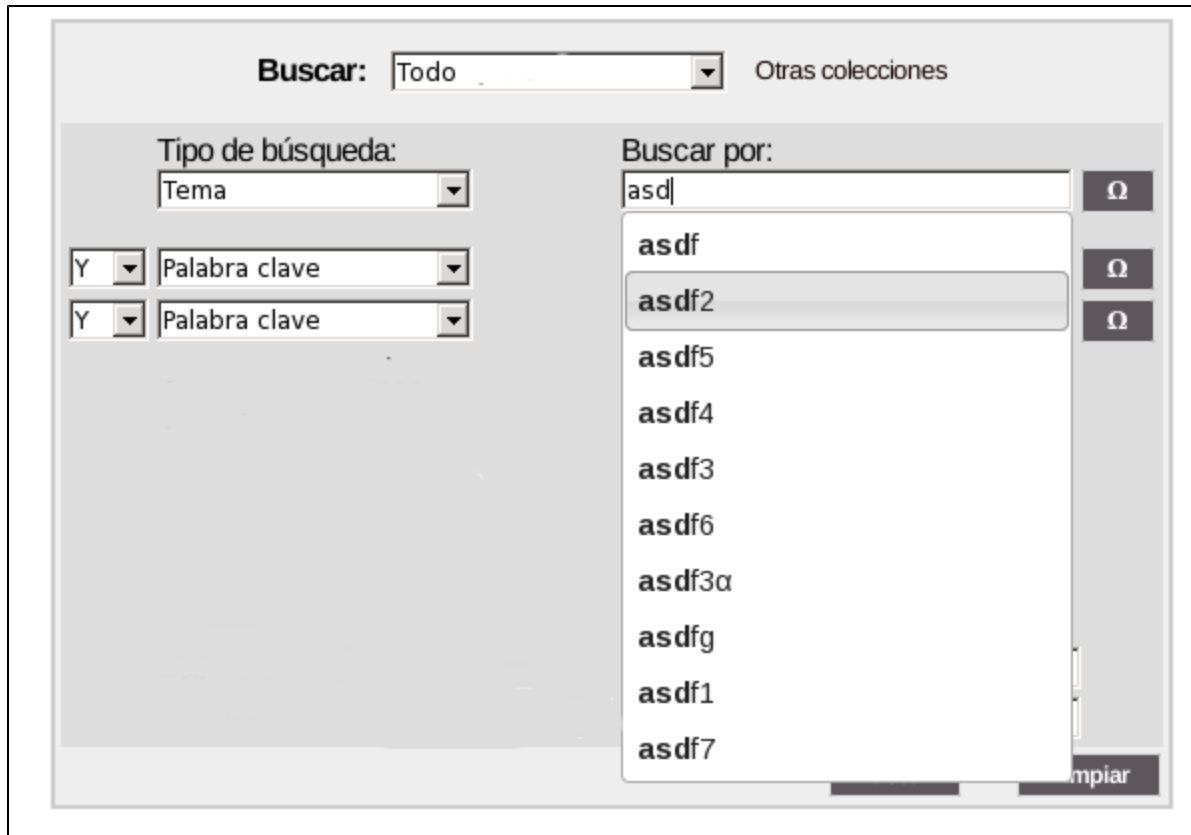


# Autocomplete functionality on JSPUI advanced search form

Using jquery and jquery-ui libraries we'll try to setup the autocomplete functionality on every textbox of advanced search page, we'll take advantage of Discovery Solr cores to add an additional search core for JSPUI.

*Note:* This implementation will display only words or partial words when using autocomplete functionality. JSPUI interface indexes fields by splitting them into words and removing some suffixes, this is bad for searching full text terms like author names. In a future section I will describe how to set custom text indexers like "full text" to our fields in JSPUI.

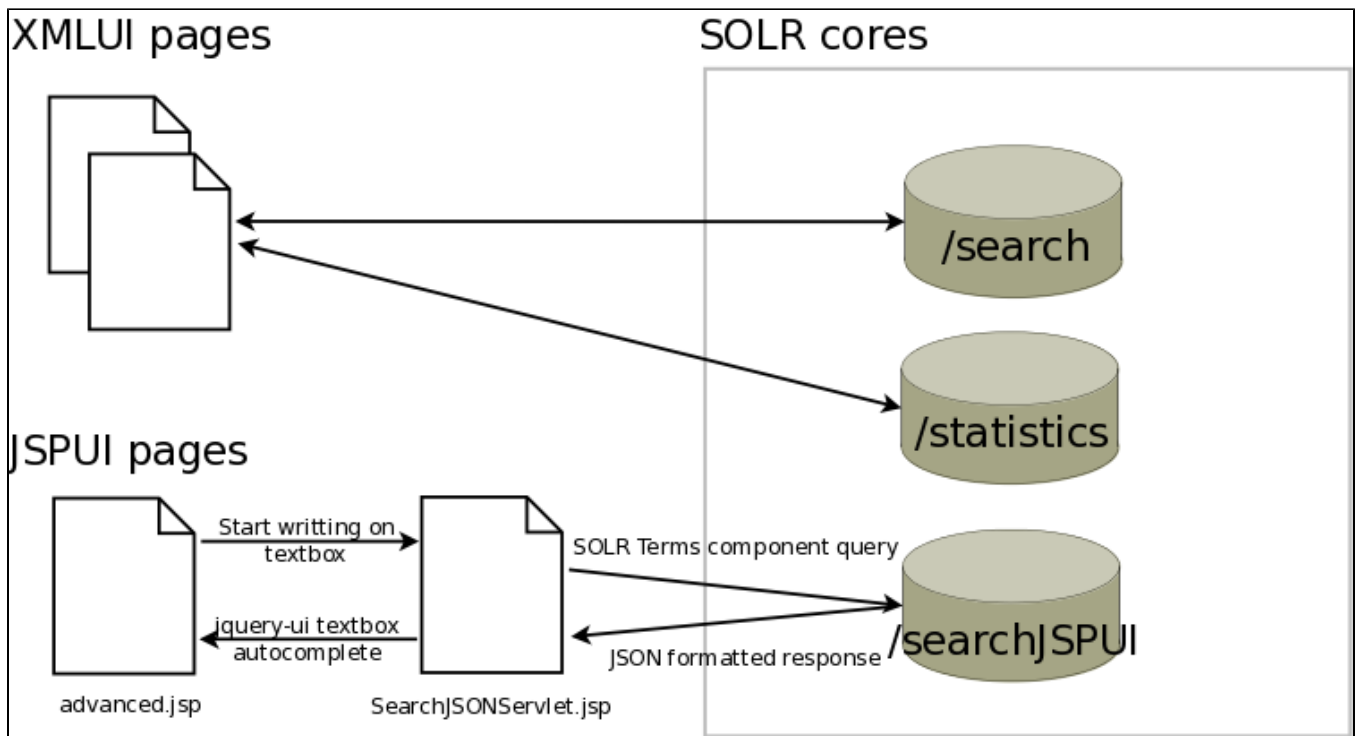
Once finished our textboxes must work like this:

The image shows a screenshot of the JSPUI advanced search interface. At the top, there is a search bar labeled "Buscar:" with a dropdown menu set to "Todo" and a link "Otras colecciones". Below this, on the left, is a section "Tipo de búsqueda:" with a dropdown menu set to "Tema". Underneath, there are two rows of checkboxes (both checked with 'Y') and dropdown menus, both labeled "Palabra clave". On the right, there is a section "Buscar por:" with a text input field containing "asd". A dropdown menu is open below this field, displaying a list of suggestions: "asdf", "asdf2" (highlighted), "asdf5", "asdf4", "asdf3", "asdf6", "asdf3α", "asdfg", "asdf1", and "asdf7". To the right of the input field and the suggestions list are three buttons, each with the Greek letter Omega (Ω). At the bottom right, there is a button labeled "Buscar".

Administrators that use XMLUI interface will notice that Discovery functionality strongly increases searching and browsing by the use of layered navigation and textbox autocomplete among others. To achieve these goals an Apache Solr search server is deployed in [dspace-source]/dspace/solr to take care of all query-reponse handling.

JSPUI interface use an Apache Lucene Directory (located on [dspace]/search ) to search and browse by, but it leaks of previously mentioned functionalities.

We'll use a servlet class to query a SOLR core to generate JSON responses, this responses will be wired to textboxes with autocomplete functionality. Here is a simple diagram of this dataflow.



## First steps: Preparing JSPUI advanced search page

First of all we'll add jquery and jquery-ui libraries to advanced search .jsp page. These libraries have the necessary code to implement autocomplete behavior. We can add them on head section of advanced search page [dspace-source]/dspace-jspui/dspace-jspui-webapp/src/main/webapp/search/advanced.jsp, or use a on/off switch approach to attach them as described on [Adding jQuery \(or other script library\) support on JSPUI](#). In this manual we'll use second approach because it's the fastest way to disable scriptaculous libraries to avoid colliding with jquery (both use the same notation \$ for calling functions ).

We'll edit [dspace-source]/dspace-jspui/dspace-jspui-webapp/src/main/webapp/search/advanced.jsp

## advanced.jsp

```
...
<%
//Add this code on top section
//Generate JSPUI Solr path url
String scheme = request.getScheme();           // http
String serverName = request.getServerName();    //request.getServerName();    // hostname.com
int serverPort = request.getServerPort();       // 80
String parentContextPath = "";                 //Change this value if dspace not installed on html root
folder
String contextPath = "/jspui";
String servletPath = "/searchJSON";
// Reconstruct original requesting URL
StringBuffer url = new StringBuffer();
url.append(scheme).append("://").append(serverName);
if (serverPort != 80)
url.append(":").append(serverPort);
url.append(parentContextPath).append(contextPath).append(servletPath);
String solrPath = url.toString();

%>
...

<dspace:layout locbar="nolink" titlekey="jsp.search.advanced.title" scriptaculous="off" jquery="on">
<script type="text/javascript" src="<%=request.getContextPath()%>/static/js/autocomplete.js"></script>
<script type="text/javascript">
window.onload = function() {
    monkeyPatchAutocomplete();
    replaceSAXExpression('<%=solrPath %>', 10,'tfield1', 'tquery1');
    replaceSAXExpression('<%=solrPath %>', 10,'tfield2', 'tquery2');
    replaceSAXExpression('<%=solrPath %>', 10,'tfield3', 'tquery3');
}
</script>
...
<!-- Add id field and onchange event call for every search combobox, add id field on search textbox too-->
<select name="field1" id="tfield1" onchange="JavaScript:replaceSAXExpression('<%=solrPath%>', 10 , 'tfield1',
'tquery1');">
<input type="text" name="query1" id="tquery1" value="<%=StringEscapeUtils.escapeHtml(query1)%>" size="30" />
...
<select name="field2" id="tfield2" onchange="JavaScript:replaceSAXExpression('<%=solrPath%>', 10 , 'tfield2',
'tquery2');">
<input type="text" name="query2" id="tquery2" value="<%=StringEscapeUtils.escapeHtml(query2)%>" size="30"/>
...
<select name="field3" id="tfield3" onchange="JavaScript:replaceSAXExpression('<%=solrPath%>', 10 , 'tfield3',
'tquery3');">
<input type="text" name="query3" id="tquery3" value="<%=StringEscapeUtils.escapeHtml(query3)%>" size="30"/>
```

## Adding javascript functions

It's time to create a javascript library file that will contain a pair of functions. First one will patch autocomplete default behavior and will bold every part of result text that match our typed words. Second one will replace SAX expression used in autocomplete textbox everytime we change select value (we must search by other index). It will reside in [dspace-source]/dspace-jspui/dspace-jspui-webapp/src/main/webapp/static/js/autocomplete.js

## autocomplete.js

```
/**
 * Functions used on advanced search page
 */
function monkeyPatchAutocomplete() {
    // don't really need this, but in case I did, I could store it and chain
    var oldFn = $.ui.autocomplete.prototype._renderItem;

    $.ui.autocomplete.prototype._renderItem = function( ul, item ){
        var term = this.term.split(' ').join('|');
        var re = new RegExp("(" + term + ")", "gi" );
        var t = item.label.replace(re,"<b>$1</b>");
        return $( "<li></li>" )
            .data( "item.autocomplete", item )
            .append( "<a>" + t + "</a>" )
            .appendTo( ul );
    };
}

function replaceSAXExpression(baseUrl , limit ,fieldName, fieldText){
    var field = $('#'+ fieldName).val();
    $('#'+ fieldText).val('');
    $('#'+ fieldText).autocomplete({
        source: function( request, response ) {
            $.ajax({
                url: baseUrl + "/terms?terms=true&terms.limit=" + limit + "&terms.
sort=count&terms.regex.flag=case_insensitive&terms.fl=" + field + "&terms.regex=.*" + request.term + ".
*&wt=json",
                dataType: "json",
                data: {
                    style: "full",
                    maxRows: 5,
                    name_startsWith: request.term,
                },
                success: function( data ) {
                    response( $.map( data.terms[field], function( item ) {
                        if (!$.isNumeric(item)){
                            return{
                                label: item,
                                value: "\"" + item + "\"",
                            }
                        }
                    })
                );
            });
        },
        minLength: 1,
        select: function( event, ui ) {
            },
        open: function() {
            $( this ).removeClass( "ui-corner-all" ).addClass( "ui-corner-top" );
        },
        close: function() {
            $( this ).removeClass( "ui-corner-top" ).addClass( "ui-corner-all" );
        }
    });
}
```

Now we must create a new SOLR core inside [dspace-source]/solr directory. To do so we'll use one of the existing cores and copy its structure with this command:

```
bash$ cp -R [dspace-source]/dSPACE/solr/search [dspace-source]/dSPACE/solr/searchJSPUI
bash$ mkdir [dspace-source]/dSPACE/solr/searchJSPUI/data
bash$ mkdir [dspace-source]/dSPACE/solr/searchJSPUI/data/index
```

Edit [dspace-source]/dSPACE/solr/solr.xml and add one extra line with our new search core.

```
<cores adminPath="/admin/cores">
  <core name="search" instanceDir="search" />
  <core name="statistics" instanceDir="statistics" />
  <core name="searchJSPUI" instanceDir="searchJSPUI" /> <!-- Add this line-->
</cores>
```

Overwrite new solr core configuration file [dspace-source]/dSPACE/solr/searchJSPUI/conf/schema.xml with this file [schema.xml](#) to handle lucene indexes generated by JSPUI.

If no modification to search indexes has been done on dSPACE.cfg file this configuration file will be enough.

If we have added new search fields we must add them to schema.xml file in order to be accessible by SOLR and allow autocomplete functionality. Example in schema.xml:

```
<field name="publisher" type="text" indexed="true" stored="true" />
<!-- field must have the same name value as we defined in search index in dSPACE.cfg -->
```

Then we'll edit [dspace-source]/dSPACE/config/dSPACE.cfg and modify default JSPUI search index folder:

```
# Where to put search index files
search.dir = ${dSPACE.dir}/solr/searchJSPUI/data/index
```

So now our SOLR core is set up, new uploaded items indexes will be stored inside its data folder. Proceed with next step:

## Creating SearchJSONServlet.java

Now we'll create the servlet responsible of reading autocomplete petitions and response with solr JSON structures. We could omit this intermediate class but solr it's blocked from external queries due to security concerns, so this inner class will act as a bridge from client browser queries and internal solr server. We'll create [dspace-source]/dSPACE-jspui/dSPACE-jspui-api/src/main/java/org/dSPACE/app/webui/servlet/SearchJSONServlet.java class and then will fill its content with this code:

### SearchJSONServlet.java

```
package org.dSPACE.app.webui.servlet;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
import java.sql.SQLException;

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.transform.stream.StreamResult;

import org.apache.log4j.Logger;
import org.dSPACE.authorize.AuthorizeException;
```

```

import org.dspace.core.Context;

public class SearchJSONServlet extends DSpaceServlet
{
    private static final long serialVersionUID = 1L;
    /** log4j category */
    private static Logger log = Logger.getLogger(SearchJSONServlet.class);

    /**
     * Create an DSpaceServlet Servlet
     */
    public SearchJSONServlet()
    {
        super();
    }

    protected void doDSPost(Context context, HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException,
        SQLException, AuthorizeException
    {
        doDSGet(context, request, response);
    }

    protected void doDSGet(Context context, HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException,
        SQLException, AuthorizeException
    {
        String pathString = request.getPathInfo();
        String queryString = request.getQueryString();

        //Build localhost solr query path
        String scheme = request.getScheme();           // http
        String serverName = "127.0.0.1";               // Solr can only be accessed from localhost
        int serverPort = request.getServerPort();      // 80

        String contextPath = "/solr";
        String servletPath = "/searchJSPUI";
        String parentContextPath = "";                //Change this value if your dspace installation is not on base
        web server directory
        StringBuffer url = new StringBuffer();
        url.append(scheme).append("://").append(serverName);
        if (serverPort != 80)
            url.append(":").append(serverPort);
        url.append(parentContextPath).append(contextPath).append(servletPath);
        url.append(pathString + "?");
        url.append(queryString);
        String solrPath = url.toString();

        //Get response from solr server and send it as servlet reponse
        StringBuilder jsonResponse = new StringBuilder();
        URL solrURL = new URL(solrPath);
        URLConnection solrConnection = solrURL.openConnection();
        BufferedReader in = new BufferedReader(new InputStreamReader(solrConnection.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            jsonResponse.append(inputLine + "\n");
        in.close();
        StreamResult result = new StreamResult(response.getOutputStream());
        jsonResponse.write(result);
        response.setHeader("Content-Length", String.valueOf(jsonResponse.length()));
        response.getOutputStream().flush();

        byte[] buf = jsonResponse.toString().getBytes();
        response.setContentLength(buf.length);
        ServletOutputStream servletOut = response.getOutputStream();
        servletOut.write(buf);
    }
}

```

This code captures input query string and builds a conformant SOLR query string that is send to SOLR server, it's reponse it's directly returned via result variable to reponse output stream.

Now we'll link our new servlet with its appropriate url, to do so we we'll add this lines in [dspace-source]/dspace-jspui/dspace-jspui-webapp/src/main/webapp/WEB-INF/web.xml

#### web.xml

```
<servlet>
    <servlet-name>searchJSON</servlet-name>
    <servlet-class>org.dspace.app.webui.servlet.SearchJSONServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>searchJSON</servlet-name>
    <url-pattern>/searchJSON/*</url-pattern>
</servlet-mapping>
```

## Check results

After we package and install our modified version of JPSUI interface in our webserver, we will check a pair aspects to be sure everthing works as expected.

First we'll browse locally to <http://127.0.0.1/solr/> or <http://127.0.0.1:8080/solr/> (depending on your webserver port and configuration) and there must appear three SOLR cores as hyperlinks:

1. search
2. statistics
3. searchJSPUI.

*Note:* If there's no desktop environment installed on your server you can use "links" package to browse it on text mode or even use "curl" program to send http request to our web server.

We'll click on last one and try to access it's admin webpage and then to schema browser, there must be three categories on left column bar: fields, dynamic fields and field types.

## Other possible scenarios

If this modification is done over a running instance with data in it we must follow this steps after run "ant update":

- Verify that running dspace.cfg contains this configuration line:

```
search.dir = ${dspace.dir}/solr/searchJSPUI/data/index
```

- run an index rebuild to populate [dspace]/solr/searchJSPUI/data/index folder with current search indexes (recommended) or copy them directly from [dspace]/search