

Rico LiveGrid Tutorial

The Rico JavaScript library provides a behavior for connecting an HTML table to a live data source via Ajax.

For a discussion on Ajax, see the Appendix A, What is Ajax?

Rico LiveGrid

The LiveGrid behavior takes an ordinary HTML table and

- Creates a scrollbar that becomes the live navigator for making Ajax data requests
- Connects it live to Ajax data responses
- Automatically populates the response data into the table cells
- Updates contents of the cells only to improve performance
- Employs data buffering and event compression strategies to improve performance

Overview of LiveGrid

Using the LiveGrid to make an HTML table become live with data is very straightforward. There are two basic steps.

1. Create your HTML table and give the table element a unique *id*.
2. Create a new Rico.LiveGrid object in the body's onload, passing the
 - a. HTML table's id
 - b. The data request handler URL
 - c. Additional options to customize its behavior

Before starting, make sure that you have included both prototype.js and rico.js in your HTML page. You should include these two library files in your <head></head> section like this:

```
<script src="scripts/prototype.js"></script>  
<script src="scripts/rico.js"></script>
```

Next, make sure that you are creating the Rico.LiveGrid object in the body onload.

Add the following code to your HTML <body> tag.

Rico LiveGrid Tutorial

```
<body onload="javascript:bodyOnLoad()" >
```

Then write your JavaScript `bodyOnLoad()` function to create the `Rico.LiveGrid`. In our LiveGrid – Data Table demo the body of our method would look like this:

```
var opts = { prefetchBuffer: true, onscroll: updateHeader }  
new Rico.LiveGrid ('data_grid', 5, 1000, 'getMovieTableContent.do',  
opts);
```

Don't worry about the details. They will be discussed later.

Data Grid Movies Demo

Lets use the Data Grid Movies demo from the openrico web site (<http://openrico.org/livegrid.page>)

The table shows that it has 950 movies and we are currently looking at rows 19-28.

Listing movies 19 - 28 of 950

#	Title	Genre	Rating	Votes	Year
19	Aliens	Action	8.2	64570	1986
20	Kill Bill: Vol. 1	Action	8.2	65566	2003
21	Braveheart	Action	8.2	92120	1995
22	All Quiet on the Western Front	Action	8.2	6806	1930
23	Wo hu cang long	Action	8.2	52043	2000
24	Salaire de la peur, Le	Action	8.2	2802	1953
25	The Adventures of Robin Hood	Action	8.1	7330	1938
26	Sin City	Action	8.1	22697	2005
27	Kill Bill: Vol. 2	Action	8.1	44831	2004
28	Kumonosu jo	Action	8.1	3309	1957

For this demo we built a server-side process to handle requests movie listings. It is called

When the scroller is scrolled, the LiveGrid behavior comes into play to make requests to this request handler and handles the responses that are returned by intelligently updating the table's contents on the fly.

Step 1: Create Your HTML Table

Create an HTML table and give it a unique *id*. The *id* will be used by the LiveGrid behavior to determine how to get updates to your table. In our example we also want a header. Currently we do this by creating an additional table that defines the header rows.

Rico LiveGrid Tutorial

Here is how it looks for the Movies Data Grid header table:

```
<table id="data_grid_header" class="fixedTable" cellspacing="0"
      cellpadding="0" style="width:560px">
  <tr>
    <th class="first tableCellHeader"
        style="width:30px;text-align:center">#</th>
    <th class="tableCellHeader" style="width:280px">Title</th>
    <th class="tableCellHeader" style="width:80px">Genre</th>
    <th class="tableCellHeader" style="width:50px">Rating</th>
    <th class="tableCellHeader" style="width:60px">Votes</th>
    <th class="tableCellHeader" style="width:60px">Year</th>
  </tr>
</table>
```

The next table we set up is the one that will contain our movie listings. In our example, we want to show 10 rows of data. We could create an empty table with 10 rows or we could pre-populate the data in the rows. We have chosen the latter and we have used Java Server Pages (JSP) to populate the table.

Rico LiveGrid Tutorial

```
<div id="viewPort" style="float:left">
<table id="data_grid"
  class="fixedTable"
  cellspacing="0"
  cellpadding="0"
  style="float:left;width:560px; border-left:1px solid
#ababab">
<logic:iterate id="movie"
  collection="<%= GetMovieTableContentAction.getMovies(request) %>"
  length="<%= "" + pageSize %>"
  type="org.openrico.demos.beans.Movie">

  <tr>
    <td class="cell"
      style="width:30px;text-align:center"><%=i+1%></td>
    <td class="cell"
      style="width:280px">
      <bean:write name="movie" property="title"/></td>
    <td class="cell" style="width:80px">
      <bean:write name="movie" property="genre"/></td>
    <td class="cell" style="width:50px">
      <bean:write name="movie" property="userRating"/></td>
    <td class="cell" style="width:60px">
      <bean:write name="movie" property="userVotes"/></td>
    <td class="cell" style="width:60px">
      <bean:write name="movie" property="year"/></td>

  </tr>
  <% i++; %>
</logic:iterate>
</table>
</div>
```

This just simply makes a request from a backend object to get a collection of movies and loop until all of the rows & cells for the HTML table are created. It does not matter how the table gets built, it just needs to be well formed HTML with the correct number of rows built.

Beta 2 change:

The extra div surrounding the table is required for beta 2.

This markup will be simplified for the final 1.1 version.

Step 2: Create LiveGrid Behavior

Think of the LiveGrid as being the handler for making data requests and populating the table from data responses.

Rico LiveGrid Tutorial

```
<body onload="javascript:bodyOnLoad()" >
```

Then write your JavaScript `bodyOnLoad()` function to create the `Rico.LiveGrid`. In our LiveGrid – Data Table demo the body of our method would look like this:

```
var opts = { prefetchBuffer: true, onscroll: updateHeader }  
new Rico.LiveGrid ('data_grid', 5, 1000, 'getMovieTableContent.do',  
opts);
```

In order to set up the LiveGrid though you need to pass to the LiveGrid constructor:

- The HTML table id (e.g., 'data_grid')
- The Request Handler URL (e.g., 'getMovieTableContent.do')
- The set of options for configuring the LiveGrid (e.g., prefetch and onscroll parameters)

We've already created the HTML table. So let's look at the Request Handler.

Step 2a. The Request Handler

Try this URL:

http://openrico.org/getMovieTableContent.do?id=grid_data&offset=0&page_size=2

You should get a response like this:

Rico LiveGrid Tutorial

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ajax-response>
  <response type="object" id='data_grid_updater'>
    <rows update_ui='true' >
      <tr>
        <td>1</td>
        <td convert_spaces="true"> Shichinin no samurai</td>
        <td>Action</td>
        <td>8.9</td>
        <td>31947</td>
        <td>1954</td>
      </tr>
      <tr>
        <td>2</td>
        <td convert_spaces="true"> The Lord of the Rings: The Return of the
King</td>
        <td>Action</td>
        <td>8.8</td>
        <td>103911</td>
        <td>2003</td>
      </tr>
    </rows>
  </response>
</ajax-response>
```

When requests are made for our movies in this demo, the `page_size` parameter is set to a larger number (usually larger than 10 to account for buffering).

The LiveGrid behavior processes this response to populate your table data contents during scrolling.

If you have read the Rico AjaxEngine Tutorial you might be wondering whether you have to register the request with the `registerRequest` method or register the HTML table element with the `registerAjaxElement` method. The answer is no. The LiveGrid behavior performs these steps for you.

When you are creating the response in your request handler you must set the content-type of the response header to `text/xml`. Also you will need to specify the xml version. Here is how the first couple of lines would look in Java Server Pages (JSP):

```
<% response.setHeader("Content-Type", "text/xml"); %>
<?xml version="1.0" encoding="ISO-8859-1"?>
```

And this is how they would look in PHP

Rico LiveGrid Tutorial

```
header("Content-Type: text/xml");  
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Understanding the ajax-response in Rico

Notice several important items about the Ajax response

First the response is wrapped in the tags `<ajax-response></ajax-response>`. Every Rico Ajax response must have this element as the root of the XML returned.

Second notice the response contained within the `ajax-response`. The response tags (`<response></response>`.) wrap the response content. An `ajax-response` in Rico is not limited to a single response. It can contain multiple responses. Each response is marked by the `<response></response>` set of tags.

Every Rico Ajax response must have the `<ajax-response>` element as its root and at contain at least one `<response>` element.

Third, notice the value of the `type` attribute. It is set to `object`. Also notice the value of the `id` attribute is set to the name of our HTML table ID plus the string `'_updater'`. This is what routes the response xml to the LiveGrid behavior to handle the data population .

Step 2c: Passing Optional Parameters

The LiveGrid accepts several options that can configure the behavior of the LiveGrid. Here is a list of the

available set of options.

Option	Meaning
<code>prefetchBuffer</code>	If set to false, there will not be an initial request for data. If set to true, data will be fetched when the LiveGrid is constructed.
<code>onscroll</code>	If set, is the name of a JavaScript function that will be called as the scrollbar is scrolled. This allows you to add custom behavior like tooltips, etc.
<code>onscrollidle</code>	If set, is the name of a JavaScript function that will be called when scrolling pauses or stops
<code>requestParameters</code>	A string of normal HTTP request parameters that you want to be passed to your request handler. For example, if we were doing a search we might set the string to <code>'query=flowers'</code> . During scrolling these request parameters are used in each request. Note that you can also set the request parameters after

Rico LiveGrid Tutorial

	construction with a call to <code>setRequestParams(String requestStr)</code>
--	--

Caveats, Issues, Bugs

Ok, here are some things to keep in mind currently.

- The LiveGrid expects all rows to be the same height. Set a style class on your rows and set the content to nowrap. Also truncate data as necessary.
- In order to initiate a request outside of the scrollbar you will need to make two calls back to back (I know we are going to fix this ☺)

```
myLiveGrid.fetchBuffer(0, false, true);  
myLiveGrid.requestContentRefresh(0, true);
```

You might proceed it with the request parameters that are unique. For example for a search that requires a 'query' parameter you might call:

```
myLiveGrid.setRequestParams('query=me');
```

- Scrolling one line at a time is a little odd. We are working on it. Some caveats with IE.
- We use the native scrollbar. Sometimes this bites us. We assume the scroller is less than 19px (with border the scroller setting in windows should be 17 pixels or less). We are working on a fix for this.
- We are still trying to decide the best type of feedback during scrolling. In the Rico Yahoo Search we use a tooltip and update the results header during scrolling. We are also experimenting with a busy icon in a non-annoying location that lets you know we are fetching data. Maybe it is just a small ball that subtly lights up or a firefox style twirling dial?

Rico LiveGrid Tutorial

Appendix A What is Ajax?

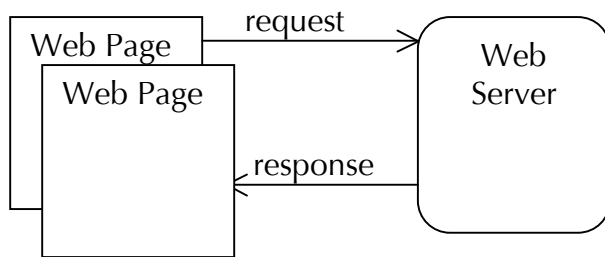
Wikipedia has the following definition for Ajax:

Traditional web applications allow users to fill out forms, and when these forms are submitted, a request is sent to a web server. The web server acts upon whatever was sent by the form, and then responds back by sending a new web page. A lot of bandwidth is wasted since much of the HTML from the first page is present in the second page. Because a request to the web server has to be transmitted on every interaction with the application, the application's response time is dependant on the response time of the web server. This leads to user interfaces that are much slower than their native counterparts.

AJAX applications, on the other hand, can send requests to the web server to retrieve only the data that is needed, usually using SOAP or some other XML-based web services dialect, and using JavaScript in the client to process the web server response. The result is more responsive applications, since the amount of data interchanged between the web browser and web server is vastly reduced. Web server processing time is also saved, since a lot of this is done on the computer from which the request came.

Comparing Ajax to Normal HTTP Mechanism

As stated above, the traditional way users interact with a page is to click a link (usually translates to an HTTP GET request) or click a submit button on a form (usually translates to an HTTP POST request). In the first case, a new page is requested and the browser refreshes with new content. In the second case, either a new page or the same page with modified values is returned.



In Ajax the request is made using the JavaScript function XMLHttpRequest. The request asks for a block of XML data (rather than a whole page to refresh) that the JavaScript code can handle in whatever way it sees fit.

For example, here are some ways the XML data could be interpreted:

- XML data that the JavaScript code parses to extract data. This data in turn is used to fill in field values or tables in a display.
- XML data that the JavaScript runs an XSLT process on to convert into HTML code to be inserted on the page somewhere

Rico LiveGrid Tutorial

- XML data that holds JavaScript that the JavaScript code evaluates to issue commands
- XML data that contains HTML code that can be placed inside another HTML element on the page

This list is not exhaustive. The point is – Ajax allows you to make a server side call outside of the normal page refresh cycle. The data that is returned can be defined in any manner that XML allows and the way it is interpreted is open to the application's determined usage of this data.