Rich Internet Clients – First Steps with AJAX

By Thomas Maul, General Manager, 4D Germany.
TN 06-05

Abstract

This Tech Note gives basic background information on Ajax, describes the basic steps to implement Ajax-based designs and explains how to do it with 4D. Ajax relies on heavy use of XML and 4D 2004's rich XML features allow fast development of Ajax based solutions.

Introduction

While most Internet applications have a simple user interface a new generation of Web applications has recently appeared. These Web applications use a rich interface that mimics a normal desktop application. Google (Google Mail, Google Map or Google Suggest) or Flickr (http://flickr.com/) are some examples.

These Web pages use features like Type Ahead, Drag & Drop and Animations combined with asynchronous communications with the Web server. Unlike "normal" Web pages they do not submit entries to the server and reload a whole new page; only small parts are sent and replaced inside the existing page, which allows for faster and more user-friendly clients.

This technique is called Asynchronous JavaScript and XML (**Ajax**) and involves a combination of several existing standards. Although only very few Web sites are using this technique, it is not a new idea. There was a session during 4D Summit 2004 in New Orleans about Ajax, named "XMLHTTPRequest" (explaining how to use Ajax for Dashboard). This is the first 4D Tech Note on the subject.

What is Ajax?

Wikipedia (http://en.wikipedia.org/wiki/Wikipedia) has the following definition for Ajax:

Asynchronous **J**avaScript **A**nd **X**ML, or **Ajax** (pronounced as a word, not as individual letters), is a Web development technique for creating interactive Web applications using a combination of:

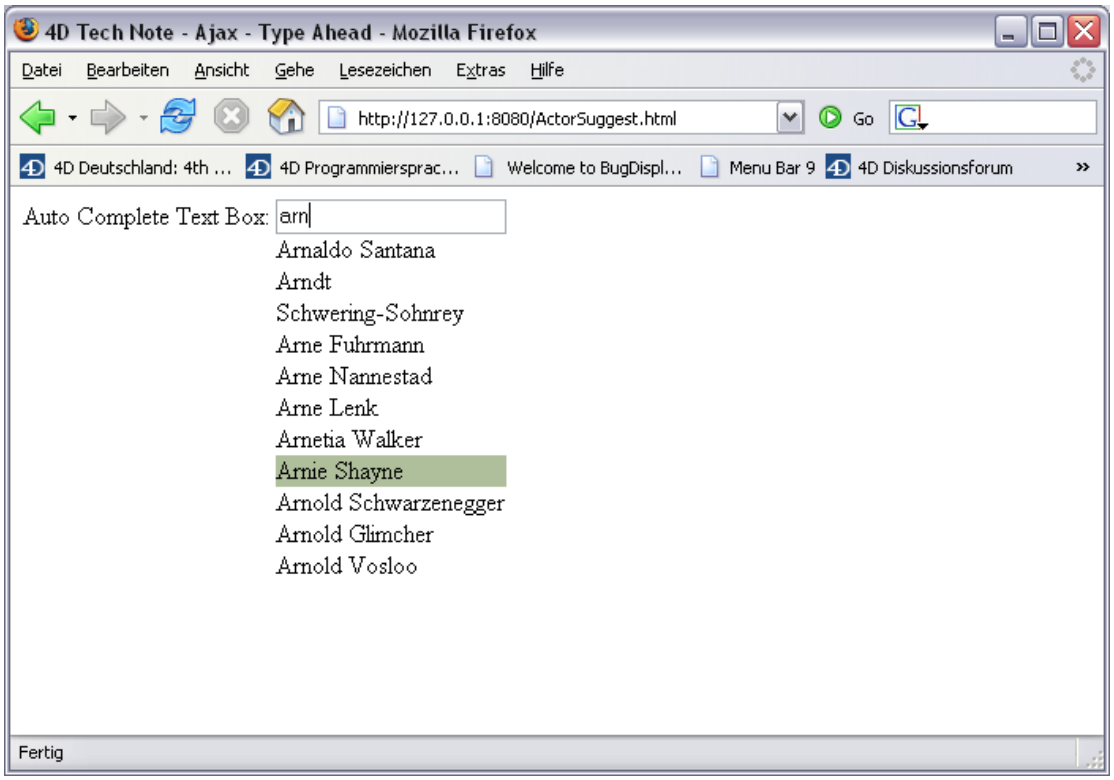
- XHTML (or HTML) and CSS for marking up and styling information. (XML is commonly used, although any format will work, including preformatted HTML, plain text, JSON and even EBML).
- The Document Object Model manipulated through JavaScript to dynamically display and interact with the information presented
- The XMLHttpRequest object to exchange data asynchronously with the Web server. In some Ajax frameworks and in some situations, an IFrame object is used instead of the XMLHttpRequest object to exchange data with the Web server.

Some examples...

This definition might sound very complex and difficult. It is much easier to understand the power of Ajax by simply trying some examples. Start the example database, either on Mac or Windows, with 4D 2004. The example is based on the "Video Library" example (installed with the 2004 product CD, folder "Application Samples"). The database contains approximately 1500 movies and 13000 actors.

Click on the "Launch the Browser" button to start a Web browser and connect to the local 4D Web server (or enter http://127.0.0.1:8080 to connect manually). Note that the Web server will run in demo-mode if a 4D 2004 Web Server license is not installed.

Click the "Type Ahead: Actor Suggest" link. This loads the "ActorSuggest.html" page:



Start to type a first or last name; try your first name or the name of your favorite movie actor. While a name is typed the name, the system will start to search and show possible matches below the text box. Note: the browser must support JavaScript to do this. If JavaScript is disabled Type Ahead will not work. Notice that the Type Ahead function is quite fast. As characters are entered the suggestion list quickly changes. The system searches for first names and displays the first 10 actors. If there are less than 10, it also searches for last names that match what has been typed. For example, try "cu" or "cur". Notice the speed. There are 13000 actors in the database. Note: 4D needs to load the index into the cache before the search speed will be maximized. Thus one or two searches will need to be executed before full speed is achieved.

Next try using Up/Down Cursor keys to select a name. The Enter key inserts the full name. The mouse may also be used to navigate.

If Firefox is installed with the "Live HTTP Headers" extension, open the "Live HTTP Headers" dialog and observe the headers as changes are made to the "Auto Complete Text Box". Note: the "Live HTTP Headers" extension for Firefox may be downloaded for free from:

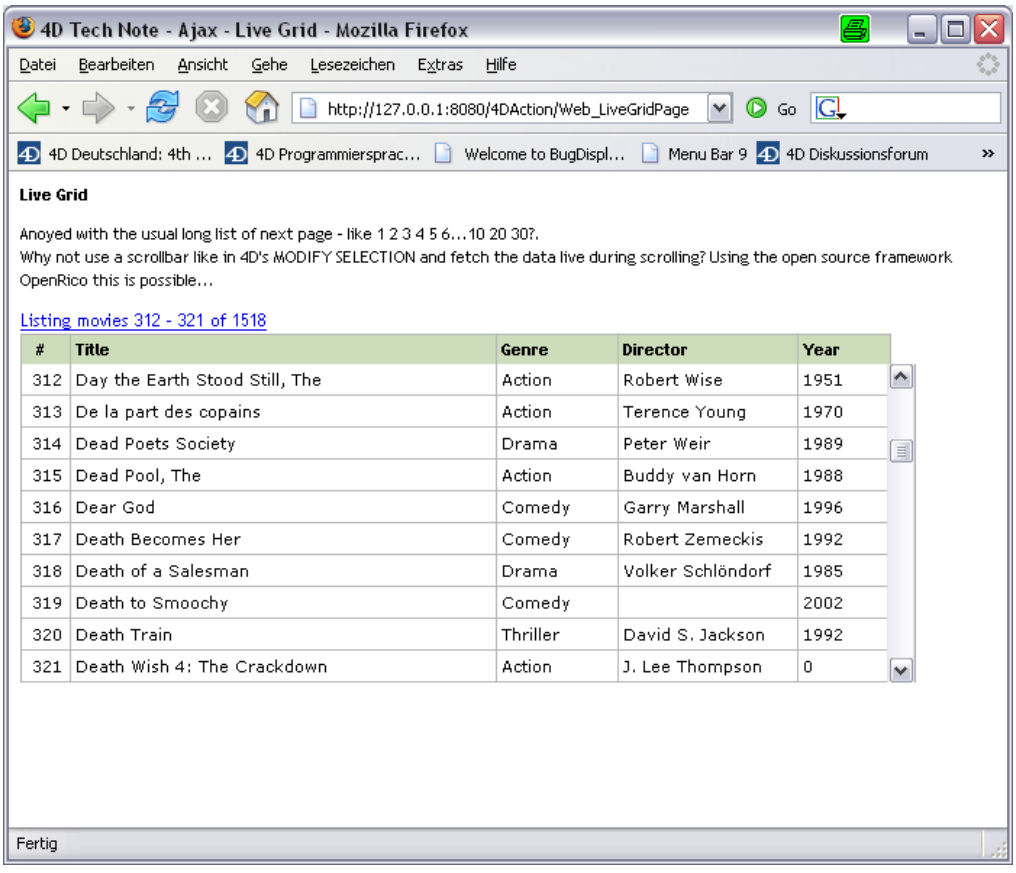
http://livehttpheaders.mozdev.org

This extension is very useful for debugging Web projects and is a must-have for Ajax development.

Notice that while text is entered there are invisible connections sent from the browser to the 4D Web server with the current text box content and 4D answers with the list of possible matches. Notice that the client uses its own cache to avoid additional requests if the data is already available. To test this, try typing "alc". There are only 4 names that match that name fragment. Entering "alca" or "alco" will not send another request to the server because the result is already known.

Impressed? Ok, so try the next example. Use the Back button from the browser. Notice one click is all it takes to go back to the main home page! Even if several searches were made, all communication done for the Type Ahead search was done without changing the URL of the Web page, so the browser thinks it is still on the same page.

Click the "Live Grid" link to open the next example:



Note the scrollable area.

Many common Website designs involve generating links to handle multiple pages of data, e.g.:

<Previous page> 1 2 3 4 5 6 7 8 9 10 20 30 40 <next page>

This design is so prevalent that it is often used it without thinking.

On the other hand, 4D developers will be very familiar with the concept of MODIFY SELECTION, which 4D has featured for over 20 years. Additionally, since 4D Server 1.0 (1992), the user interface of 4D Client is designed to show only the visible part of the selection and only the visible part is transferred over the network. The user can then use a scrollbar to browse through the selection.

Would it not be nice to have a similar user interface in a Web application? Ajax allows this!

Try to move the scrollbar on the "Live Grid" page. Depending upon the speed of the client computer, the redraw of the new page will be instant or have a short delay. While the scrollbar is moved, the browser requests the data from 4D dynamically, all without altering the URL of the Web page. Also try the scrollbar up/down buttons.

If the client/server communication is observed with "HTTP Live Headers" notice that the server communication is highly optimized. The client requests the first 70 movies (even though only 10 are displayed). This allows scrolling without additional requests. If the scrollbar is directly moved, the next 70 records are requested. If there is no user action for the 1 second, the 70 previous records are requested in order to allow scrolling up, using the cache.

The Live Grid page allows sorting by column heading as well. Try clicking on "Year" or "Director". A second click will change the sorting order.

How does Ajax work?

Ajax is a combination of several technologies.

First, and most important, is JavaScript. JavaScript controls all actions on the client (Web browser). The Ajax design concept will not work without JavaScript. However, this is also the major disadvantage of Ajax. If JavaScript is disabled, nothing will work. Because JavaScript may be disabled for security reasons it may be necessary to have a non-Ajax user interface as a backup solution.

The Document Object Model (DOM) allows for the manipulation of objects, like a cell in a table (the LiveGrid example uses a table with 10 rows; the JavaScript code replaces the contents of the table cells to emulate scrolling).

Cascading Style Sheets (CSS) allows for manipulating the appearance of, creating, moving and resizing objects on the fly, like in the Type Ahead example.

The last important technology is the "XMLHTTPRequest" object, which was introduced with Microsoft Internet Explorer 5.5 and Safari 1.2. This is the key element that allows asynchronous requests to be sent from a Web browser and handled in the background without disturbing the user on slow internet lines.

This Tech Note will not cover JavaScript, DOM nor CSS concepts in detail (with exception of a very basic JavaScript chapter). There are many training books and Web sites available on these subjects.

JavaScript

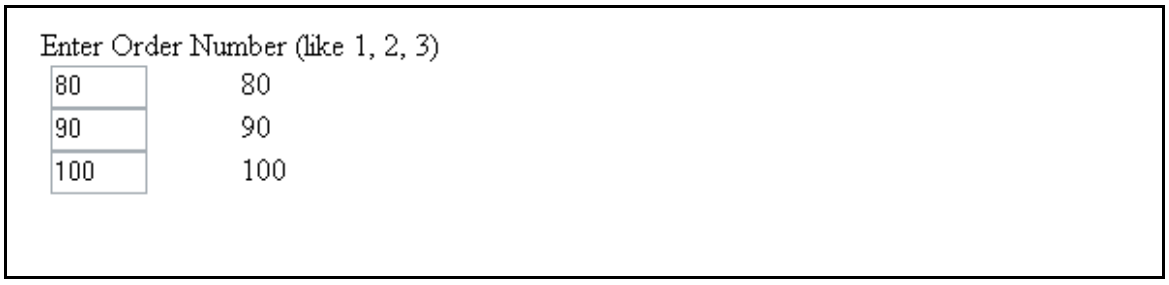
You may want to skip this section if you have JavaScript and DOM experience.

JavaScript is a simple script language. It is related to Java only in syntax (and even then it is not as strict). For more information see Wikipedia:

http://en.wikipedia.org/wiki/JavaScript

This page contains basic steps for learning JavaScript and a collection of links to tutorials, guides, etc.

A simple JavaScript example is contained in the sample database. Use the back button in the Web browser (or connect again to http://127.0.0.1:8080, 4D must be running with the demo database) and click on "Simple HTML":



In the text boxes enter some numbers (or characters). As soon you leave a box the entered value is copied into another (non-enterable) text box. This behavior very similar to using the "On Data Change Event" in 4D.

Right-click on the Web page and view the source (or open the file "SimpleHTML.html" in the folder "WebFolder" with a text/html editor). The HTML code (body part) contains a form and, inside the form, a table. The table contains 4 rows, one for the header and 3 for the text boxes. Each row contains two cells. Here is the HTML for the cells:

```
<td width="103" colspan="2" class="in_label" nowrap>
        <input name="artNew1" type="text" class="inp" style="width: 50px;
                margin-left: 5px;" ID="Input1"
                onChange="setTextValue('Text1', this)">
</td>
<td width="450" class="in_label" ID="Text1">
</td>
```

Note that each cell defines an ID. ID's are the key element of this concept as they allow to direct access any object inside the page via JavaScript.

Aside from the visual information for the text box, the first cell also contains code:

```
onChange="setTextValue('Text1', this)"
```

In 4D this could be thought of as calling the "setTextValue" method for the "On Data Changed" event in an object method:

```
$event:=Form event
Case of
        : ($event:=On Data Change )
                setTextValue("Text1", Self)
End case
```

The JavaScript method gets two parameters, a text constant with content "Text1" and a pointer to the current object (which is called "Self" in 4D and "this" in JavaScript).

The first parameter "Text1" is the ID of the cell to be changed. In the header of the HTML page is the definition of the JavaScript method:

```
<script type="text/javascript">
function setTextValue(id, obj)
{
        document.getElementById(id).innerHTML=obj.value;
}
</script>
```

The function body is only one line. It accesses the document (remember DOM = Document Object Model) and inside the document it looks for the element using the ID. This concept allows access to all existing objects in the document as long they have an unique ID. The attribute "innerHTML" allows

assigning any value, including HTML code. The second parameter "obj" is a kind of pointer on the calling object, which contains the new value.

Make sure the JavaScript us understood before continuing with the following chapters. Without basic JavaScript knowledge the material that follows may be difficult to understand, so an introduction to JavaScript may be required before moving on.

Debugging JavaScript problems can be difficult. For example, sometimes the browser appears to do nothing. This is the usual reaction on a programming error; the execution will simply stop. Firefox includes a tool named "JavaScript Console" which lists all execution errors. This is a big help during development. Also take a look at the Venkman JavaScript Debugger, a free debugger allowing to trace JavaScript and read/write variables:

http://www.mozilla.org/projects/venkman/

The XMLHTTPRequest Object

The XMLHTTPRequest object was originally developed by Microsoft. It has been available since Internet Explorer 5.0 as an ActiveX object accessible via JScript, VBScript, or other scripting languages supported by the browser. Mozilla contributors then implemented a compatible native version in Mozilla 1.0. This was later followed with an implementation by Apple in Safari 1.2 and Opera Software in Opera 8.0. This object allows retrieving data from a Web server as a background operation. While the data is often XML, it can be any text based data.

This object is explored in greater detail in the 4D Summit 2004 session notes on pages 377-381.

A minimal XMLHTTPRequest session looks like this:

```
var oXMLHTTP = new ActiveXObject("Microsoft.XMLHTTP")
var sURL = "/csutomerIDcheck.xml?username="+custid;
oXMLHTTP.open("POST", sURL, false);
oXMLHTTP.send();
alert(oXMLHTTP.responseText);
```

This minimal code may or may not work as it uses a Microsoft Object not available on any other browser.

Mozilla uses another object, so we need to try to check which object is available in order to support both browsers:

```
if (window.XMLHttpRequest){
 oXMLHTTP =new XMLHttpRequest();
 } else if (window.ActiveXObject){
 oXMLHTTP =new ActiveXObject("Microsoft.XMLHTTP");
```

```
}
```

The third parameter of the "XMLHTTPRequest.open" method is the async flag, which tells the browser whether or not to wait until the server has answered a request. In order to operate in asynchronous mode code will need to be added to handle the responses from the server (this code is covered later).

Exception handling in asynchronous mode can be a difficult job. The good news is that many examples for this already exist. Simply use Google to search for "XMLHttpRequest". The Apple Web site contains an example (with support for Microsoft Internet Explorer on Windows):

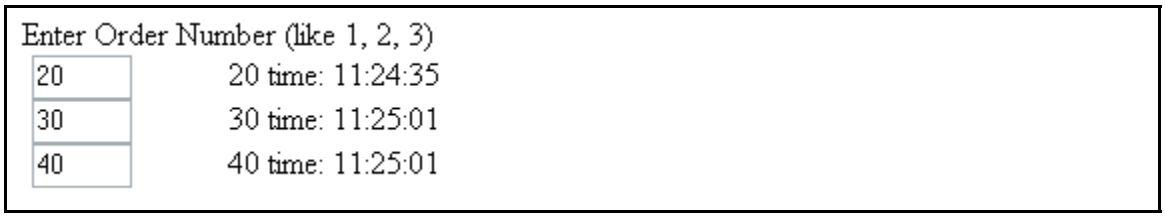
http://developer.apple.com/internet/webcontent/xmlhttpreq.html

In order to develop more complex dynamic Web applications it is helpful to use a JavaScript Framework. These Frameworks, like "Prototype" or "Rico", already contain code to handle the XMLHttpRequest object. See the "Type Ahead" or "Live Grid" examples for more information.

The Rich Internet Client

The next step is to put it all together using JavaScript, the XMLHTTPRequest object, and 4D.

Take a look at the "Simple Ajax" example:



This example is similar to the "Simple HTML" example, but uses the XMLHTTPRequest to talk to 4D.

Enter any number or character code (e.g. simulating order numbers) in the text boxes. As soon you leave the box (e.g. by typing tab or clicking in the next box) the entered text plus the current time is displayed. This data comes from 4D.

Open the HTML file "SimpleAjax.html" (in the folder "Webfolder") and the file "SimpleHTML.html" with a text or HTML editor, so both can be compared.

The HTML body of the two files is identical (except for comments). The difference is in the HTML header.

In the "SimpleHTML.html" example without Ajax the setTextValue function was defined as:

```
function setTextValue(id, obj)
        {
        document.getElementById(id).innerHTML=obj.value;
        }
```

The Ajax version has two functions:

```
function setTextValue(id, obj)
        {
        var url = "/4DAction/Ajax_SimpleRequest/"+obj.value;
        var loader=new net.ContentLoader(url, ActionCompleted);
        loader.returntarget = id;
        }
function ActionCompleted() {
        document.getElementById(this.returntarget).innerHTML=
                this.req.responseText;
        }
```

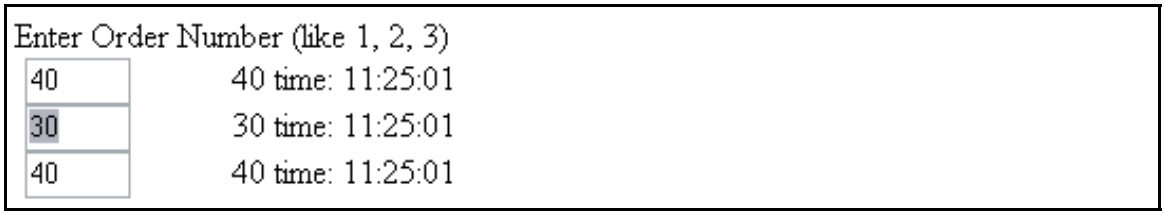
The first line of the function "setTextValue" creates a variable and assigns an URL to it. The URL contains a 4D method named "Ajax_SimpleRequest", which will be called using 4DAction, and the content of the text box as a parameter.

The second line defines the object that will handle all of the Ajax communication. This object is defined in the file "scripts/ContentLoader.js". This line creates the object and passes the URL and the name of a function ("ActionCompleted") to be called as soon the server answers.

The last line adds a variable to this object to remember the id we want to change.

The function "ActionCompleted" is called as soon the Web server answers. The data is stored in this.req.responseText.

This example shows a possible problem. Wait a minute and then enter the same value again (in the same text box or a different one). Note that the time is not updated:



This happens because the Web browser did not send the request to the 4D Web server. Instead it used the cache. This can be verified with Live HTTP

Headers. If the returned answer from the server is the same (e.g. a product name), this can be seen as feature to avoid network traffic. If the answer is likely to be different (e.g. amount of products in stock), this is a problem. The next chapter explains how to solve that.

Take a look at the 4D code for "Ajax_SimpleRequest" in the sample database (click the "Goto Design Mode" button to get into Design Mode):

```
$request:=Substring($1;2)    ` parameter contains entered text, starting with "/"
$answer:=$request+" time: "+String(Current time)    ` calculate the answer
SEND HTML TEXT($answer)    ` send the answer
```

Set a breakpoint at the first line. Switch back to the Web browser and enter some new values in all three text boxes. The debugger in 4D will stop on the breakpoint. This simulates a "very slow" Web server; 4D will not answer until execution continues. Notice that the text beside the box did not change. However, even if the Web server is slow (or does not answer at all), the client is still useable. The application is working, only the back end responses are missing, thanks to the asynchronous calls.

Switch back to 4D and click the "No trace" button (green triangle) in the debugger (note that there will be one debugger window for each request sent). In the browser window the responses should appear. The variable "loader.returntarget" identifies which object the response belongs too.

The Web Browser Cache

It was shown before that the Web browser cache can prevent requests from being sent to the server using asynchronous requests. There are several ways to handle this problem. Doing a Google Search for "Ajax browser cache" shows several discussions and possible solutions.

The simplest solution is to send the request as an HTML FORM using a POST command. In this case the browser will not use the cache and always sent the request to the Web server.

Try the "Simple Ajax – using form" example. Send a request, wait a minute and use the same value again. The updated time shows that the browser cache is not used.

In this example the 4D code is slightly modified:

```
ARRAY TEXT($name;0)
ARRAY TEXT($values;0)
GET WEB FORM VARIABLES($name;$values)
If (Size of array($values)>0)
        $request:=$values{1}
        $answer:=$request+" time: "+String(Current time)
```

```
        SEND HTML TEXT($answer)
Else
        $answer:="Server communication error - no parameter passed"
        SEND HTML TEXT($answer)
End if
```

The command GET WEB FORM VARIABLES is used to retrieve the parameter and the current time is appended to it.

Another possible solution to the Web browser cache problem is to modify the HTTP header and change the expire date. This solution is used in the more advanced examples presented in this Tech Note ("Actor Suggest" and "Live Grid").

Examples In Depth...

Double Combo Box

The "Simple Ajax" examples are very simple. Still they implement the basic functionality of an Ajax design and demonstrate a drastically improved user experience.
The next example, "Double Combo Box", needs more JavaScript code and shows how generic code can be used.

This example is based on chapter 9 from the book "Ajax in Action", which is available as a free download from:

http://www.manning.com/books/crane/chapters

This chapter of the book explains in detail how to build a combo box that modifies the content of a second combo box. Imagine an online reservation system to rent videos. The first combo box selects the movie category. The second shows the available movies after asking the server to check the availably.

Open the "Double combo box" example in the Web browser. Note that the second combo box is empty. Use the first one to select a category, like Fantasy. The second combo box will show the available movies.

This example also shows a possible problem. Select the category "Action" and immediately open the second combo box. It will still show the old content. If the combo box is kept open it will display the correct values after 2 or 3 seconds. This delay occurs because there are nearly 500 movies for the "Action" category and it takes the JavaScript a while to build such a large combo box. This is an example of a bad use of a combo box (a list box would be better) but it is impressive that it works and shows the power of asynchronous code.

The 4D code for this example is very similar to the previous example. Again GET WEB FORM VARIABLES is used to retrieve the parameters. Because the reply contains many more items, XML is used to send the data. Please note while the "x" in Ajax stands for XML, this is the first time XML has been used in the examples. As mentioned before it is possible to answer with any kind of text-based data; plain text, ready prepared HTML, XML, encoded pictures, etc. The DOM commands of 4D 2004 are used to prepare the XML document.

Here is the HTML for the "Double Combo Box" example:

```
<head>
 <script type="text/javascript"  src="ricoscripts/prototype.js"></script>
 <script type="text/javascript"  src="ricoscripts/rico.js"></script>
 <script type="text/javascript"  src="ricoscripts/doubleCombo.js"></script>
 <script type="text/javascript"  >
  function injectComponentBehaviors()  {
  var doubleComboOptions = { };
  new DoubleCombo( 'Group',
      'Movies',
      '/4DAction/Ajax_DoubleCombo',
      doubleComboOptions );
  }
  </script>
</head>

<body onload="injectComponentBehaviors()">
...
<select name="Group"  ID="Group">
          <option  value="-1">Pick  a categorie</option>
          ...
</select>
<select name="Movies"  ID="Movies"  style="width:200px">
...
</body>
```

Note that the JavaScript code is based on the frameworks mentioned previously:

Read over with the HTML body. Note that the two combo boxes (noted by the <select> tag) do not contain any JavaScript! This allows easy integration into existing Web pages without any modification of the HTML page. The code in the header "injects" the functionality. The object DoubleCombo (defined in the function "injectComponentBehaviors") does the main job; the first parameter sets the main combo box; the second parameter sets the secondary combo box (the objects are identified using the ID tags); the third parameter is the URL to be called; the last parameter allows setting options which are not needed at this point.

This concept makes the enhancement of existing Web pages very easy. For detailed information about the JavaScript code read chapter 9 of "Ajax in Action". Note that some minor modifications to "DoubleCombo.js" were made in order to support Safari.

Type Ahead

The design of this example was explained previously. Take a look the 4D code.

The 4D method "AjaxActorSuggest" handles the back-end part. The code is very similar to the combo box example. The parameters are retrieved using GET WEB FORM VARIABLES and the response is built as an XML document.

The JavaScript code uses the same concepts as the "Double Combo Box". A generic function "injects" the feature into the HTML code. Take a look at "ActorSuggest.html". It is very similar to "DoubleCombo.shtml".

Live Grid

This is the most complex example in this Tech Note. It makes heavy usage of the OpenRico framework, which itself uses the Prototype framework.

To further explore the OpenRico framework take a look at:

http://ww.openrico.org

See specially the examples for graphical features, like animations. There is also a PDF explaining how to use the LiveGrid feature.

The basic steps are:

- Build an HTML table, make sure to use table ID's to identify the table and <DIV> statements to organize the table (container, header, body). The table may contain real data for the first rows, but also can be empty. The table must contain at least 1 additional row than later displayed.
- In Onload call Rico.LiveGrid to build the LiveGrid functionality, passing the name of the table, the number of rows and the URL of the back end. This object will redesign the table to the correct amount of rows, request the data from the server, build an internal cache, create a scrollbar, create the sort on header functionality…in fact it pretty much does all the work.
- Create a 4D method to build the backend.

The Rico.LiveGrid object allows the inclusion of a method to be called for each scroll, which allows for modifying other objects in the page. The "Live

Grid" example demonstrates this with a header showing the current scroll position. This header also contains an URL which can be used as a bookmark. The bookmark contains the current scroll position and the sorting (object and order).

The 4D method "Ajax_LiveGrid" handles the back end. The method is very similar to the "Combo Box" and "Type Ahead" examples in that it reads the parameters and builds a XML answer.

To see the content of the "Live Grid" XML answer, type this URL into the Web browser:

http://127.0.0.1:8080/4DAction/Ajax_LiveGrid?id=data_grid&page_size=70&offset=400&

Note that the XML answer contains HTML code. This is the preformatted part that will be inserted into the HTML table.

Ajax Resources

Even though Ajax is a combination of existing concepts, there were very few books specifically covering Ajax development available at the time this Tech Note was written. There are many Web sites devoted to the topic but, as the Ajax design is relatively new, the URL's often change so a list of resources is not given here.

There is an introduction to Ajax on the IBM website, which contains links to many other articles:

http://www-128.ibm.com/developerworks/web/library/wa-ajaxintro1.html

Wikipedia also has a list of Tutorials:

http://en.wikipedia.org/wiki/Ajax_%28programming%29

This Tech Note uses an object introduced in chapter 9 of the book "Ajax in Action" by Dave Crane (ISBN 1932394613), which can be downloaded from:

http://www.manning.com/books/crane/chapters

The JavaScript source can be loaded from the same page.

This book contains an introduction to JavaScript, CSS and DOM so it is a big help for starting Ajax development. It explains how to do things in detail and then shows how to create more generic useable objects. There is also an 18 minute movie on the website showing content and examples from the book.

Conclusion

This Tech Note gave basic background information on Ajax, described the basic steps to implement Ajax-based designs and explained how to do it with 4D. An example database was provided to illustrate the Ajax design in action.