necessary to declare the namespace, tag name and implementation of any server tag that is used. When including a reference by dragging a control onto the page this directive is included at the top of the page. If you use the same tags multiple times in your web application you can also use the declaration mechanism that is available through the web.config file in the web application root folder. So if you do not find the declarations have a look at web.config.

## Delivering controls to the browser

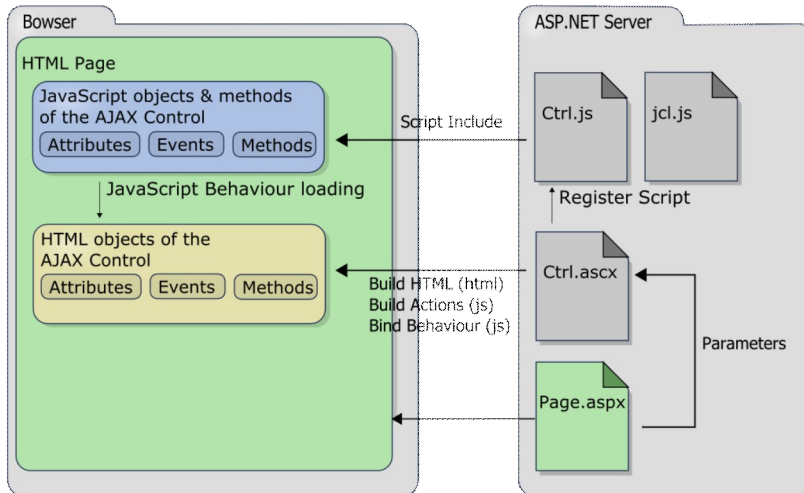
Delivering a JavaScript enabled control to the client consists of the following elements:

- The global JavaScript include file jcl.js must be present on the client. It must be included once by referencing to it using a <script> element in the header of each page.
- The JavaScript include file of the control that contains the control's Behaviour implementation must be present on the client. It must be included once only even if multiple controls of the same type are used on the same page.
- The control's html code has to be included for every instance. The parameters that are specified in the server side declaration have to be

# Building JavaScript enabled web controls

brought to the client for make them available to the Behaviour implementation.

- The JavaScript Behavior must be bound to the generated HTML element by calling the `jcl.LoadBehaviour` method once for every instance of the control



Of course it is possible to bring all these components to the client without using a web control but some server side logic implemented by the programmer who implements the control can help hiding these details from the programmer who uses the control.

## Registering Script includes

A very useful server-side functionality of ASP.NET Web Forms is the way how controls can control that a JavaScript include file is needed for a control to work successfully. The function "RegisterClientScriptBlock" that is available on the Page object as well as on the new Page.ClientScriptManager object in ASP.NET 2.0 can be used to specify some html code that will be included inside a HTML form element before all other HTML content.

With "IsClientScriptBlockRegistered" it is possible to check if a include file is already registered.

Using this a mechanism is perfect for building AJAX Controls because the web programmer needs not to know about the control specific JavaScript include files that must be included for a Control to work properly. He can just include the AJAX controls at the right place and everything else gets magically done.

```
Page.RegisterClientScriptBlock("CommonBehaviour",
    "<script type='text/javascript' src='"
    + Page.ResolveUrl("~/controls/jcl.js ")
    + "'><" + "/script>\n");
```

When having a HTML form element on the page the form element will automatically include all the submitted scripts before all other contained HTML content.

## Registering Script includes without a form element

Using this trick, it makes no difference whether a HTML form element exists in the page or not. This is important especially for AJAX applications because

If you want to use ASP.NET version 1.1 you need only include a HTML form element before all other HTML code to get the includes into the page. This doesn't mean that the AJAX Controls must be inside the form element, they also may be positioned after an empty form element.

# Building JavaScript enabled web controls

the avoid using any form functionality and get rid of reloading the whole page.

When using ASP.NET 2.0 you can use a little trick to get the HTML script tags for the include files added to the end of the HTML head element.

The <head> element must be marked with "runat=server" to make this trick work. If the "runat=server" is missing on the page the Page.Header object will be set to null and it is not possible to dynamically add new nested controls.

```
// register the JavaScripts includes without need for a HTML form.
if (!Page.ClientScript.IsClientScriptBlockRegistered(Page.GetType(),
    "CommonBehaviour")) {
    Page.ClientScript.RegisterClientScriptBlock(Page.GetType(),
        "CommonBehaviour", String.Empty);
    Page.Header.Controls.Add(new LiteralControl(
        "<script type='text/javascript' src='"
        + Page.ResolveUrl("~/controls/jcl.js ")
        + "'><" + "/script>\n"));
} // if
```

The RegisterClientScriptBlock function is called by using an empty string value. This makes it possible to detect within other controls, that the JavaScript file is already included. The first instance of a control is the one that effectively includes this marker and additionally includes a LiteralControl that contains the script file reference.

Because the jcl.js file is required by all the web controls it is important that the same type and name is use in every control. I use Page.GetType() and "CommonBehaviour" for jcl.js.

For the JavaScript files that contain the Behaviour implementations the controls type is used through this.GetType() and the name is always "MyBehaviour".

The script fragment that binds the Behaviour to the html element is included into directly just after rendering the html code of the control:

```
<script defer="defer" type="text/javascript">
    jcl.LoadBehaviour("<%=this.UniqueID %>", PropInputBehaviour);
</script>
```

## Parameters

Specifying parameters that control the specific functionality of the AJAX Control are part of the specific source code of a page by using attributes on the tag of the control:

```
<ajax:Lookup ... LookupService="OrteLookup" ... />
```

What you see here is NOT HTML code but a descriptive declaration of rendering some HTML code here. When using ASP.NET User Controls these attributes do not get automatically rendered as HTML attributes. Instead the ASP.NET framework matches them to properties of the class that builds the control on the server. So any attribute that is used as a parameter must also be defined as a field or property of the control's class to make the value available on the server.

```
<%@ Control Language="C#" ... %>
<script runat="server">
    public string lookupservice = "DefaultService.asmx";
    ...
</script>
```

To make it available on the client the values of these members must then be written out in the HTML code that is generated by this control:

# Building JavaScript enabled web controls

```
<input ... lookupservice="<%=this.lookupservice %>" ... />
```

The consequence of this is that the default-values for parameters that are not specified in the source code must be specified in the initialization of the class members and that values assigned to in the JavaScript prototype objects are always overridden.

## HTML Code

Writing specific HTML code for a User Control is simply done by writing it down at the end of the \*.ascx file. It can be as complex as you like it to be.

Be sure to also add the unique id of the control into the generated HTML code:

```
id="<%=this.UniqueID %>"
```

An ASP.NET User control doesn't automatically create an outer HTML object. It is also possible to generate multiple objects in a row. In this case the JavaScript behavior is attached to the object that is assigned the unique id.

If you need a reference to another web resource you can use the ResolveUrl method of the Page object:

```
src="<%=Page.ResolveUrl("~/controls/images/drop.gif") %>"
```

## Programming the Behaviour

The specific JavaScript behavior that should be used to implement the client-side functionality for a User Control should be implemented in a separate JavaScript include file. This is not strictly necessary but is good for the overall performance because it can be cached in the browser.

I use the same name as the \*.ascx file for this control specific include file and place them all into the ~/controls folder.

To attach the behavior to the html object a small JavaScript fragment is also part of the rendered HTML code:

```
<script defer="defer" type="text/javascript">
  jcl.LoadBehaviour("<%=this.UniqueID %>", LookUpBehaviour);
</script>
```

## Registering the script includes

Before the HTML text is send to the client all the JavaScript include files that are needed by the control must be registered on the page. This can be done in the OnPreRender method:

```
protected override void OnPreRender(EventArgs e) {
    base.OnPreRender(e);

    ...

    // register the JavaScripts includes without need for a Form.
    if (!Page.ClientScript.IsClientScriptBlockRegistered(Page.GetType(),
        "CommonBehaviour")) {
        Page.ClientScript.RegisterClientScriptBlock(Page.GetType(),
            "CommonBehaviour", String.Empty);
        ((HtmlHead)Page.Header).Controls.Add(new LiteralControl(
            "<script type='text/javascript' src='"
            + Page.ResolveUrl("~/controls/jcl.js")
            + "'><" + "/script>\n"));
    } // if

    if (!Page.ClientScript.IsClientScriptBlockRegistered(this.GetType(),
```

# Building JavaScript enabled web controls

```
"MyBehaviour")) {
    Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
        "MyBehaviour", String.Empty);
    ((HtmlHead)Page.Header).Controls.Add(new LiteralControl(
        "<script type='text/javascript' src='"
        + Page.ResolveUrl("~/controls/LookUp.js")
        + "'>" + "/script>\n"));
} // if
} // OnPreRender
```

All the code fragments can be found in many web controls implemented in the `~/controls/*.ascx` and the `~/App_Code/*.cs` files.

## Integration into ASP.NET

### No Form element

AJAX controls are built for NOT posting back, submitting the changes the user did to the server by using the HTML form element and reloading the whole page. That's why we want AJAX in our web applications. In contrary, ASP.NET controls are made for exact this kind of client-server interaction and therefore are most of the time placed inside a HTML form element. Because we do not use autopostback functionalities nor do we directly implement some script to call `form.submit()` we do not get problems with a possible existing HTML form element.

If you plan to extend your existing ASP.NET Web Forms with AJAX Controls – which is one of the scenarios I want to support – you will need this HTML form element. If you build new web pages without using the server side functionality of ASP.NET web forms you can eliminate this HTML form element completely.

### The Return key

One thing will disturb users when a form element is present: pressing the `<return>` or `<enter>` key in an input field is (like F5) reloading the page. All information that came to the page since the time it was loaded and displayed by using some asynchronous communications will be lost too.

Pressing return on input-fields outside a HTML form element normally has no visible effect because the return character will not be added to the value of the HTML input element.

There is a simple trick built into the `jcl.js` file that can help you to get around this situation. By capturing all keypress events and test if return was pressed, it is possible to cancel the keypress event early and prevent the automatic call of the submit functionality.

To enable this trick for the input fields they just must have an attribute `"nosubmit"` set to `"true"`. You can add this attribute to the HTML code or add it to the JavaScript behavior bound element too.

# Connecting Controls

Separating and encapsulating the functionality into components helps to build a reusable framework. But most components do not exist alone but must be linked to other elements.

## The classical approach

When using the standard built-in controls of ASP.NET you will use the events that are exposed on buttons, fields and the page and write small methods that response to these events. Inside these event handlers you reference to other objects directly because they are part of the page or control class or by using their id or at runtime using a search method.

These handlers are executed by using a submit() of the HTML form element - a post-back situation that should not happen in AJAX applications. We need a solution on the client side of the application in JavaScript.

You can also follow this approach on the client by writing event handlers and attaching them to the native HTML events of the HTML objects.

The drawback of these architecture is that when using complex controls the object that exposes this event might be an inner part of the control without an id or maybe not existing while the page is still loading.

If you want to follow this approach you can use `jcl.AttachEvent`. This is a method from `jcl.js` include file that you can use for this purpose and that works across the browser platforms and that allows multiple event registrations..

Even when you know all inner details of the actual control you might not want to use that knowledge because you will give away the chance to extend or re-implement a control without also changing all the implementations on the pages.

## The WebPart approach

With Sharepoint and the ASP.NET 2.0 WebPart Framework came another mechanism of connecting components by using a provider-subscription pattern. Components can register themselves for publishing (provider) or consuming (subscription) a specific property-value. Every time when the value of a property changes in a provider all controls that registered a subscription for this property are informed about that.

There are some advantages for this approach:

- It works even if the controls that build up a page are not known at compile-time. (Sharepoint)
- Multiple controls can easily publish the same property.
- Multiple controls can easily consume the same property.
- We do not need any browser- or object-specific events.
- The IDs of the objects must not be known.

You see that this is a higher level approach that fits better into the problem space of component based pages and portals and it can in addition be combined with the eventing model of the native HTML objects.

## The Page Properties mechanism

In the framework there is a lightweight mechanism implemented that is very similar to the WebPart approach and can be used to publish and subscribe to named properties that are available on the page level. All the methods of this mechanism are available through the `jcl.DataConnections` object.

All the available methods can be found in the reference section of this book.

A JavaScript Object can register itself as a provider, as a consumer or for both:

```
jcl.DataConnections.RegisterProvider(theObj, theName);
jcl.DataConnections.RegisterConsumer(theObj, theName);
```

To change the value of a property the `Raise` method can be used:

```
jcl.DataConnections.Raise(theName, theValue);
```

To receive a notification when a property has changed the `GetValue` method of the registered object is called. A simple implementation may look like this:

```
GetValue: function (propName, propValue) {
    this.value = propValue;
}, // GetValue
```

The names of the properties are always converted to lowercase characters so they should only be compared by after a `toLowerCase` conversion.

Based on this mechanism a solution for the “Back Button Problem” is also available by using the `PropHistory` control.

## The Connection Test Sample

This sample uses 3 kinds of controls implemented with JavaScript Behaviours to show (and test) the implementation and usage of the client side controls connections. There are 3 properties used in this sample (x, y and z) that can be modified by 2 different Controls and that are visualized by a simple bar chart.

[http://www.mathertel.de/AJAXEngine/S03\\_AJAXControls/ConnectionsTestPage.aspx](http://www.mathertel.de/AJAXEngine/S03_AJAXControls/ConnectionsTestPage.aspx)

## Simple Controls using page properties

### PropInput Control

This control renders a HTML input element that is extended by a JavaScript Behaviour to raises a change of its value to the property that is specified by the "name" attribute. It is used by implementing:

```
<ajax:PropInput runat="server" name="x" />
```

This control also registers itself as a consumer to display the actual value when changed by another control.



### PropSlider

This control implements is a horizontal moveable rectangle that acts as a slider. It can be used to change a property value that is specified by the "name" attribute in the range from 0 to 100.

This control also registers itself as a consumer to display the actual value when changed by another control.

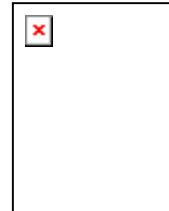


# AJAX enabled web controls

```
<ajax:PropHSlider ID="PropHSlider1" runat="server" name="x" />
```

## PropBarChart

This control implements a simple bar chart to display multiple values. The names of the properties are displayed below the bars. The chart itself is implemented by using a table layout with some inner <div> elements with a blue background color.



The names of the values can be specified by the "properties" attribute by using a semicolon separated list of names.

The max displayable number can be specified by the "maxvalue" attribute. This value is used as a scale factor for the display.

```
<ajax:PropBarChart ID="barchart" runat="server" properties="x;y;z" maxvalue="100" />
```

## PropEventLog

This control logs every change of any property and lists the new value on the page inside a region. When developing a new control that uses page properties, this little log can help a lot.

**DataConnections Property changes:**

- z:60
- z:62
- z:64
- z:66
- z:67
- z:68
- z:69
- z:70
- z:71
- z:72
- z:73
- z:74
- z:75

## AJAX enabled web controls

## The Back Button Problem of AJAX applications

A lot of people already wrote about this topic. Search "[AJAX BACK BUTTON PROBLEM](#)" on a search portal and you can find a lot of stuff about it!

But there are still aspects that haven't been talked about a lot.

Let me explain that the problem is old, well known, not an AJAX specific problem and after all: technically solved.

### What the back button does - and why not

Still some people think that the web is a mass of pages pointing to each other using hyperlinks. While navigating along these hyperlinks every visited page is recorded into a local list often called "history". The original idea was to support this list of pages with some functionality.

When framesets came into the browser model this simple idea got the first scratch because when you use hyperlinks that change only the content of a frame the url doesn't change. The url that you can see in the address is not the only information you need to restore a browser situation.

Up to here the back button still works in the browsers, because they also record this extra information together with the url - at least when working with the browser history functionality.

Then there are web servers that found a (not so) wonderful trick to use forms for navigating from one page to another and things get complicated. Hyperlinks can be used to submit the content of a form (including the current browser state) back to the server which then returns the next or the modified page. A problem on this approach is that the same url (main.do, main.aspx, main.jsp, main.php, ...) can be reused for the whole application.

Here the back button also has to record all the data that was posted to the server so it can be re-posted when the user wants to go one or more pages back.

This kind of request cannot in generally be distinguished by the server from clicking the same button of a form twice and often there are filters available on the server to redirect these requests to another page.

And this is why navigating back using the back button of the browser often doesn't work.

And this is why many applications start a new window at the beginning without displaying the navigation bar that includes the back button.

And this is why many users like the back button on the mouse or keyboard (ALT-LEFT).

And this is why the web doesn't work like this (too often).

### The favorite problem

As with the history list a very similar problem exists with the favorites and with web urls stored in files. When you save a current page to your favorites list you might think that this helps you coming back to this page but if the server needs post information or the page is built using frames you might

become disappointed because this kind of information is not stored in hyperlinks.

The IE implements a special \*.url file type that is created when using the IE built in "Add to favorites..." command that contains extra information about the current urls of the frames and maybe more.

But the \*.lnk file type that is created when dragging an URL from the IE address bar to the desktop as well as the url file that is created by Mozilla/Firefox doesn't hold this information.

What's the problem?

The problem with the back button existed long before AJAX applications became widely known. The core of the problem is about building a web application that works by using meaningful urls whether using frames or not and enabling the usage of the history and favorites feature of the existing browsers.

AJAX applications do have the same problem like those applications doing everything through a single url: it's meaningless and you cannot rely on any posted information because it just doesn't exist.

So what we all have to do is giving our applications meaningful urls.

## Meaningful urls

Not every click should change the url and in many applications there is no clear "back" situation. If you browse your mail inbox for example a click to open a message may be a clear "navigate to" situation. When you delete the message by using a click on a button the main UI might navigate for you to the next message in the list. But where is the "back" position now - on the deleted mail?

You can also argue that moving on to the next mail is neither forward nor back but something like up and down and back will always bring you back to the list view you started.

So what I want to say is that the (forward and) back direction is often very application or problem specific. With AJAX you have the chance to model these situations the way you like and the way that is the best for your application and you are not bound to the GET and POST requests done by hyperlinks and forms.

I use the url as a textual key to the global addressable state of an Ajax application. By moving some information into the url and reading this information out of the url when a page loads or the urls changes, there is a clear interface between the Ajax page and the external world that can be used for storing browser history, favorites and links.

The back problem is technically solved

If you look for a good article on how to implement a technical solution to the problem have a look at the article "AJAX: How to Handle Bookmarks and Back Buttons" by Brad Neuberg

<http://www.onjava.com/lpt/a/6293>.

... by tricking IE

The Mozilla/Firefox solves this problem really straight forward and it works as expected.

# AJAX enabled web controls

When using the method `window.location.replace(newHash)` only the current URL is changed.

By assigning a new value to the `window.location.hash` property a new entry in the history list is created.

The IE is very confused about URL-hash values that do not really point to existing anchors and does not record any history and needs an invisible iframe with a page coming from the same server. It's a little bit tricky to get all situations working and you can see the current implementation by looking into the JavaScript implementation of the PropHistory Behaviour. Have a look at the source:

[http://www.mathertel.de/AJAXEngine/ViewSrc.aspx?file=~/\\_controls/PropHistory.js](http://www.mathertel.de/AJAXEngine/ViewSrc.aspx?file=~/_controls/PropHistory.js)

## An Implementation

The AJAXEngine already has a client side state mechanism (Page Properties or DataConnections) I introduced in September 2005 that stores key-value pairs. Instead of building a new API and leave it to the AJAX application to call a method to change the URL you only have to declare what page properties should be used in the URL. When a Page Property's value changes the URL is updated automatically and when the URL changes all connected controls will get informed about it.

When one of the properties, that define a state that should be persisted to the history changes, a new history entry is recorded.

All you need to do to make it work is to include the PropHistory control:

```
<ajax:PropHistory runat="server" stateList="version,book,chapter" propList="vers" />
```

I've added 2 attributes to the PropHistory web control that can be declared to avoid some scripting on the page:

`stateList` specifies the page properties that together form the current state of the page. A change to the value of any of these properties will cause a new entry in the history of the browser.

`propList` specifies the page properties that are persisted in the hash part of the url so that hyperlinks will work.

The implementation takes care of the fact that all properties in the `stateList` must also be part of the url but you don't have to specify them again in the `propList` attribute.

If you need some default-values that will be used when no values are given by the URL then script them like this:

```
<script defer="defer" type="text/javascript">
// set default values:
jcl.DataConnections.Load("version", "luther1912");
jcl.DataConnections.Load("book", "1");
jcl.DataConnections.Load("chapter", "1");
jcl.DataConnections.Load("vers", "1");
</script>
```

The sample comes from the Bible reader application:

[http://www.mathertel.de/AJAXEngine/S03\\_AJAXControls/BiblePage.aspx](http://www.mathertel.de/AJAXEngine/S03_AJAXControls/BiblePage.aspx)

When you navigate through the verses of a chapter, only the URL gets changed. The navigation to a new book, version or chapter a new entry in the history stack is created.

## AJAX enabled web controls

You now also can use URLs to jump directly to some important content.  
Donald E. Knuth likes reading any book chapter 3 verse 16 and specially

☐ [http://www.mathertel.de/AJAXEngine/S03\\_AJAXControls/BiblePage.aspx#version=sf\\_kjv\\_strongs\\_rev1b&book=43&chapter=3&vers=16](http://www.mathertel.de/AJAXEngine/S03_AJAXControls/BiblePage.aspx#version=sf_kjv_strongs_rev1b&book=43&chapter=3&vers=16)

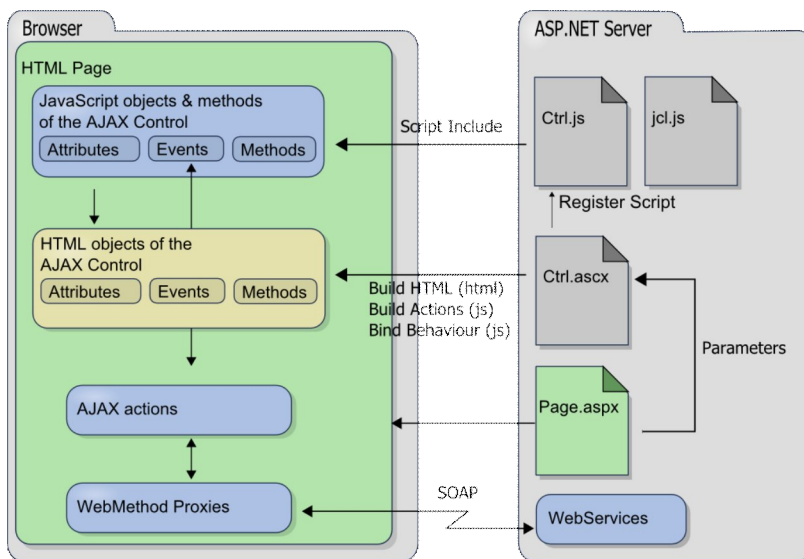
An important verse of the Da Vinci code story by Dan Brown is

☐ [http://www.mathertel.de/AJAXEngine/S03\\_AJAXControls/BiblePage.aspx#version=en\\_gb\\_KJV2000\\_1&book=18&chapter=38&vers=11](http://www.mathertel.de/AJAXEngine/S03_AJAXControls/BiblePage.aspx#version=en_gb_KJV2000_1&book=18&chapter=38&vers=11)

# AJAX enabled web controls

Writing an AJAX enabled control (here AJAX Control) is as easy as writing another AJAX enabled web applications. The only difference lies in the kind of separating HTML, JavaScript and the clueing stuff like AJAX actions and server calls into the right places so that the control can be reused in other places.

If you just want to separate a part of your application so that other members of the team can work on it separately then you can place all the code of the control into the User Control's \*.ascx file.



## AJAX Actions inside JavaScript Behaviours

The AJAX Action that is used by the Control can be declared as a part of the Behaviour's prototype object. In the LookUp Control you can find the `_fillAction` property that is the declaration object of the AJAX action.

```
// declare an AJAX action to the lookup service
_fillAction: {
  delay: 100,
  prepare: function(fld) { fld.savevalue = fld.value; return (fld.value); },
  call: "proxies.ServerInfo.GetDetails",
  finish: function (val, fld) {
    if (fld.savevalue != fld.value) {
      ajax.Start(fld._fillAction, fld); // again
    } else {
      var dd = fld.CreateDropdown(fld);
      fld.FillDropdown(dd, val);
    } // if
  }, // finish
  onException: proxies.alertException
}, // _fillAction
```

If you only have one single instance of the Control on the page you will not get into trouble. If there are multiple instances of the Control on the same page these actions definitions will be shared by all controls so you better make a copy of it using the `jcl.CloneObject` function because they will use different WebServices.

# AJAX enabled web controls

The reference to the `WebService` should also be set in the `init` function after the page was loaded because it is not guaranteed whether the `WebService` proxies are already setup correctly when the behavior is attached.

```
this._fillAction = jcl.CloneObject(this._fillAction);  
this._fillAction.call = proxies[this.lookupService].GetPrefixedEntries;
```

Because the action object and the finish function are both declared inside the `Behaviour` element the finish method cannot be declared/referenced directly because the outer `JavaScript` object is not known at this time.

The `AJAX` engine has a workaround for this case. You can just use a string instead of a function pointer for the `call`, `prepare` and `finish` references. These strings are evaluated and will be replaced by the function reference when the action is started for the first time.

## Using AJAX enabled controls in ASP.NET Forms

The use of web forms is for many web applications the most important functionality. They rely very much on it because it was implemented by all old browsers they also the modern ones still provide this functionality.

Using `AJAX` functionalities in your web applications doesn't mean that everything needs to be re-implemented. With the `AJAX` functionality these applications can easily be extended with controls using asynchronous functionalities in the background without breaking up the existing applications structure:

- Validate values immediately on the server after the user exits the field without reloading the form.
- Help the user to find the right value by extending a field with a specific lookup function.
- Show or hide a field, depending of already entered values.
- Fill fields in the form, depending of already entered values.
- Fill the `OPTIONS` of `HTML SELECT` elements depending of already entered values.

Common to these scenarios is that a pure client-side implementation is often hard to do or impossible because some data is required and this data is not available on the client directly. On the server in contrary, the information is available and is in classical `Web Forms` used to validate the input of the user before accepting and executing the functionality.

If you integrate `AJAX` controls into your existing web applications it is possible to make your forms more responsive and give faster feedback and help to the user. The processing of the `Web Form` can stay as it is by submitting the data of the form to the server and reloading or redirecting the page after the server side processing is done.



## The Samples

### Validator Demo

[http://www.mathertel.de/AjaxEngine/S03\\_AJAXControls/ValidatorDemo.aspx](http://www.mathertel.de/AjaxEngine/S03_AJAXControls/ValidatorDemo.aspx)

Name:  This sample shows how to validate the actual input of field values in a form while the user enters the data into the fields. Here an email address is validated by using a server-side DNS lookup that checks whether the host part of an email address is known on the web.

E-Mail:

Errors on the form:  
Please enter a valid e-mail

### Table Data

[http://www.mathertel.de/AjaxEngine/S03\\_AJAXControls/TableData.aspx](http://www.mathertel.de/AjaxEngine/S03_AJAXControls/TableData.aspx)

This sample uses several AJAX enabled controls to load tabular data from the server. The data for visible rows is automatically loaded when scrolling down the table or by selecting another page of the available data.

This technique enables using huge data tables on pages without using complete server round trips.

### Cascading select elements ("Bible Reader")

[http://www.mathertel.de/AjaxEngine/S03\\_AJAXControls/BiblePage.aspx](http://www.mathertel.de/AjaxEngine/S03_AJAXControls/BiblePage.aspx)

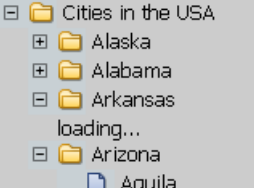
1. Select a version:  2. Select a book:  3. Select a chapter:  4. Select a vers:

Cascading Select elements is a standard situation where AJAX technology can decrease the amount of HTML code when loading the page by also restricting the possible input to the valid combinations

This sample shows how to populate OPTION elements of a SELECT element dependent of other values without reloading the whole form.

### Tree View AJAX Control

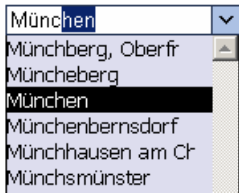
[http://www.mathertel.de/AjaxEngine/S03\\_AJAXControls/TreeView.aspx](http://www.mathertel.de/AjaxEngine/S03_AJAXControls/TreeView.aspx)

 This AJAX enabled control shows a tree of data. The only thing that you need to implement is a Webservice that is called for retrieving the sub nodes of a given node.

# AJAX enabled web controls

## AJAX LookUp Element

[http://www.mathertel.de/AJAXEngine/S03\\_AJAXControls/OrteLookUp.aspx](http://www.mathertel.de/AJAXEngine/S03_AJAXControls/OrteLookUp.aspx)



A web control that suggest input values by completing the already typed text.

The suggestion is implemented by using a huge server side list of many cities in Germany – a list that is definitively too large to be downloaded to clients and used in Select Option elements.

## AJAX PopUp Element

[http://www.mathertel.de/AJAXEngine/S03\\_AJAXControls/AJAXPopUpDemo.aspx](http://www.mathertel.de/AJAXEngine/S03_AJAXControls/AJAXPopUpDemo.aspx)  
[http://www.mathertel.de/AjaxEngine/S03\\_AJAXControls/TreeView.aspx](http://www.mathertel.de/AjaxEngine/S03_AJAXControls/TreeView.aspx)



This sample see shows how additional information can be retrieved from the server and displayed in an html popup.

This implementation is enabling a popup on hyperlinks and is displaying some information about the server.

## Custom Validation AJAX Control Sample

The ASP.NET Web Forms offer a collection of validation controls out of the box to check the values in the fields of a form. The validation process takes place in 2 situations:

- When the form data is sent back to the server a specific server-side functionality of the control or the page is called.
- When the user changes the value of a field and leaves the field a specific client-side function is called.

Now, with the help of AJAX and the possibility of asynchronous calls to the server in the background a third kind of validation can be implemented without great effort and combines the power of the server side processing with the better user experience of the immediate verification in the client.

- When the user changes the value of a field and leaves the field a specific WebService can check the value.

The implementation is straight forward. We use the built-in CustomValidator control to attach a client side function:

```
<input autocomplete="off" id="EMAIL_IN" runat="server" name="EMAIL" />
<asp:CustomValidator ID="CustomValidator1" runat="server"
  ControlToValidate="EMAIL_IN"
  ErrorMessage="Please enter a valid e-mail address."
  ClientValidationFunction="validateEMail">*</asp:CustomValidator>
```

The client-side validation function that is used by the control as specified by the ClientValidationFunction attribute always leaves the arguments.isvalid flag set on true (default) to hide any shown error and then start the Ajax action with the source object as parameter.

```
function validateEMail(source, arguments) {
  ajax.Start(validateAction, source);
} // validateEMail
```

The AJAX action needs to be declared and is based on some implementation details of the client-side validation mechanism:

```
var validateAction = {
  delay: 0,
  prepare: function (source) {
    // from the Validator Common JavaScript
    return(ValidatorGetValue(source.controltovalidate));
  },
  call: proxies.ValidatorDemo.ValidateEMail,

  finish: function (val, source) {
    // from the Validator Common JavaScript
    source.isvalid = val;
    if (! val)
      Page_IsValid = false;
    ValidatorUpdateDisplay(source);
    ValidationSummaryOnSubmit(null);
  },
  onException: proxies.alertException
} // validateAction
```

There is nothing data-verification specific here except the link to the WebService we use to validate e-mail addresses so the declaration can be reused for different purpose.

The integration of the Ajax action is a little bit tricky as you can see because the validation process of the built-in validation controls of ASP.NET do not

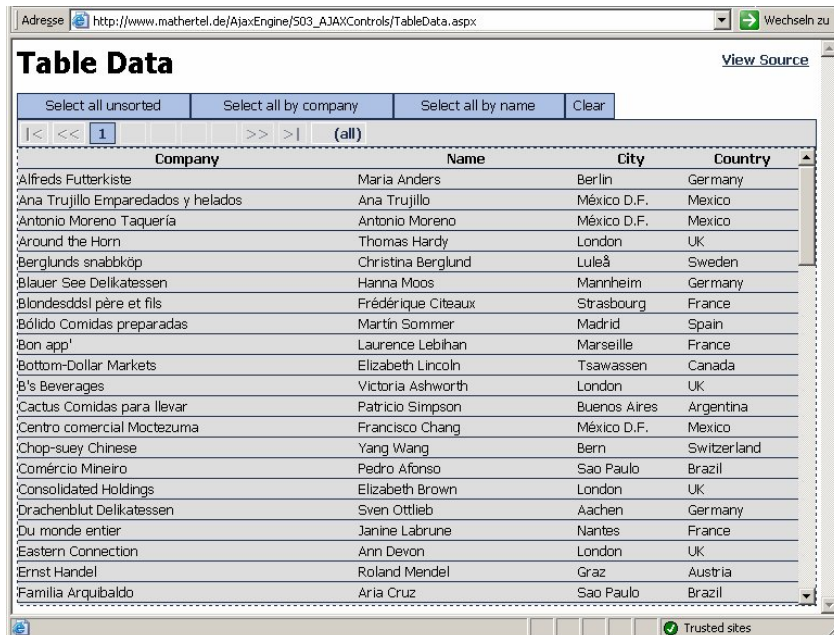
## AJAX enabled web controls

expect any asynchronous processing but wants an immediate decision to be taken and the Validation Summary sometimes does not update correctly.

The WebService itself can do what ever it wants to do. In my sample I use the DNS class to resolve the domain name and check if it exists. That cannot be done by the client and is a sample for the power you have on the server available, but not on the client on its own.

[http://www.mathertel.de/AjaxEngine/S03\\_AJAXControls/ValidatorDemo.aspx](http://www.mathertel.de/AjaxEngine/S03_AJAXControls/ValidatorDemo.aspx)

## Displaying huge tables using AJAX



<http://www.mathertel.de/AjaxEngine/S03 AJAXControls/TableData.aspx>

There are many implementations available in the ASP.NET framework to display tabular data. Most of them send the content of the table to the client as a part of the page including all the other parts of the page. If it comes to mass data situations the implementations offer a mechanism to navigate through pages containing a smaller set of the records and sending the whole page again and again.

Here comes another approach that offers paging and scrolling even with huge datasets without refreshing the page that uses AJAX calls to a WebService to fetch the data.

If you have followed this documentation you can easily identify the AJAX part of the sample. Most of the JavaScript you can see on the client is written to handle all the events and the HTML DOM operations.

In fact it's a 3 steps approach:

1. When the page loads an empty table that acts as the template is downloaded to the client. No data integration is needed for this phase to finish.
2. Then a query is started by passing some parameters to the WebService that selects all the rows that should be displayed and returns a unique ID for each row.
3. Then a row gets visible the data of this row is fetched from the server using another WebService call that returns a XML document for each row.

The amount of data that is transferred has some overhead in the case of the first page that is displayed but if you scroll down to the next page by using the scrollbars or the page navigation buttons, only the new row data is fetched from the server - far less than a whole page refresh. It gets better when you scroll back to a region or page that was already displayed because the data is already on the client and can be displayed without requesting the server at all.

Also when you request another set of records by using another sort criteria all the rows that are already known on the client can they can be displayed without fetching them again from the server.

As you can imagine, using this approach it is possible to send huge tables to a client without the need to send all record data immediately.

#### Walk-Through

First select some data using one of the 3 Select buttons. You can now page through the recordset or (by favorite) press the all-button and scroll through the table. You can see the lazy loading of the table rows.

The implementation consists of a WebService, 2 Web Controls and the AJAX Engine that does the asynchronous communication and the caching.

### The DataTablePager Control

The DataTablePager is the client side controller of the MVC pattern and is an AJAX Control that implements the AJAX action for retrieving the dataset on the client side. This control acts together with additional buttons as the client-side controller of the whole implementation.

There are some methods available that can be used by additional buttons to load a specific view or to clear the actual data.

Buttons can use these functions to select by supplying a filter and a sorting criteria.

The events that are triggered by scrolling the table also use methods of this control to fetch data from the server.

### The DataTable Control

The DataTable control is the client side view of the MVC pattern and is responsible for building up a table that displays the record data fetched from the server formatted by the format given by the template row.

When the page load the template gets saved and when new rows are added this template is copied for every row that should be displayed.

Right now there is only a very simple template processing functionality implemented that can only display columns as simple text and the template is constructed on the server by interpreting the cols attribute of the datatable tag.

Every time a row gets visible a fetching the data is initiated by calling the DataTablePager Control's FetchRow Method.

When data comes back from the server the AJAX Action will call the DataTable Control's FillRow method to fill up the corresponding row with the record data.

### The TableData WebService

The WebService that brings the data and the functionality is located at:

[http://www.mathertel.de/AjaxEngine/S03\\_AJAXControls/TableData.asmx](http://www.mathertel.de/AjaxEngine/S03_AJAXControls/TableData.asmx).

The WebService that is used by the AJAX Actions of the DataTablePager Control is the server side model of the implementation.

To bind all 3 elements together only some lines of coding are needed and the sample page itself, containing the controls and the WebService's proxy, has only some specific attributes that must be set. – Have a look.

## Tuning the TableData

After realizing the first working version of TableData I found several things that did not work as expected so I published a better version with some enhancements.

### Speed

The first, non optimized version is still available at

[http://www.mathertel.de/AjaxEngine/S03\\_AJAXControls/TableData1.aspx](http://www.mathertel.de/AjaxEngine/S03_AJAXControls/TableData1.aspx)

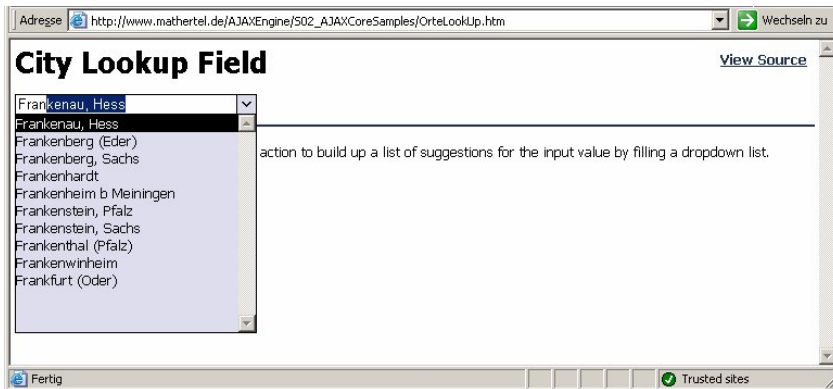
It retrieved the row from the Webservice one row each time. The time that is needed to update a page full of visible rows was only a second or two when developing on my local system but is about 4-6 second on the real internet. The reason for this slow down is the time a package needs to be transferred from the client to the server and back. You can use the tracert command (tracert http://www.mathertel.de) line tool to verify this timing. You can see how long an almost empty package needs to be transferred and that is extremely longer than a package that doesn't leave a development machine.

Reducing the number of packages is the key and I decided to bundle the Fetch call for all newly visible rows together into a single call to the server.

### Reusability

When trying to reuse the controls on a different project I found that I need more parameters to make them portable. The name (alias) of the service that is used for selecting and fetching the data is now configurable.

## An AJAX enabled bible reader



[http://www.mathertel.de/AJAXEngine/S03\\_AJAXControls/BiblePage.aspx](http://www.mathertel.de/AJAXEngine/S03_AJAXControls/BiblePage.aspx)

The sample is already an AJAX application and more than just a small technology driven demo page that shows how to use a specific control or piece of code.

I started working on this application to show how it all works together and to see where we need common functionality and what features we need within the controls.

### A Walk-Through

The reader can be started just by opening a window with the given URL. The page loads completely and starts with one of the available versions (of course) on the first book, first chapter, first verse: "In the beginning...".

You can select one of the server-side installed bibles by using the first select box:

- sf\_kjv\_strongs\_rev1b: An English version of the bible.
- en\_gb\_KJV2000\_1: A more modern English version of the bible.
- luther1912: A German version of the bible.
- biblehebr: A Hebrew version of the bible.

Yes, its right to left - Does it look right?  
I see some differences between FireFox and IE.  
Maybe anyone can help on that topic!

In the second select box you can choose one of the available book. Some bibles provide names for the books, others don't and you have to use the given numbers. The New Testament starts with book 40.

In the third select box you can choose the chapter you want to read and in the fourth select box you can finally choose the verse.

As you change the values of these select boxes you will see that the select lists will be updated automatically so that only valid entries are available. If you choose a book, the list of chapters will be updated, if you choose a chapter, the list of verses will be updated...

In the big text box below the text of the selected verse will be shown.



# AJAX enabled web controls

It is possible to change the version without changing the selected book, chapter and verse so you can easily switch between German and English or different English versions.

Below the text boxes are 2 buttons that enables the forward and backward navigation cross all book and chapter boundaries in a given version.

You navigate across multiple 5 MByte sized books. Fast!

## Technology

### Cascading Select Elements

The 4 select boxes at the top of the page are dynamically linked together by raising and listening to some page-level data connections. Together they form the cascading 4-level specification of the available verses.

Here, the AJAX Select box control is used together with an AJAX action and some JavaScript code. The chapter select box is defined by

```
<ajax:Select id="ChapterSelect" runat="server" pageproperty="chapter" style="width: 120px">
</ajax:Select>
```

and the JavaScript code just below it reacts on changes of the version or the book. The AJAX Action is used to fetch all the Chapters from the server by using the BibleData WebService and the ListChapters method.

```
<script defer="defer" type="text/javascript">
  var obj = document.getElementById("ChapterSelect");
  obj.GetValue = function(prop, value) {
    if ((prop == "version") || (prop == "book")) && (jcl.DataConnections.GetPropValue(prop) !=
value)) {
      ajax.Start(this.ChapterAction, this);
    }
  }; // GetValue
  jcl.DataConnections.RegisterConsumer(obj, "*");

  obj.ChapterAction = {
    delay: 10,
    prepare: function(obj) {
      key = jcl.DataConnections.GetPropValue("version")
        + ";" + jcl.DataConnections.GetPropValue("book");
      return (key); },
    call: proxies.BibleData.ListChapters,
    finish: function (value, obj) {
      obj.CreateOptions(value);
      if (obj.selLast == true)
        obj.options[obj.options.length-1].selected = true;
      obj.selLast = false;
    },
    onException: proxies.alertException
  }; // ChapterAction
</script>
```

Very similar scripts can be found for the other 2 cascaded select boxes.

### Retrieving the Vers Text

To fetch the text of a verse from the server, another AJAX action, named VersTextAction is used. The 4 properties version, book, chapter and verse are retrieved and passed to the BibleData WebService and the GetVers function. The returned text is then written into the text area.

```
obj.VersTextAction = {
  delay: 10,
  prepare:
  function(obj) {
    key = jcl.DataConnections.GetPropValue("version")
      + ";" + jcl.DataConnections.GetPropValue("book")
```

## AJAX enabled web controls

```
+ ";" + jcl.DataConnections.GetPropValue("chapter")
+ ";" + jcl.DataConnections.GetPropValue("vers");
return (key);
}, // prepare
call:
  proxies.BibleData.GetVers,
finish:
  function (value, obj) {
    obj.style.direction = ((jcl.DataConnections.GetPropValue("version") == "biblehebr") ? "rtl"
: "ltr");
    obj.innerHTML = value;
  },
onException:
  proxies.alertException
}; // VersTextAction
```

A very similar AJAX Action is used to look for a Prolog Text that might be describing the current book. The rest of JavaScript programming is for the sequential navigation through the verses, chapters and books.

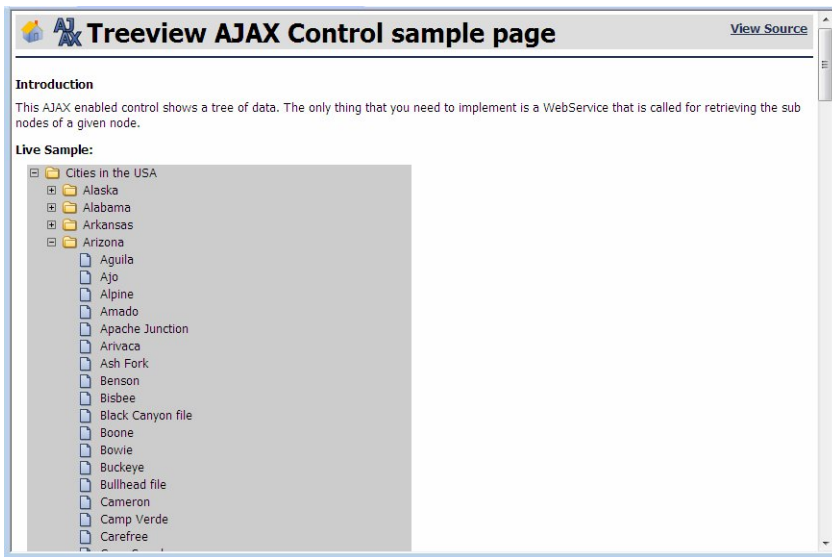
On the Server an ASP.NET Webservice, written in C# with 5 methods is used to deliver all the data that is needed by the Bible Reader Web page.

[http://www.mathertel.de/AJAXEngine/S03\\_AJAXControls/BibleData.asmx](http://www.mathertel.de/AJAXEngine/S03_AJAXControls/BibleData.asmx)

The entire source you need for this application has only about 15 kByte of text. You might not get any application of this functionality with less coding.

Programming AJAX Web Applications can be easy and without a lot of source code if you choose the right technology and level of programming.

## Treeview AJAX Control



[http://www.mathertel.de/AjaxEngine/S03\\_AJAXControls/TreeView.aspx](http://www.mathertel.de/AjaxEngine/S03_AJAXControls/TreeView.aspx)

Hierarchically structured data can be found in various places on this world. File systems, network structures, dependencies, organization charts, bill of materials etc. are hierarchical by nature or by design or can be viewed to be so.

(I personally think that the reason behind the fact that so many structures are hierarchically organized is that more complex structures are too chaotic to be understood easily.)

Displaying hierarchical data as trees is relatively easy to be implemented and you can find many implementations with ASP.NET Controls on the web. But when it comes to mass data situations (like the content of a file system) it is almost impossible to load the complete tree at. A simple paging approach, that is known from huge table-data doesn't fit for structured data.

Displaying this kind of data of a good place for using the AJAX technology and the architecture of my AJAX controls is also used here. The best solution is to load and open only the root level of the tree when loading the page and to load all sub-levels when the user wants to dig down by clicking a folder. Already loaded sub-levels can stay at the client until the page gets reloaded.

The AJAX engine can be used to solve the server call in the background very easily by using a Webservice that can be queried for sub-level data. With JavaScript and the XMLHttpRequest object this data can be requested from the server. The returned xml is then transformed into html code by using a XSLT transformation to display all the new found nested items.

### 1. Defining a contract

The AJAX control, after being loaded is using a Webservice for retrieving the list of sub-nodes (folders and files) from a given node (the user clicks). A Webservice that can be used for this purpose has a very simple interface with only a single method. The parameter is used with a key that uniquely defines a already known folder node and the return value is a structure with the folders and files under that node.

## AJAX enabled web controls

The unique key of a node can be described by a path string and the return value is typed as a XML document:

```
[WebMethod]
public XmlDocument GetSubNodes(string path);
```

Every folder and every file must have a name that must be unique among the sub nodes of a folder and we can concatenate these names as we usually do for file systems to get an overall unique identifying string.

The returned XML has the form:

```
<folder>
  <folder name="[name of the folder]" title="[some more text]" />
  <file name="[name of a file]" title="[some more text]" />
</folder>
```

There can be as multiple files and folders in any order.

## 2. Implementing a Webservice

The Webservice implementation we need is very simple. I've published a version that loads a XML file and uses this data source.

## 3. Javascript and HTML: building a tree layout

There are many <table> - based sample implementations for tree structures on the web. I use a very simple approach here that works fine but supports the indenting of sub nodes, but no lines in front of the icons that uses <div>-elements. Here is the basic structure of a folder with the container of the sub nodes:

```
<div class="du" name="foldername"><span class="ft">foldertitle</span></div>
<div class="subframe"></div>
```

The click event is captured at the root level and has to do the following things:

- identify the clicked folder object (<div> with class=[do|dc|de|du])
- hiding (do) or showing (dc) the container with the subnodes
- or starting an AJAX action (du) that loads the subnodes

## 4. The AJAX action

The context parameter of the only Ajax action we need is the node of the folder with the still unknown sub nodes.

The prepare function builds the path string for this folder.

The asynchronous call will return the xml document that is passed to the finish method.

In the finish method we transform the returned xml document into the html we need by using a simple xslt transformation that can be found inline in the file of the page.

All we have to code for that is an AJAX Action:

```
// Retrieve the sub-nodes of a given folder.
ExploreAction: {
  delay: 10,
  queueMultiple: true,

  prepare:
  function(src) {
    var path = "";
    var root = jcl.FindBehaviourElement(src, TreeViewBehaviour);
    while ((src != null) && (src != root)) {
      if (src.className == "subframe") {
        src = src.previousSibling;
      } else if (src.className == "do") {
```

## AJAX enabled web controls

```

        path = "/" + src.name + path;
        src = src.parentNode;
    }
}
while (path.substr(0,2) == "//")
    path = path.substr(1);
return (path);
},

call: "proxies.TreeView.GetSubNodes",

finish:
function(data, src) {
    jcl.FindBehaviourElement(src, TreeViewBehaviour).ExtendTree(src, data);
},

onException: proxies.alertException
}, // FetchAction

```

That's still a lot of code, so we build a Control to get that coding off from our daily work by implementing a control.

### 5. The AJAX control

All the coding up to here should be covered by the AJAX Control so the developer only adds this control with the following attributes:

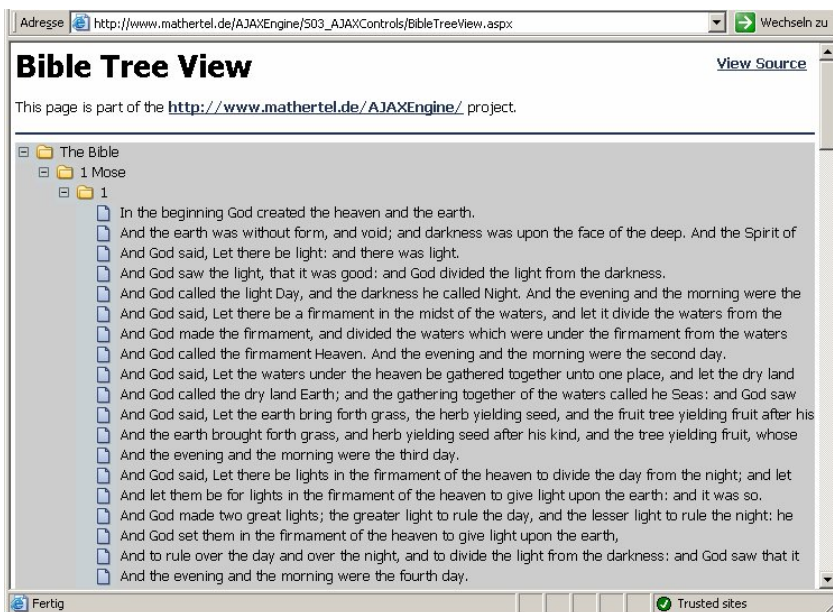
```
<ajax:TreeView runat="server" title="Cities of USA" service="WebServiceAlias" />
```

The *title* attribute is the label shown on the root folder.

The *service* attribute is the alias name of the web service that serves the data of the tree

The complete implementation can be found in the 2 files that build the AJAX Control and the JavaScript behavior: *TreeView.ascx* and *TreeView.js* that can be found in the controls folder.

## An AJAX based Tree View for the Bible



I just published another Page displaying an English Bible using an AJAX Tree View Control. It just took me half an hour, including some other housekeeping.

## AJAX enabled web controls

Have a Look at

[http://www.mathertel.de/AJAXEngine/S03\\_AJAXControls/BibleTreeView.aspx](http://www.mathertel.de/AJAXEngine/S03_AJAXControls/BibleTreeView.aspx)

# Visual Effects Library

---

## Why AJAX needs visual effects

Implementing a web based application by using AJAX techniques is about improving the backside, will enable the right user experience and will make your application work the way it should.

If you want that your application really looks like AJAX you also have to give your application a modern design together with all kinds of visual effects. That will really impress also the non-technical users!

Don't you think that this is nonsense? - I did for some time last year but I had to change my mind - at least a little bit.

If you travel across the web sides of the commercial AJAX frameworks you will find on almost every site a bunch of visual effect libraries that are technically not bound to asynchronous JavaScript and XML nor xmlhttp.

Commercial frameworks like backbase or frameworks like script.aculo.us do offer visual effects side by side with AJAX functionality.

Some of the effects shown here are really stupid eye-catcher effects.

Web applications that do not use xmlhttp but do asynchronous image loading are seen as AJAX applications.

Articles about AJAX do mix up client-server communications with visual effects. A sample on that:

<http://www.maxkiesler.com/index.php/weblog/comments/451/>

Another article: AJAX Rounded Corner Tutorials at <http://encytmedia.com/blog/articles/2005/12/01/rico-rounded-corners-without-all-of-rico>

Another article: Ajax Workshop 2 - Building Tabbed Content at <http://www.ajaxlessons.com/2006/02/18/ajax-workshop-2-building-tabbed-content/>

... and you can find a lot more of them

Visual effects are definitively an important aspect for AJAX applications today. Maybe the "market" will learn the technical difference in some near future. Maybe a "get-it-all-together" solution is the right approach for the "market".

So here is a list of visual effects and other "cool" looking stuff. Some of these effects can also be used in any other html based application; some of them will use AJAX. I'll explain the HTML+CSS+JavaScript that make the effect but I will also bring the higher level approach using ASP.NET web controls that makes using them really easy.

AJAX applications look great! :-)

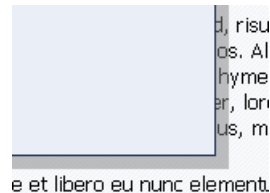
So here comes a library of visual effects. From time to time I will add another one and you will find them also on the demo side at <http://www.mathertel.de/AJAXEngine/> see Examples -> Visual Effects.

## HTML + CSS Shadow Effect with real transparency

If you want to give your objects a kind of 3D feeling then you might want to use shadows for those objects that are placed upon the web page.

### Using special graphics

An often solution for a shadow effect is to use table layout around the main content and arrange a set of graphics around the right and bottom border. You can get really wonderful shadow dropping effects using this approach but because semi transparent images are not very well supported on IE you will have to pay attention to the background color you use on the page and have to mix it into your shadow images.



### Using a Microsoft IE specific CSS filter attribute

When using the IE only you can also use one of the wonderful filter effects that are available in this browser. Just add a CSS filter attribute like that:

```
filter: progid:DXImageTransform.Microsoft.dropShadow(Color=AAAAAA,offX=8,offY=8,positive= true);
```

If you use IE that you can see these shadows on the [entry page](#) for the visual effects library. These shadows are also drawn by using solid colors.

### Using CSS opacity

Here is a better solution that really resembles a kind of shadow because the text and graphics in the shadow are really displayed in dimmed light. The clue to this effect is a built-in opacity CSS graphics effect that is available IE, Mozilla, Firefox - but in different way.

In Internet Explorer: style="filter: alpha(opacity= 50)"

The IE notation of the opacity effect is using a proprietary CSS attribute named filter that can be used to add various effects to a HTML element. Here we use the alpha filter.

In Mozilla/Firefox: style= "-moz-opacity:0.5"

The Mozilla / Firefox provides a proprietary css attribute named -moz-opacity that allows specifying a opacity value.

In Opera: style= "opacity:0.5"

The Opera browser also has a proprietary css attribute named opacity that allows specifying an opacity value.

These 3 CSS attributes can be combined together and every browser ignores all the attributes that are unknown:

```
style="filter: alpha(opacity= 50); -moz-opacity:0.5; opacity: 0.5"
```

### The HTML shadow object

This effect must be applied to rectangle region with a fixed width and height that is over the main content by using an absolute position by using the same size as the html part that is dropping this shadow.

Here is a simple sample that uses this trick. You can see it live at [http://www.mathertel.de/AJAXEngine/S04\\_VisualEffects/ShadowDemo.aspx](http://www.mathertel.de/AJAXEngine/S04_VisualEffects/ShadowDemo.aspx)

The good about this solution is, that no graphics are used and that the shadow does not hide the text below but only seems to dim the available light.



- The outer div element is used to do the absolute positioning of the whole group.
- The first inner <div> element is the object that has the opacity effect applied. It is positioned some pixel to the right and down by using an absolute positioning (relative to the containing <div>) too.
- The Content is placed inside the second inner <div> element that must have a relative position without any offset to be displayed above the shadow object.

Here is the plain HTML code:

```
<div class="VEPart" style="position: relative; width: 120px; top:-90px; left: 40px;">
  <div class="VShadow" style="position: absolute; left: 10px; top: 10px;
    width: 120px; height: 84px; background-color: black;
    filter: alpha(opacity=30); -moz-opacity: 0.3; opacity: 0.3;"> </div>
  <div class="VEContent" style="position: relative; height: 80px;
    background-color: #FFFFDD;"> I am flying above the text and dropping a shadow.</div>
</div>
```

You can see how it looks like on the demo website at

[http://www.mathertel.de/AJAXEngine/S04\\_VisualEffects/ShadowDemo.aspx](http://www.mathertel.de/AJAXEngine/S04_VisualEffects/ShadowDemo.aspx)

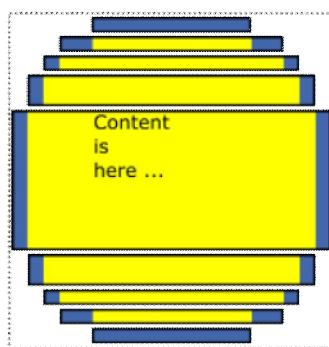
## HTML elements with rounded corners

HTML elements up to now have always a rectangle shape. In the upcoming CSS3 standard (still a proposal) there might be also other border shapes available including rounded corners by specifying a border radius. But we do not have to wait to get this visual effect shown up.

dipiscing elit. Proin  
ellus ornare purus

The trick that is shown here to get the usual rectangle corners into a round shape is to use a collection of thin HTML elements to build a non rectangle border.

The simplest solution to this is to use <div> at the top and bottom and arrange them like this:



### Why not using a client side solution

The Rico framework for example is offering a client-side method to round the elements:

```
<div id='roundDemo' style='width:300px'>Insert content here.</div>
<script>Rico.Corner.round('roundDemo', {corners:'tl br',bgColor:'#adba8c'});</script>
```

By using a JavaScript solution on the client to generate these elements every div element can be transformed to a rounded corner look. The disadvantage of this approach is that there is always a delay and maybe a light flickering because the page will get rendered twice because the borders will be expanded by adding the additional elements.

### Why not using graphic elements

I've also seen solutions that use 4 specific corner graphics. The drawback about this approach is that the colors of these images must correspond exactly to the colors of the other html elements. If you want to change your style then you must change the images too. Again here will be a light flickering because images are often loaded after displaying the page for the first time so it doesn't look very perfect.

### A server-side solution

It is also possible to generate all the additional html elements on the server. This adds some html tags and some more data to the http response when the page gets loaded but also eliminates a lot of JavaScript or images.

For ASP.NET I've written a web control that derives from asp:Panel that does the same and avoids the flickering:

```
<ajax:RoundedArea runat='server' id='any' width='300'>Insert content here.</ajax:RoundedArea>
```

You can see a live sample of it at

[http://www.mathertel.de/AJAXEngine/S04\\_VisualEffects/RoundedDemo.aspx](http://www.mathertel.de/AJAXEngine/S04_VisualEffects/RoundedDemo.aspx)

## Simple sliding sample to move HTML elements around

[http://www.mathertel.de/AJAXEngine/S04\\_VisualEffects/SlidingDemo.aspx](http://www.mathertel.de/AJAXEngine/S04_VisualEffects/SlidingDemo.aspx)

Moving HTML objects to new positions is not hard to implement. You can use absolute positioning and just set the style attributes left and top to the new values but it doesn't look cool if the objects use this kind of hyper speed: the eyes cannot follow and there may be a confusing effect of a sudden rearrangement.



Moving objects more slowly can be simulated by moving them using several steps and only a small distance at once. All we need is a timer to start the next step after the current position is rendered, a little bit of algebra and a heuristic to find the right distance for one step.

Because no local variables can be used when executing timer scripts we declare 4 global variables that hold all the information we need when the next step is to be started:

```
var slidingTimer = null; // the current running timer object
var slidingTarget = null; // the moving object
var slidingLeft = null; // the target left position
var slidingTop = null; // the target top position
```

Starting a sliding movement can be done by calling the startSlidePos function with 3 parameters: the object that should be moved, the new left and the new top coordinate. You can also refer the object by using the unique id.

```
onclick="startSlidePos('i', 10, 270)"
```

Every timer event now calculates the remaining vector but takes no more than 20 pixel length or a third of it whatever is shorter. Only if the target is very close the object will be positioned exactly:

```
// calc the remaining vector
dx = slidingLeft - left;
dy = slidingTop - top;

// calc the movement length along the vector
len = Math.sqrt(dx * dx + dy * dy);
delta = Math.min(20, len/3);

if (len <= 2) {
  // snap exactly
  left = slidingLeft;
  top = slidingTop;
} else {
  left += Math.round(dx * delta / len);
  top += Math.round(dy * delta / len);
} // if
```

Using this calculation the movement gets slower at the end when the target is almost reached.

[view source code online](#)

## Drag and drop HTML objects around using CSS and JavaScript

[http://www.mathertel.de/AJAXEngine/S04\\_VisualEffects/MoverDemo.aspx](http://www.mathertel.de/AJAXEngine/S04_VisualEffects/MoverDemo.aspx)

If you want to enable a custom page layout for the user or a drag & drop functionality in your solution you need a mechanism that enables moving html objects around. Here is a cross browser compatible solution that enables moving of HTML objects.

### Attaching the mouse events

Three events must be captured to drag objects around:

#### **onmousedown**

This event starts the moving scenario and you need to attach a method to this event.

It is sometimes necessary not to move the object that got this event but to identify a specific parent object that also contains other content elements. In this demo sample the title area of a web part is used to drag the whole part around. In the method that is attached to this event the parentNode references are searched until an html object is found that is marked with the className "VEPart".

Then current mouse offset to the left upper corner of the moving object is calculated. This is because you will not start dragging by using a exactly known single point of the object.

Now that we know that a specific object should be dragged around the 2 other events must be monitored and 2 other methods are attached to them.

#### **onmousemove**

This event will be thrown multiple times and we will move the object around by using the new mouse coordinates given each time the event gets fired.

#### **onmouseup**

This event will be thrown at the end when the user wants to place the object at the new position by releasing the mouse button.

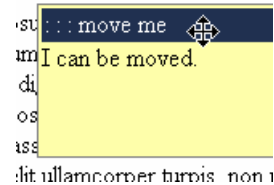
The first event can be caught on specific objects that enable the moving. I use the CSS class "VEMover" to mark these elements and attach a move cursor. The object that is moves is marked by using the CSS class "VEPart" that contains the VEMover.

The 2 other events should be caught on the document level because they will not be thrown always on the object that initiates the moving especially when the mouse pointer is moved very fast.

Because we need a reference to the object that is moved around the onmousedown event also saves a reference to the object by using properties of the MoverBehaviour element like a global variable so we can find the object again when onmousemove and onmouseup events are caught.

### A simple moveable object

```
<div class="VEPart" style="width:180px;height:90px">
  <div class="VEMover">::: move me</div>
  I can be moved.
</div>
```



## The JavaScript implementation

To make the implementation easier I use my JavaScript Control Library that enables writing compatible behaviors for HTML objects. All we need here is to include the 9 kByte jcl.js file and attach the behavior to the VEMover object.

The MoverBehaviour implements the 3 event handlers:

[view source code online](#)

```
var MoverBehaviour = {
  mo: null, // reference to the movable obj,
  x: 0, y: 0,

  // ----- Events -----
  onmousedown: function (evt) {
    evt = evt || window.event;
    this.MoveStart(evt);
  }, // onmousedown

  // track mouse moves. This handler will be attached to the document level !
  _onmousemove: function (evt) {
    evt = evt || window.event;
    MoverBehaviour.MoveIt(evt);
  }, // onmousemove

  // track mouse button up. This handler will be attached to the document level !
  _onmouseup: function (evt) {
    evt = evt || window.event;
    MoverBehaviour.MoveEnd(evt);
  }, // onmouseup

  // ----- Methods -----
  MoveStart: function (evt) {
    // find the moving part (position:absolute or class="VEPart")
    var mo = this;
    while ((mo != null) && (mo.className != "VEPart"))
      mo = mo.parentNode;

    if (mo == null)
      return; // don't move
    MoverBehaviour.mo = mo;

    // calculate mousepointer-object distance
    mo.x = mo.y = 0;
    obj = mo;
    while (obj != null) {
      mo.x += obj.offsetLeft;
      mo.y += obj.offsetTop;
      obj = obj.offsetParent;
    } // while
    mo.x = evt.clientX - mo.x;
    mo.y = evt.clientY - mo.y;

    // make the moving object globally available when mouse is leaving this object.
    jcl.AttachEvent(document, "onmousemove", this._onmousemove);
    jcl.AttachEvent(document, "onmouseup", this._onmouseup);
  }, // MoveStart

  MoveIt: function (evt) {
    var mo = MoverBehaviour.mo;
    if (mo != null) {
      var p = (evt.clientX - mo.x) + "px";
      if (p != mo.style.left) mo.style.left = p;
      p = (evt.clientY - mo.y) + "px";
      if (p != mo.style.top) mo.style.top = p;
    } // if
    // cancel selecting anything
    evt.cancelBubble = true;
  }
};
```

```
    evt.returnValue = false;
  }, // MoveIt

  MoveEnd: function () {
    var mo = MoverBehaviour.mo;
    if (mo != null) {
      MoverBehaviour.mo = null;
      jcl.DetachEvent(document, "onmousemove", this._onmousemove);
      jcl.DetachEvent(document, "onmouseup", this._onmouseup);
    } // if
  } // MoveEnd
} // MoverBehaviour

jcl.LoadBehaviour("moveme", MoverBehaviour);
```

[http://www.mathertel.de/AJAXEngine/S04\\_VisualEffects/MoverDemo.aspx](http://www.mathertel.de/AJAXEngine/S04_VisualEffects/MoverDemo.aspx)

The MoverBehaviour is also available as a separate include file if you want to follow my advice of reusing behaviors by separating them into a JavaScript include file. I will have a typical WebPart sample soon and will use the mover functionality there again.

## PopUp Information

[http://www.mathertel.de/AJAXEngine/S04\\_VisualEffects/PopUpDemo.aspx](http://www.mathertel.de/AJAXEngine/S04_VisualEffects/PopUpDemo.aspx)

PopUp windows and PopUp boxes can be used to show additional information or help text on specific region, buttons, links or words on the page. Move the mouse-cursor over some of the **blue, bold words** in the text of the sample page to see it in action.

The html and CSS code that is used to implement a popup is using some <div> elements and some graphics.

ie id="VEPopUp". Later whe  
 yed application!  
 nt before this time is over n  
 he onmouseover event start

```
<div id="popUp" style="width: 220px;">
  
  <div style="position:relative;">
    <div class="VShadow" style="height:80px"></div>
    <div style="position:relative;border: solid #203050 1px; padding: 4px;height:70px;
background-image:url(../controls/images/buttonbackpushed.png);background-color:#eaeef7;
background-position:top;background-repeat:repeat-x;">
      More information is displayed when the mouse is over the keyword for some time.
    </div>
  </div>
</div>
```

### Starting a popUp

The popup implementation exists only once on a page and is registered for the whole document. To identify html objects that will cause a popup to be shown the new attribute **popText** is used. Every object that has this property set will cause a popUp to be shown.

This attribute alone will not help a lot because the user also needs some hint to see that there is some information available when pointing with the mouse at it. To have a consistent look & feel I use a span element and the classname VEKeyword.

```
<span class="VEKeyword" popText="More details is available here.">[more]</span>
```

The current CSS definition for this class is

```
.VEKeyword {font-weight:bold;color:#203050;cursor:pointer}
```

and it can be found in the central css file ([view source](#)) I use for this site.

### Positioning the popUp

When displaying the popUp element an absolute position is used to place it nearby the object that should be explained.

First the method **\_absolutePosition** is calculating the absolute position of the object by walking the offsetparent hierarchy up the hierarchy. This method is returning a JavaScript object with the values top left width and height.

This position together with the text that should be displayed is the used by the **\_create** method.

If a popUp is displayed for the first time a new object is created and gets the id="VEPopUp". Later when another popup is needed this object will be reused.

### Delaying the popup

When the mouse enters the region of an object the popup should not displayed immediately to avoid flickering but after some time (I choose 300 msec.) the popUp should appear. When the mouse pointer moves on to another element before this time is over no popUp should appear.



2 methods are registered to the 2 events onmouseover and onmouseout. The onmouseover event starts a timer by using a timeout that will then start the show method. If the mouse leaves the element in this time the timer is killed before the popup is shown.

#### Using a shadow

By re-using the **shadow effect** the popUp element appears to fly over the normal surface of the page.

#### Building a JavaScript Behaviour

The sourcecode for the popup mechanism can be found in the **popup.js** and all you have to do is including this file on every page. This file contains a JavaScript Behaviour element that will be attached to the document object. Be sure that the jcl.js file that implements attaching the event handlers is also included.

To make the integration of the popup mechanism into existing pages easy for you and to support the design view of the development environments the mechanism is also wrapped into a web control implemented in **popup.ascx**. Just include this control in the page or master page. It takes care of also including the jcl.js file.

#### The 2 layout versions

The popup always has the width of 220px. The height depends of the text that is displayed.

When the keyword is on the left of the page the popup is extended to the right.

When the keyword is on the right of the page the popup is extended to the left to avoid displaying it outside the current view.

When using web controls they have to be registered on each page or global for the whole web application. To avoid repeating the declarations on every page and for using the same namespace-alias every time I add the registration to the web.config file of this site.

## Building a AJAX enabled popup control

[http://www.mathertel.de/AJAXEngine/S03\\_AJAXControls/AJAXPopUpDemo.aspx](http://www.mathertel.de/AJAXEngine/S03_AJAXControls/AJAXPopUpDemo.aspx)

When fetching the extra information costs a lot of resources (CPU, memory or time) on the server or is more than just a few words it is better to not include it into the page all the time but get it from the server when the user requests for it. This scenario is perfect for using the AJAX technology.

### Using the AJAX engine

To get the information from the server we need a web service with one method that gets a key string as parameter and returns the popup information. We can trust this method so far that we can rely that it is returning valid html code.

The timer object that we need to delay the popup a little bit can be completely replaced by using the delay option of the AJAX engine.

The *onmouseover* event code only needs to start the AJAX engine after checking for a valid html object.

```
ajax.Start(AJAXPopUpBehaviour.action, obj);
```

The action that describes the AJAX mechanism for the popup keeps all the elements of the asynchronous execution together. It prepares the server call by fetching the url of the hyperlink, calls the GetDetails method on the server and then finishes the action by showing the popup control.

```
action: {
  delay: 300,
  queueMultiple: false,
  prepare: function(obj) { return (obj.href); },
  call: "proxies.ServerInfo.GetDetails",
  finish: "AJAXPopUpBehaviour.show",
  onException: proxies.alertException
}
```

### Using a Web Control

But it still can be easier by using a web control:

```
<ajax:PopUp runat="server" id="ajaxpopup1"
  infomethod="proxies.ServerInfo.GetDetails" />
```

The method that returns the information to the client is made available to the client by including the WebService and creating a JavaScript proxy:

```
<script type="text/javascript"
  src="../ajaxcore/GetJavaScriptProxy.aspx?service=../S03_AJAXControls/ServerInfo.asmx">
</script>
```

### Implementing the sample web control

On the client a JavaScript Behavior is used that is included into the page by the web control that is executed on the server.

[http://www.mathertel.de/AJAXEngine/S03\\_AJAXControls/AJAXPopUpDemo.aspx](http://www.mathertel.de/AJAXEngine/S03_AJAXControls/AJAXPopUpDemo.aspx)



# Some HTML and http basics

## Caching with AJAX applications

AJAX applications offer better response times and are faster (or at least seems to be faster) than traditional web applications.

The main reason behind this is the separation of the initial page loading from the loading of additional data and the absence of reloading this page again and again when the data of the page changes.

Loading the page the first time works by using a conventional call of a URL using the http-get mechanisms. Calling the server in the background by using asynchronous requests using the XMLHttpRequest object is the second and often repeated part.

Caching can help to speed up both kinds of server requests and the best results can be bought out by preventing calls to the server.

### Caching the initial page download

The page that is downloaded by navigating to the URL can be improved most effectively using the client-side cache features of the web browser. After adding the right http headers the browser will not ask the server for the specified time for new versions of the url-resource and will rely on the bytes that can be found in the client-side cache. Some things must be taken care of to make this working properly.

There is a very useful tool for windows from Eric Lawrence called fiddler.

<http://www.fiddlertool.com/>.

He also wrote a good article on tuning and the http protocol:

<http://www.fiddlertool.com/Fiddler/help/http/HTTPPerf.mht>.

Another article on this topic that is worth reading:

<http://msdn.microsoft.com/library/default.asp?url=/workshop/author/perf/perftips.asp>

- The initial page download **MUST NOT** contain any data or information that changes frequently or must be available to the client very soon after it changes.  
If any change of this data is initiated by a click or any other action on the client it is possible to force a request in this case even if the cache period has not ended yet. In this case you can live with a long caching period and an immediate change of the page.
- The page **MUST** be designed to be a (almost) static resource from the view of the client.  
It however can vary for different users. When delivering personalized versions, the caching proxy servers as well as the caching features of the server must be turned off.
- The smaller, the faster.  
I do not recommend using huge graphics and many inline style attributes. The Google applications show, that building fast web applications without many graphics is possible.
- Use include files for JavaScript and CSS-files.  
Include files can be cached too on the client and can also be shared among different pages. It is good to use include files with common

functionality or styles. Rarely ore once-only include files slow down the application.

- Use only lowercase character in URLs.

It is not obvious to windows users that page.aspx and Page.aspx are two different resources on the web. Even if the server (IIS and ASP.NET) treats these resources as equal, the client will retrieve and store them twice. The fact that makes them different is the kind of writing in the references in HTML tags "src" and "href" attributes.

Because I am never sure how a reference is written in other places I prefer using lowercase character only.

- Use the right http headers.

For the IE there are the 2 special cache specific attributes pre-check and post-check that should be set correctly so that the IE does NOT even ask for a version but silently uses the version of the resources found in the client cache.

### Caching the asynchronous requests

After the page is loaded there are more requests to the server now by using asynchronous calls in the background using the XMLHttpRequest objects.

When calling long running methods on the server for example complex SQL retrievals or expensive calculations is possible to instruct the server to cache the results and returning them without executing the same methods multiple times.

In ASP.NET you can use the CacheDuration property on the WebMethod attribute to specify the number of seconds the result may stay in the web server cache.

```
[WebMethod(CacheDuration=60)]
```

A simple sample on this can be found in the article at:

<http://support.microsoft.com/default.aspx?scid=kb;en-us;318299>

The advantage in this approach is that all clients share the same cache and if there are multiple clients requesting the same service you might get very good response times.

It's also possible to cache on the client. An approach that leads to less traffic on the net because repeating the same calls can be prevented. Http headers do not help in these situations because the request is not an http-get request and there is always a payload in the http body. Caching must therefore be done by some scripting on the client.

The caching feature in the JavaScript WebService proxy implementation can be enabled by calling the proxies.EnableCache method and passing the function that should further use caching. I added a button to the CalcFactorsAJAX.htm sample to how to enable show this:

```
proxies.EnableCache(proxies.CalcService.CalcPrimeFactors)
```

Also the TableData.aspx sample uses caching to prevent retrieving the same records multiple times.

By calling this method a JavaScript object is added that stored all results and is used to prevent a call to the server if an entry for the parameter already exists inside this object.

This is not a perfect solution, but it works under the following circumstances:

- The parameter must be a string or number that can be used for indexing the properties of a JavaScript object.

- The cache doesn't clear itself. It can be cleared by calling `EnableCache` once again.
- Only methods with a single parameter are supported.

Caching works good into AJAX applications and speeds up by preventing calculations downloads and web server calls.

# Listings

## ~/ajaxcore/GetJavaScriptProxy.aspx

```

<%@ Page Language="C#" Debug="true" %>

<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Xml.Xsl" %>

<!--
/// 19.07.2005 white space removed
/// 20.07.2005 more datatypes and XML Documents
/// 04.09.2005 XslCompiledTransform
-->

<script runat="server">
    private string FetchWsdL(string url) {
        if ((url != null) && (url.StartsWith("~/")))
            url = Request.ApplicationPath + url.Substring(1);
        Uri uri = new Uri(Request.Url, url + "?WSDL");

        HttpWebRequest req = (HttpWebRequest)WebRequest.Create(uri);
        req.Credentials = CredentialCache.DefaultCredentials;
        // req.Proxy = WebRequest.DefaultWebProxy; // running on the same server !
        req.Timeout = 6 * 1000; // 6 seconds

        WebResponse res = req.GetResponse();
#if DOTNET11
        XmlDocument data = new XmlDocument();
        data.Load(res.GetResponseStream());

        XslTransform xsl = new XslTransform();
        xsl.Load(Server.MapPath("~/ajaxcore/wsdL.xslt"));

        System.IO.StringWriter sOut = new System.IO.StringWriter();
        xsl.Transform(data, null, sOut, null);
#else
        XmlReader data = XmlReader.Create(res.GetResponseStream());

        XslCompiledTransform xsl = new XslCompiledTransform();
        xsl.Load(Server.MapPath("~/ajaxcore/wsdL.xslt"));

        System.IO.StringWriter sOut = new System.IO.StringWriter();
        xsl.Transform(data, null, sOut);
#endif
        return (sOut.ToString());
    } // FetchWsdL
</script>

<%
    string asText = Request.QueryString["html"];

    Response.Clear();
    if (asText != null) {
        Response.ContentType = "text/html";
        Response.Write("<pre>");
    } else {
        Response.ContentType = "text/text";
    } // if

    string fileName = Request.QueryString["service"];
    if (fileName == null)
        fileName = "CalcService";

    // get good filenames only (local folder)

```

```

    if ((fileName.IndexOf('$') >= 0) || (Regex.IsMatch(fileName,
@"\b(COM\d|LPT\d|CON|PRN|AUX|NUL)\b", RegexOptions.IgnoreCase)))
        throw new ApplicationException("Error in filename.");

    if (! Server.MapPath(fileName).StartsWith(Request.PhysicalApplicationPath,
StringComparison.InvariantCultureIgnoreCase))
        throw new ApplicationException("Can show local files only.");

    string ret = FetchWsdL(fileName);
    ret = Regex.Replace(ret, @"\n *", "\n");
    ret = Regex.Replace(ret, @"\r\n *", "");
    ret = Regex.Replace(ret, @"\r\n", ",\n");
    ret = Regex.Replace(ret, @"\r\n]", "]");
    ret = Regex.Replace(ret, @"\r\n; *", ";");
    Response.Write(ret);
%>

```

## ~/ajaxcore/wsdL.xslt

```

<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
>
  <!--
  // wsdl.xslt
  // The WSDL to JavaScript transformation.
  // Copyright by Matthias Hertel, http://www.mathertel.de
  // This work is licensed under a Creative Commons Attribution 2.0 Germany License.
  // See http://creativecommons.org/licenses/by/2.0/de/
  // - - - - -
  // 19.07.2005 optional documentation
  // 20.07.2005 more datatypes and XML Documents
  // 20.07.2005 more datatypes and XML Documents fixed
  // 03.12.2005 compatible to axis and bea WebServices. Thanks to Thomas Rudin
  // 07.03.2006 Now this xslt is independent of the alias used for the namespace of XMLSchema in
  the wsdl.
  //      Thanks to António Cruz for this great trick.
  // 31.03.2006 Bug on xml types fixed.
  // 19.11.2006 supporting (old) RPC encoding.
  // 26.11.2006 better serializing complex objects
  -->
  <xsl:strip-space elements="*" />

  <xsl:output method="text" version="4.0" />
  <xsl:param name="alias">
    <xsl:value-of select="wsdl:definitions/wsdl:service/@name" />
  </xsl:param>

  <xsl:variable name="XSDPrefix"
  select="name(//namespace::*[.='http://www.w3.org/2001/XMLSchema'])" />

  <xsl:template match="/">
    // javascript proxy for SOAP based web services
    // by Matthias Hertel
    /* <xsl:value-of select="wsdl:definitions/wsdl:documentation" /> */
    <xsl:for-each select="/wsdl:definitions/wsdl:service/wsdl:port[soap:address]">
      <xsl:call-template name="soapport" />
    </xsl:for-each>
  </xsl:template>

  <xsl:template name="soapport">
    proxies.<xsl:value-of select="$alias" /> = {
    url: "<xsl:value-of select="soap:address/@location" />",
    ns: "<xsl:value-of select="/wsdl:definitions/wsdl:types/s:schema/@targetNamespace" />"
    } // proxies.<xsl:value-of select="$alias" />
    <xsl:text>&#x000D;&#x000A;</xsl:text>

```

```

<xsl:for-each select="/wsdl:definitions/wsdl:binding[@name = substring-
after(current()/@binding, ':')]">
  <xsl:call-template name="soapbinding11" />
</xsl:for-each>
</xsl:template>

<xsl:template name="soapbinding11">
  <xsl:variable name="portTypeName" select="substring-after(current()/@type, ':')" />
  <xsl:for-each select="wsdl:operation">
    <xsl:variable name="inputMessageName" select="substring-
after(/wsdl:definitions/wsdl:portType[@name = $portTypeName]/wsdl:operation[@name =
current()/@name]/wsdl:input/@message, ':')" />
    <xsl:variable name="outputMessageName" select="substring-
after(/wsdl:definitions/wsdl:portType[@name = $portTypeName]/wsdl:operation[@name =
current()/@name]/wsdl:output/@message, ':')" />
    /* inputMessageName='<xsl:value-of select="$inputMessageName" />',
outputMessageName='<xsl:value-of select="$outputMessageName" />' */

    <xsl:for-each select="/wsdl:definitions/wsdl:portType[@name =
$portTypeName]/wsdl:operation[@name = current()/@name]/wsdl:documentation">
      /** <xsl:value-of select="." /> */
    </xsl:for-each>
    proxies.<xsl:value-of select="$alias" />.<xsl:value-of select="@name" /> = function () {
return(proxies.callSoap(arguments)); }
    proxies.<xsl:value-of select="$alias" />.<xsl:value-of select="@name" />.fname =
"<xsl:value-of select="@name" />";
    proxies.<xsl:value-of select="$alias" />.<xsl:value-of select="@name" />.service =
proxies.<xsl:value-of select="$alias" />;
    proxies.<xsl:value-of select="$alias" />.<xsl:value-of select="@name" />.action =
"\<xsl:value-of select="soap:operation/@soapAction" />\\"";
    proxies.<xsl:value-of select="$alias" />.<xsl:value-of select="@name" />.params =
[<xsl:for-each select="/wsdl:definitions/wsdl:message[@name = $inputMessageName]">
  <xsl:call-template name="soapMessage" />
</xsl:for-each>];
    proxies.<xsl:value-of select="$alias" />.<xsl:value-of select="@name" />.rtype = [<xsl:for-
each select="/wsdl:definitions/wsdl:message[@name = $outputMessageName]">
  <xsl:call-template name="soapMessage" />
</xsl:for-each>];
  </xsl:for-each>
</xsl:template>

<xsl:template name="soapElem">
  <xsl:param name="type"/>
  <xsl:param name="name"/>
  <!-- An annotation to comparisation of the types:
In XPath 1.0 there is no built in function to check if a string matches a specific type
declaration.
The trick with $XSDPrefix and the 2 following variables help out of this.
Thanks to Ant3nio Cruz for this great trick.

This condition works on ASP.NET with ms - extensions available:
when test="msxsl:namespace-uri($type)='http://www.w3.org/2001/XMLSchema' and msxsl:local-
name($type)='boolean' "
This condition works with XPath 2.0 functions available:, (see http://www.w3.org/TR/xpath-
functions/#func-namespace-uri-from-QName)
when test="namespace-uri-from-QName($type)='http://www.w3.org/2001/XMLSchema' and local-
name-from-QName($type)='boolean' "
-->
  <xsl:variable name="pre" select="substring-before($type, ':')" />
  <xsl:variable name="post" select="substring-after($type, ':')" />
  <xsl:choose>
    <xsl:when test="$pre != '' and $pre!=$XSDPrefix and $pre!='tns'">
      "<xsl:value-of select="$name" />"
    </xsl:when>

    <xsl:when test="$post='string'">
      "<xsl:value-of select="$name" />"
    </xsl:when>

    <xsl:when test="$post='int' or $post='unsignedInt' or $post='short' or
$post='unsignedShort'
or $post='unsignedLong' or $post='long'">
      "<xsl:value-of select="$name" />:int"
  </xsl:choose>

```



```

</xsl:when>

<xsl:when test="$post='double' or $post='float'">
  "<xsl:value-of select="$name" />:float"
</xsl:when>
<xsl:when test="$post='dateTime'">
  "<xsl:value-of select="$name" />:date"
</xsl:when>

<xsl:when test="$post='boolean'">
  "<xsl:value-of select="$name" />:bool"
</xsl:when>

<!-- arrays !-->
<xsl:when test="$type='tns:ArrayOfString'">
  "<xsl:value-of select="$name" />:s[]"
</xsl:when>
<xsl:when test="$type='tns:ArrayOfInt' or $type='tns:ArrayOfUnsignedInt' or
$type='tns:ArrayOfShort' or $type='tns:ArrayOfUnsignedShort' or $type='tns:ArrayOfLong' or
$type='tns:ArrayOfUnsignedLong'">
  "<xsl:value-of select="$name" />:int[]"
</xsl:when>
<xsl:when test="$type='tns:ArrayOfFloat'">
  "<xsl:value-of select="$name" />:float[]"
</xsl:when>
<xsl:when test="$type='tns:ArrayOfBoolean'">
  "<xsl:value-of select="$name" />:bool[]"
</xsl:when>

<!-- ASP.NET datasets-->
<xsl:when test="count(/s:complexType/s:sequence/*) > 1">
  "<xsl:value-of select="$name" />:ds"
</xsl:when>

<!-- XML Documents -->
<xsl:when test="/s:complexType/s:sequence/s:any">
  "<xsl:value-of select="$name" />:x"
</xsl:when>

<!-- complex objects -->
<xsl:when test="substring-before($type, ':')='tns' or not($type)">
  "<xsl:value-of select="$name" />:ds"
</xsl:when>

<xsl:otherwise>
  "<xsl:value-of select="$name" />"
</xsl:otherwise>
</xsl:choose>
</xsl:template>

<xsl:template name="soapMessage">
  <xsl:choose>
    <xsl:when test="wsdl:part[@type]">
      <!-- SOAP RPC encoding -->
      <xsl:for-each select="wsdl:part">
        <xsl:call-template name="soapElem">
          <xsl:with-param name="name" select="@name" />
          <xsl:with-param name="type" select="@type" />
        </xsl:call-template>
        <xsl:if test="position()=last()">,</xsl:if>
      </xsl:for-each>
    </xsl:when>

    <xsl:otherwise>
      <!-- SOAP Document encoding -->
      <xsl:variable name="inputElementName" select="substring-after(wsdl:part/@element, ':')"/>

      <xsl:for-each
select="/wsdl:definitions/wsdl:types/s:schema/s:element[@name=$inputElementName]//s:element">
        <xsl:call-template name="soapElem">
          <xsl:with-param name="name" select="@name" />

```

```

        <xsl:with-param name="type" select="@type" />
    </xsl:call-template>
    <xsl:if test="position()=last()"></xsl:if>
</xsl:for-each>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

</xsl:stylesheet>

```

## ~/ajaxcore/ajax.js

```

// ajax.js
// Common Javascript methods and global objects
// Ajax framework for Internet Explorer (6.0, ...) and Firefox (1.0, ...)
// Copyright by Matthias Hertel, http://www.mathertel.de
// This work is licensed under a Creative Commons Attribution 2.0 Germany License.
// See http://creativecommons.org/licenses/by/2.0/de/
// More information on: http://ajaxaspects.blogspot.com/ and http://ajaxaspekte.blogspot.com/
// -----
// 05.06.2005 created by Matthias Hertel.
// 19.06.2005 minor corrections to webservices.
// 25.06.2005 ajax action queue and timing.
// 02.07.2005 queue up actions fixed.
// 10.07.2005 ajax.timeout
// 10.07.2005 a option object that is passed from ajax.Start() to prepare() is also queued.
// 10.07.2005 a option object that is passed from ajax.Start() to prepare(), finish()
// and onException() is also queued.
// 12.07.2005 correct xml encoding when CallSoap()
// 20.07.2005 more datatypes and XML Documents
// 20.07.2005 more datatypes and XML Documents fixed
// 06.08.2005 caching implemented.
// 07.08.2005 bugs fixed, when queuing without a delay time.
// 04.09.2005 bugs fixed, when entering non-multiple actions.
// 07.09.2005 proxies.IsActive added
// 27.09.2005 fixed error in handling bool as a datatype
// 13.12.2005 WebServices with arrays on strings, ints, floats and booleans - still undocumented
// 27.12.2005 fixed: empty string return values enabled.
// 27.12.2005 enable the late binding of proxy methods.
// 21.01.2006 void return bug fixed.
// 18.02.2006 typo: Finsh -> Finish.
// 25.02.2006 better xmlhttp request object retrieval, see
http://blogs.msdn.com/ie/archive/2006/01/23/516393.aspx
// 22.04.2006 progress indicator added.
// 28.01.2006 void return bug fixed again?
// 09.03.2006 enable late binding of prepare and finish methods by using an expression.

// ----- global variable for the proxies to webservices. -----

/// <summary>The root object for the proxies to webservices.</summary>
var proxies = new Object();

proxies.current = null; // the current active webservice call.
proxies.xmlhttp = null; // The current active xmlhttp object.

// ----- global variable for the ajax engine. -----

/// <summary>The root object for the ajax engine.</summary>
var ajax = new Object();

ajax.current = null; /// The current active AJAX action.
ajax.option = null; /// The options for the current active AJAX action.

ajax.queue = new Array(); /// The pending AJAX actions.
ajax.options = new Array(); /// The options for the pending AJAX actions.

ajax.timer = null; /// The timer for delayed actions.

ajax.progress = false; /// show a progress indicator
ajax.progressTimer = null; /// a timer-object that help displaying the progress indicator not too
often.

```

```

// ----- AJAX engine and actions implementation -----

///

```

```

if (ajax.queue.length == 0)
    return; // nothing to do.

ca = ajax.queue[0];
co = ajax.options[0];
if ((forceStart == true) || (ca.delay == null) || (ca.delay == 0)) {
    // start top action
    ajax.current = ca;
    ajax.queue.shift();
    ajax.option = co;
    ajax.options.shift();

    // get the data
    if (ca.prepare != null)
        try {
            data = ca.prepare(co);
        } catch (ex) { }

    if (ca.call == null) {
        // no call
        ajax.Finish(data);
    } else {
        ajax.StartProgress();

        // start the call
        ca.call.func = ajax.Finish;
        ca.call.onException = ajax.Exception;
        ca.call(data);
        // start timeout timer
        if (ca.timeout != null)
            ajax.timer = window.setTimeout(ajax.Cancel, ca.timeout * 1000);
    } // if

} else {
    // start a timer and wait
    ajax.timer = window.setTimeout(ajax.EndWait, ca.delay);
} // if
} // ajax._next

///

```

```

// reset the running action
ajax.current = null;
ajax.option = null;
ajax.EndProgress();
ajax._next(false)
} // ajax.Finish

///

```

```

a.style.position = "absolute";
a.style.right = "2px";
a.style.top = document.documentElement.scrollTop + 2 + "px";
a.style.width = "98px";
a.style.height = "16px";
a.style.padding = "2px";
a.style.verticalAlign = "bottom";
a.style.backgroundColor="#51c77d";

a.innerHTML = "<img style='vertical-align:bottom' src='" + path + "ajax-
loader.gif'>&nbsp;please wait...";
document.body.appendChild(a);

} else if (a) {
a.style.display="none";
} // if
} // ajax.ShowProgress

// ----- webservice proxy implementation -----

///<summary>Execute a soap call.
///Build the xml for the call of a soap method of a webservice
///and post it to the server.</summary>
proxies.callSoap = function (args) {
var p = args.callee;
var x = null;

// check for existing cache-entry
if (p._cache != null) {
if ((p.params.length == 1) && (args.length == 1) && (p._cache[args[0]] != null)) {
if (p.func != null) {
p.func(p._cache[args[0]]);
return(null);
} else {
return(p._cache[args[0]]);
} // if
} else {
p._cachekey = args[0];
} // if
} // if

proxies.current = p;

// from http://blogs.msdn.com/ie/archive/2006/01/23/516393.aspx
if (window.XMLHttpRequest) {
// if IE7, Mozilla, Safari, etc: Use native object
x = new XMLHttpRequest()

} else if (window.ActiveXObject) {
// ...otherwise, use the ActiveX control for IE5.x and IE6
try { x = new ActiveXObject("Msxml2.XMLHTTP"); } catch (e) { }
if (x == null)
try { x = new ActiveXObject("Microsoft.XMLHTTP"); } catch (e) { }
} // if

proxies.xmlhttp = x;

// envelope start
var soap = "<?xml version='1.0' encoding='utf-8'?>"
+ "<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>"
+ "<soap:Body>"
+ "<" + p.fname + " xmlns='" + p.service.ns + "'>";

// parameters
for (n = 0; (n < p.params.length) && (n < args.length); n++) {
var val = args[n];
var typ = p.params[n].split(':');

if ((typ.length == 1) || (typ[1] == "string")) {
val = String(args[n]).replace(/&/g, "&amp;").replace(/</g, "&lt;").replace(/>/g, "&gt;");
} else if (typ[1] == "int") {

```

```

    val = parseInt(args[n]);
  } else if (typ[1] == "float") {
    val = parseFloat(args[n]);

  } else if ((typ[1] == "x") && (typeof(args[n]) == "string")) {
    val = args[n];

  } else if ((typ[1] == "x") && (typeof(XMLSerializer) != "undefined")) {
    val = (new XMLSerializer()).serializeToString(args[n].firstChild);

  } else if (typ[1] == "x") {
    val = args[n].xml;

  } else if ((typ[1] == "bool") && (typeof(args[n]) == "string")) {
    val = args[n].toLowerCase();

  } else if (typ[1] == "bool") {
    val = String(args[n]).toLowerCase();

  } else if (typ[1] == "date") {
    // calculate the xml format for datetime objects from a javascript date object
    var s, ret;
    ret = String(val.getFullYear());
    ret += "-";
    s = String(val.getMonth() + 1);
    ret += (s.length == 1 ? "0" + s : s);
    ret += "-";
    s = String(val.getDate() + 1);
    ret += (s.length == 1 ? "0" + s : s);
    ret += "T";
    s = String(val.getHours() + 1);
    ret += (s.length == 1 ? "0" + s : s);
    ret += ":";
    s = String(val.getMinutes() + 1);
    ret += (s.length == 1 ? "0" + s : s);
    ret += ":";
    s = String(val.getSeconds() + 1);
    ret += (s.length == 1 ? "0" + s : s);
    val = ret;

  } else if (typ[1] == "s[]") {
    val = "<string>" + args[n].join("</string><string>") + "</string>";

  } else if (typ[1] == "int[]") {
    val = "<int>" + args[n].join("</int><int>") + "</int>";

  } else if (typ[1] == "float[]") {
    val = "<float>" + args[n].join("</float><float>") + "</float>";

  } else if (typ[1] == "bool[]") {
    val = "<boolean>" + args[n].join("</boolean><boolean>") + "</boolean>";

  } // if
  soap += "<" + typ[0] + ">" + val + "</" + typ[0] + ">"
} // for

// envelope end
soap += "</" + p.fname + ">"
+ "</soap:Body>"
+ "</soap:Envelope>";

// enable cookieless sessions:
var u = p.service.url;
var cs = document.location.href.match(/\/\(.*\)\//);
if (cs != null) {
  u = p.service.url.split('/');
  u[3] += cs[0].substr(0, cs[0].length-1);
  u = u.join('/');
} // if

x.open("POST", u, (p.func != null));

```

```

x.setRequestHeader("SOAPAction", p.action);
x.setRequestHeader("Content-Type", "text/xml; charset=utf-8");

if (p.corefunc != null) {
    // async call with xmlhttp-object as parameter
    x.onreadystatechange = p.corefunc;
    x.send(soap);
} else if (p.func != null) {
    // async call
    x.onreadystatechange = proxies._response;
    x.send(soap);
} else {
    // sync call
    x.send(soap);
    return(proxies._response());
} // if
} // proxies.callSoap

// cancel the running webservice call.
// raise: set raise to false to prevent raising an exception
proxies.cancel = function(raise) {
    var cc = proxies.current;
    var cx = proxies.xmlhttp;

    if (raise == null) raise == true;

    if (proxies.xmlhttp != null) {
        proxies.xmlhttp.onreadystatechange = function() { };
        proxies.xmlhttp.abort();
        if (raise && (proxies.current.onException != null))
            proxies.current.onException("WebService call was canceled.")
        proxies.current = null;
        proxies.xmlhttp = null;
    } // if
} // proxies.cancel

// px is a proxies.service.func object !
proxies.EnableCache = function (px) {
    // attach an empty _cache object.
    px._cache = new Object();
} // proxies.EnableCache

// check, if a call is currently waiting for a result
proxies.IsActive = function () {
    return(proxies.xmlhttp != null);
} // proxies.IsActive

///<summary>Callback method for a webservice call that dispatches the response to servive.func or
service.onException.</summary>
///<remarks>for private use only.</remarks>
proxies._response = function () {
    var ret = null;
    var x = proxies.xmlhttp;
    var cc = proxies.current;
    var rtype = null;

    if ((cc.rtype.length > 0) && (cc.rtype[0] != null))
        rtype = cc.rtype[0].split(':');

    if ((x != null) && (x.readyState == 4)) {
        if (x.status == 200) {
            var xNode = null;

            if (rtype != null)
                xNode = x.responseXML.getElementsByTagName(rtype[0])[0];

            if (xNode == null) {

```



```

ret = null;

} else if (xNode.firstChild == null) { // 27.12.2005: empty string return values
ret = ((rtype.length == 1) || (rtype[1] == "string") ? "" : null);

} else if ((rtype.length == 1) || (rtype[1] == "string")) {
ret = (xNode.textContent ? xNode.textContent : xNode.text);

} else if (rtype[1] == "bool") {
ret = ((xNode.textContent ? xNode.textContent : xNode.text).toLowerCase() == "true");

} else if (rtype[1] == "int") {
ret = parseInt(xNode.textContent ? xNode.textContent : xNode.text);

} else if (rtype[1] == "float") {
ret = parseFloat(xNode.textContent ? xNode.textContent : xNode.text);

} else if ((rtype[1] == "x") && (typeof(XMLSerializer) != "undefined")) {
ret = (new XMLSerializer()).serializeToString(xNode.firstChild);
ret = ajax._getXMLDOM(ret);

} else if ((rtype[1] == "ds") && (typeof(XMLSerializer) != "undefined")) {
// ret = (new
XMLSerializer()).serializeToString(xNode.firstChild.nextSibling.firstChild);
ret = (new XMLSerializer()).serializeToString(xNode);
ret = ajax._getXMLDOM(ret);

} else if (rtype[1] == "x") {
ret = xNode.firstChild.xml;
ret = ajax._getXMLDOM(ret);

} else if (rtype[1] == "ds") {
// ret = xNode.firstChild.nextSibling.firstChild.xml;
ret = xNode.xml;
ret = ajax._getXMLDOM(ret);

} else if (rtype[1] == "s[]") {
// Array of strings
ret = new Array();
xNode = xNode.firstChild;
while (xNode != null) {
ret.push(xNode.textContent ? xNode.textContent : xNode.text);
xNode = xNode.nextSibling;
} // while

} else if (rtype[1] == "int[]") {
// Array of int
ret = new Array();
xNode = xNode.firstChild;
while (xNode != null) {
ret.push(parseInt(xNode.textContent ? xNode.textContent : xNode.text));
xNode = xNode.nextSibling;
} // while

} else if (rtype[1] == "float[]") {
// Array of float
ret = new Array();
xNode = xNode.firstChild;
while (xNode != null) {
ret.push(parseFloat(xNode.textContent ? xNode.textContent : xNode.text));
xNode = xNode.nextSibling;
} // while

} else if (rtype[1] == "bool[]") {
// Array of bool
ret = new Array();
xNode = xNode.firstChild;
while (xNode != null) {
ret.push((xNode.textContent ? xNode.textContent : xNode.text).toLowerCase() == "true");
xNode = xNode.nextSibling;
} // while

} else {

```

```

    ret = (xNode.textContent ? xNode.textContent : xNode.text);
  } // if

  // store to _cache
  if ((cc._cache != null) && (cc._cachekey != null)) {
    cc._cache[cc._cachekey] = ret;
    cc._cachekey = null;
  } // if

  proxies.xmlhttp = null;
  proxies.current = null;

  if (cc.func == null) {
    return(ret); // sync
  } else {
    cc.func(ret); // async
    return(null);
  } // if

} else if (proxies.current.onException == null) {
  // no exception

} else {
  // raise an exception
  ret = new Error();

  if (x.status == 404) {
    ret.message = "The webservice could not be found.";

  } else if (x.status == 500) {
    ret.name = "SoapException";
    var n = x.responseXML.documentElement.firstChild.firstChild.firstChild;
    while (n != null) {
      if (n.nodeName == "faultcode") ret.message = n.firstChild.nodeValue;
      if (n.nodeName == "faultstring") ret.description = n.firstChild.nodeValue;
      n = n.nextSibling;
    } // while

  } else if ((x.status == 502) || (x.status == 12031)) {
    ret.message = "The server could not be found.";

  } else {
    // no classified response.
    ret.message = "Result-Status:" + x.status + "\n" + x.responseText;
  } // if
  proxies.current.onException(ret);
} // if

proxies.xmlhttp = null;
proxies.current = null;
} // if
} // proxies._response

///<summary>Callback method to show the result of a soap call in an alert box.</summary>
///<remarks>To set up a debug output in an alert box use:
///proxies.service.method.corefunc = proxies.alertResult;</remarks>
proxies.alertResult = function () {
  var x = proxies.xmlhttp;

  if (x.readyState == 4) {
    if (x.status == 200) {
      if (x.responseXML.documentElement.firstChild.firstChild.firstChild == null)
        alert("(no result)");
      else
        alert(x.responseXML.documentElement.firstChild.firstChild.firstChild.firstChild.nodeValue);

    } else if (x.status == 404) { alert("Error!\n\nThe webservice could not be found.");

    } else if (x.status == 500) {
      // a SoapException
      var ex = new Error();

```

```

ex.name = "SoapException";
var n = x.responseXML.documentElement.firstChild.firstChild.firstChild;
while (n != null) {
    if (n.nodeName == "faultcode") ex.message = n.firstChild.nodeValue;
    if (n.nodeName == "faultstring") ex.description = n.firstChild.nodeValue;
    n = n.nextSibling;
} // while
alert("The server threw an exception.\n\n" + ex.message + "\n\n" + ex.description);

} else if (x.status == 502) { alert("Error!\n\nThe server could not be found.");

} else {
    // no classified response.
    alert("Result-Status:" + x.status + "\n" + x.responseText);
} // if

proxies.xmlhttp = null;
proxies.current = null;
} // if
} // proxies.alertResult

///Show all the details of the returned data of a webservice call.
//Use this method for debugging transmission problems.</summary>
//<remarks>To set up a debug output in an alert box use:
//proxies.service.method.corefunc = proxies.alertResponseText;</remarks>
proxies.alertResponseText = function () {
    if (proxies.xmlhttp.readyState == 4)
        alert("Status:" + proxies.xmlhttp.status + "\nRESULT:" + proxies.xmlhttp.responseText);
} // proxies.alertResponseText

///show the details about an exception.</summary>
proxies.alertException = function(ex) {
    var s = "Exception:\n\n";

    if (ex.constructor == String) {
        s = ex;
    } else {
        if ((ex.name != null) && (ex.name != ""))
            s += "Type: " + ex.name + "\n\n";

        if ((ex.message != null) && (ex.message != ""))
            s += "Message:\n" + ex.message + "\n\n";

        if ((ex.description != null) && (ex.description != "") && (ex.message != ex.description))
            s += "Description:\n" + ex.description + "\n\n";
    } // if
    alert(s);
} // proxies.alertException

///

```

```

    } catch (e) { }
  } // if

  if (obj != null) {
    obj.async = false;
    obj.validateOnParse = false;
  } // if
  obj.loadXML(xmlText);
} // if
return(obj);
} // _getXMLDOM

///<summary>show the details of a javascript object.</summary>
///<remarks>This helps a lot while developing and debugging.</remarks>
function inspectObj(obj) {
  var s = "InspectObj:";

  if (obj == null) {
    s = "(null)"; alert(s); return;
  } else if (obj.constructor == String) {
    s = "\"" + obj + "\"";
  } else if (obj.constructor == Array) {
    s += " _ARRAY";
  } else if (typeof(obj) == "function") {
    s += " [function]" + obj;
  }

  } else if ((typeof(XMLSerializer) != "undefined") && (obj.constructor == XMLDocument)) {
    s = "[XMLDocument]:\n" + (new XMLSerializer()).serializeToString(obj.firstChild);
    alert(s); return;
  }

  } else if ((obj.constructor == null) && (typeof(obj) == "object") && (obj.xml != null)) {
    s = "[XML]:\n" + obj.xml;
    alert(s); return;
  }

  for (p in obj) {
    try {
      if (obj[p] == null) {
        s += "\n" + String(p) + " (...>";

      } else if (typeof(obj[p]) == "function") {
        s += "\n" + String(p) + " [function]";

      } else if (obj[p].constructor == Array) {
        s += "\n" + String(p) + " [ARRAY]: " + obj[p];
        for (n = 0; n < obj[p].length; n++)
          s += "\n  " + n + ": " + obj[p][n];

      } else {
        s += "\n" + String(p) + " [" + typeof(obj[p]) + "]: " + obj[p];
      } // if
    } catch (e) { s+= e;}
  } // for
  alert(s);
} // inspectObj

// ----- End -----

```

## ~/controls/jcl.js

```

// jcl.js: JavaScript Common behaviors Library
// -----
// Behaviour loading and DataConnections for AJAX Controls
// Copyright by Matthias Hertel, http://www.mathertel.de
// This work is licensed under a Creative Commons Attribution 2.0 Germany License.
// See http://creativecommons.org/licenses/by/2.0/de/
// More information on: http://ajaxaspects.blogspot.com/ and http://ajaxaspekte.blogspot.com/
// -----
// 12.08.2005 created
// 31.08.2005 jcl object used instead of global methods and objects
// 04.09.2005 GetPropValue added.

```

```

// 15.09.2005 CloneObject added.
// 27.09.2005 nosubmit attribute without forms bug fixed.
// 29.12.2005 FindBehaviourElement added.
// 02.04.2005 term method added to release bound html objects.
// 07.05.2006 controlsPath added.
// 09.05.2006 Raise only changed values.
// 09.05.2006 Load() and RaiseAll() added to support default-values on page start.
// 03.06.2006 binding to the document enabled for FF.

var isIE = (window.navigator.userAgent.indexOf("MSIE") > 0);

var jcl = {

// attach events, methods and default-values to a html object (using the english spelling)

LoadBehaviour: function (obj, behavior) {
    if ((obj != null) && (obj.constructor == String))
        obj = document.getElementById(obj);

    if (obj == null) {
        alert("LoadBehaviour: obj argument is missing.");
    } else if (behavior == null) {
        alert("LoadBehaviour: behavior argument is missing.");
    } else {
        if (!(isIE) && (obj.attributes != null)) {
            // copy all attributes to this.properties
            for (var n = 0; n < obj.attributes.length; n++)
                if (obj[obj.attributes[n].name] == null)
                    obj[obj.attributes[n].name] = obj.attributes[n].value;
        } // if

        for (var p in behavior) {
            if (p.substr(0, 2) == "on") {
                this.AttachEvent(obj, p, behavior[p]);

            } else if ((behavior[p] == null) || (behavior[p].constructor != Function)) {
                // set default-value
                if (obj[p] == null)
                    obj[p] = behavior[p];

            } else {
                // attach method
                obj[p] = behavior[p];
            } // if
        } // for

        obj._attachedBehaviour = behavior;
    } // if
    if (obj != null)
        this.List.push(obj);
}, // LoadBehaviour

// find the parent node that has the bahaviour attached.
FindBehaviourElement: function (obj, behaviorDef) {
    while ((obj != null) && (obj._attachedBehaviour != behaviorDef))
        obj = obj.parentNode;
    return(obj);
},

// cross browser compatible helper to register for events
AttachEvent: function (obj, eventname, handler) {
    if (isIE) {
        obj.attachEvent(eventname, handler);
    } else {
        obj.addEventListener(eventname.substr(2), handler, false);
    } // if
}, // AttachEvent

// cross browser compatible helper to register for events
DetachEvent: function (obj, eventname, handler) {

```

```

if (isIE) {
    obj.detachEvent(eventname, handler);
} else {
    obj.removeEventListener(eventname.substr(2), handler, false);
} // if
}, // DetachEvent

CloneObject: function (srcObject) {
    var tarObject = new Object();
    for (p in srcObject)
        tarObject[p] = srcObject[p];
    return(tarObject);
}, // CloneObject

// Link all objects with behaviors
List: [],

// ----- Data connections between Controls on the client side. -----

DataConnections: {
// Providers: { },
    _consumers: { },
    _values: { },

// remember an object to be a provider
RegisterProvider: function (obj, propName) {
// no need for this yet.
},

// remember an object to be a consumer
RegisterConsumer: function (obj, propName) {
    propName = propName.toLowerCase();
    if (this._consumers[propName] == null)
        this._consumers[propName] = new Array();
    this._consumers[propName].push(obj);
},

// Load a property but do not Raise an event.
Load: function (propName, propValue) {
    propName = propName.toLowerCase();

// store actual property value
    this._values[propName] = propValue;
}, // Load

// broadcast the change notification of a property
// set force to true to raise an event even if the value has not changed.
Raise: function (propName, propValue, force) {
    propName = propName.toLowerCase();

    if ((this._values[propName] != propValue) || (force == true)) {
// store actual property value
        this._values[propName] = propValue;

// Send to property specific Consumers
        var _consumers = this._consumers[propName];
        if (_consumers != null) {
            for (var n = 0; n < _consumers.length; n++) {
                _consumers[n].GetValue(propName, propValue);
            } // for
        } // if

// Send to wildcard _consumers too
        _consumers = this._consumers["*"];
        if (_consumers != null) {
            for (var n = 0; n < _consumers.length; n++) {
                _consumers[n].GetValue(propName, propValue);
            } // for
        } // if
    }
}

```

```

    // store actual property value
    this._values[propName] = propValue;
  } // if
}, // Raise

RaiseAll: function () {
  for (prop in this._values) {
    var val = this._values[prop];
    this.Raise(prop, val, true);
  }
}, // RaiseAll

// send the change notification of a property directly to another object.
RaiseDirect: function (obj, propName, propValue) {
  if (obj.constructor == String)
    obj = document.getElementById(obj);

  propName = propName.toLowerCase();
  obj.GetValue(propName, propValue);

  // store actual property value
  this._values[propName] = propValue;
}, // RaiseDirect

// retrieve an actual property value
GetPropValue: function (propName) {
  propName = propName.toLowerCase();
  return(this._values[propName]);
}, // GetPropValue

// persist an actual property value into a local cookie
PersistPropValue: function (propName) {
  propName = propName.toLowerCase();
  window.document.cookie = "jcl." + propName + "=" + escape(this._values[propName]);
} // PersistPropValue
}, // DataConnections

// find a relative link to the controls folder containing jcl.js
GetControlsPath: function () {
  var path = "../controls/"
  for (var n in document.scripts) {
    s = String(document.scripts[n].src);
    if ((s != null) && (s.length >= 6) && (s.substr(s.length - 6).toLowerCase() == "jcl.js"))
      path = s.substr(0, s.length - 6);
  } // for
  return(path);
}, // GetControlsPath

// init all objects when the page is loaded
onload: function(evt) {
  evt = evt || window.event;

  for (var n in jcl.List) {
    var obj = jcl.List[n];
    if ((obj != null) && (obj.init != null))
      obj.init();
  } // for

  // raise all persisted values
  var pv = document.cookie.replace(/; /g, "").split(";");
  for (n in pv) {
    if (pv[n].substr(0, 4) == "jcl.") {
      var p = pv[n].substr(4).split("=");
      jcl.DataConnections.Raise(p[0], p[1]);
    } // if
  } // for
}

```

```

}, // onload

// init all objects when the page is loaded
onunload: function(evt) {
  evt = evt || window.event;

  for (var n in jcl.List) {
    var obj = jcl.List[n];
    if ((obj != null) && (obj.term != null))
      obj.term();
  } // for
}, // onunload

// allow non-submitting input elements
onkeypress: function(evt) {
  evt = evt || window.event;

  if (evt.keyCode == 13) {
    var obj = document.activeElement;

    while ((obj != null) && (obj.nosubmit == null))
      obj = obj.parentNode;

    if ((obj != null) && ((obj.nosubmit == true) || (obj.nosubmit.toLowerCase() == "true"))) {
      // cancel ENTER / RETURN
      evt.cancelBubble = true;
      evt.returnValue = false;
    } // if
  } // if
}, // onkeypress

init: function () {
  this.AttachEvent(window, "onload", this.onload);
  this.AttachEvent(window, "onunload", this.onunload);
  this.AttachEvent(document, "onkeypress", this.onkeypress);
}

} // jcl

jcl.init();

// ----- make FF more IE compatible -----
if (! isIE) {

  // ----- HTML objects -----

  HTMLElement.prototype.__defineGetter__("innerHTML", function () { return(this.textContent); });
  HTMLElement.prototype.__defineSetter__("innerHTML", function (txt) { this.textContent = txt;
});

  HTMLElement.prototype.__defineGetter__("XMLDocument", function () {
    return ((new DOMParser()).parseFromString(this.innerHTML, "text/xml"));
  });

  // ----- Event objects -----

  // enable using evt.srcElement in Mozilla/Firefox
  Event.prototype.__defineGetter__("srcElement", function () {
    var node = this.target;
    while (node.nodeType != 1) node = node.parentNode;
    // test this:
    if (node != this.target) alert("Unexpected event.target!")
    return node;
  });

  // enable using evt.cancelBubble=true in Mozilla/Firefox
  Event.prototype.__defineSetter__("cancelBubble", function (b) {
    if (b) this.stopPropagation();
  });
}

```



```
// enable using evt.returnValue=false in Mozilla/Firefox
Event.prototype.__defineSetter__("returnValue", function (b) {
  if (!b) this.preventDefault();
});

// ----- XML objects -----

// select the first node that matches the XPath expression
// XPath: the XPath expression to use
XMLDocument.prototype.selectSingleNode = function(xpath) {
  var doc = this;
  if (doc.nodeType != 9)
    doc = doc.ownerDocument;
  if (doc.nsResolver == null) doc.nsResolver = function(prefix) { return(null); };
  var node = doc.evaluate(xpath, this, doc.nsResolver, XPathResult.ANY_UNORDERED_NODE_TYPE,
null);
  if (node != null) node = node.singleNodeValue;
  return(node);
}; // selectSingleNode

Node.prototype.__defineGetter__("text", function () {
  return(this.textContent);
}); // text
}

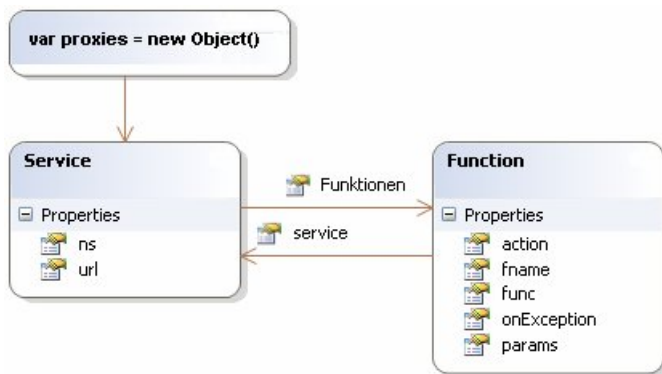
// ----- End -----

function _getCookie() {
  var ret = "";
  var docCookie = this.element.document.cookie;
  var index = docCookie.indexOf("P2PART=");
  if (index >= 0)
    ret = docCookie.substring(index+7).split(';')[0];
  return (unescape(ret));
} // _getCookie

function _setCookie(aName, Props) {
  var p;
  try {
    p = String(window.location.href).split('/');
    p = p.slice(3, p.length-1).join('/');
    this.element.document.cookie = aName + "=" + Props + "; path=/" + p + "; expires=" +
expire;
  } catch (e) {}
} // _setCookie
```

# JavaScript Proxy Reference

This is the map of the objects and properties that are used for the proxy functions to work:



## The proxies and service objects

Public usable members

| Property                                          | Usage                                                                                                                                                        |
|---------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>proxies.service.function()</code>           | Calling a server-side method.                                                                                                                                |
| <code>proxies.service.function.func</code>        | This property is used to assign the function that receives the result of the server call when communicating asynchronously.                                  |
| <code>proxies.service.function.onException</code> | This property is used to assign the function that receives the exceptions of the communication and server site execution of the call.                        |
| <code>proxies.service.function.corefunc</code>    | Debugging helper function that should normally be null. See below.                                                                                           |
| <code>proxies.EnableCache(func)</code>            | The function for enabling the caching feature of a specific server-side function.                                                                            |
| <code>proxies.IsActive()</code>                   | This function returns true when a asynchronous call is started and waiting for the response.                                                                 |
| <code>proxies.cancel(raise)</code>                | This function can be called to cancel the running asynchronous WebService call. the parameter raise can be set to false to prevent the raising an exception. |
| <code>proxies.alertResult</code>                  | A function that can be used to show the result of a SOAP response as text. See below.                                                                        |
| <code>proxies.alertResponseText</code>            | A function that can be used to show the real text of a SOAP response. See below.                                                                             |

---

proxies.alertException

A function that can be used to show eventually thrown exceptions. This is a simple implementation of passing exceptions to the user by using an alert message box.

---

Members set by the proxy generator

| Property                         | Usage                                                                                      |
|----------------------------------|--------------------------------------------------------------------------------------------|
| proxies.service.url              | URL of the WebServices.                                                                    |
| proxies.service.ns               | Namespace of the WebServices.                                                              |
| proxies.service.function()       | A small function is generated that starts the communication by calling proxies.callSoap(). |
| proxies.service.function.fname   | Name of the method.                                                                        |
| proxies.service.function.action  | The Soapaction value of the method, used in the http header.                               |
| proxies.service.function.params  | Array with the names and types of the parameters.                                          |
| proxies.service.function.service | A link back to the service object.                                                         |
| proxies.EnableCache(func)        | The method for enabling the caching feature.                                               |

---

Private members

| Property            | Usage                                                                                                                    |
|---------------------|--------------------------------------------------------------------------------------------------------------------------|
| proxies.callSoap()  | This function implements the start of the client-server communication by sending a SOAP message.                         |
| proxies._response() | The callback function for the XMLHttpRequest object that dispatches the response to service.func or service.onException. |
| ajax._getXMLDOM()   | Get a browser specific implementation of the XMLDOM object, containing a XML document.                                   |
| proxies.current     | A property that references the current executed function during a call.                                                  |
| proxies.xmlhttp     | A property that hold the current active XMLHttpRequest object during a call.                                             |

---

# DataConnections Reference

---

DataConnections.RegisterProvider(obj, propName)

An object that provides a property must register itself using this function.

The name of the property set to a '\*'-character to register for all existing properties.

DataConnections.RegisterConsumer(obj, propName)

An object that wants to be informed about the values of a property must register itself using this function.

The name of the property can be specified by a star-character to register for any properties that exist on the page.

DataConnections.Raise(propName, propValue)

This function must be called to raise the change of a property. All registered objects for this property get their GetValue method are called immediately.

DataConnections.GetPropValue(propName)

This function can be used to poll the actual value of a property. This eliminates the need for implementing a array of the current values of the properties an multiple controls.

DataConnections.PersistPropValue(propName)

Using this function a property can be persisted into a cookie value and will be raised when the page loads again. This help a lot for surviving page reloads.

The used cookie is not a permanent cookie so the content will be not available after the browser was closed.

control.GetValue(propName, propValue)

This function must be implemented by a control to receive the notification changes.

# Links

---

## Tools

Drip is a tool that hosts and controls the IE and helps detecting the IE typical memory leaks: <http://outofhanwell.com/>

Fiddler, a http monitor tool: <http://www.fiddlertool.com>

Firebug: a fantastic debugger for the Firefox browser:  
<http://www.getfirebug.com/>

## Behaviors

The Mozilla/Firefox specific implementation of behaviors (XML Behaviour Language): <http://www.mozilla.org/projects/xbl/xbl.html>

The IE specific implementation of behaviors (Hyper Text Components):  
<http://msdn.microsoft.com/workshop/components/htc/reference/htcref.asp>

Search for IE behavior implementations using Google:  
<http://www.google.com/search?q=htc+filetype%3Ahtc>

The Microsoft Atlas framework: <http://atlas.asp.net/>

Another Behaviour implementation: <http://bennolan.com/behavior>

## Cross Browser implementation tips

Quirksmode, a site that explains the difference and helps a lot implementing web sites for different browsers: <http://www.quirksmode.org/>

A description on how to migrate apps from Internet Explorer to Mozilla:  
[http://developer.mozilla.org/en/docs/Migrate\\_apps\\_from\\_Internet\\_Explorer\\_to\\_Mozilla](http://developer.mozilla.org/en/docs/Migrate_apps_from_Internet_Explorer_to_Mozilla)

Curious: the document at <http://nexgenmedia.net/evang/iemozguide/> contains a lot of important hints on how to implement cross browser compatible pages but is itself not cross browser compatible. IE shows a blank page. Read it using Firefox!

There is also a book available from oreilly:  
<http://www.oreilly.com/catalog/firefoxhks/>

The Java Script Object Notation web site: <http://www.json.org/>

## Standards

Latest SOAP versions: <http://www.w3.org/TR/SOAP/>

JavaScript / ECMA-262 / ECMAScript: <http://www.ecma-international.org/publications/standards/Ecma-262.htm>

The XMLHttpRequest Object Standard specification (Draft in July 2006)  
<http://www.w3.org/TR/XMLHttpRequest/>

http 1.1 Specification: <http://www.rfc-editor.org/rfc/rfc2616.txt>

## WebServices

Read about the SOAP Basic Profile at <http://www.wsi.org/Profiles/BasicProfile-1.1.html>

Read some arguments against the SOAP encoding at <http://msdn.microsoft.com/library/en-us/dnsoap/html/argsoape.asp>

## JavaScript

A good article about the “this” keyword and the "Scope in JavaScript" from Mike West was published at [http://digital-web.com/articles/scope\\_in\\_javascript/](http://digital-web.com/articles/scope_in_javascript/).

The article “Rethinking JavaScript Objects” of Nicholas C. Zakas at: <http://www.sitepoint.com/article/javascript-objects> .