

# HackingSolr

- [Hacking On Solr](#)
  - [Source Repository](#)
- [Hacking With Solr](#)
  - [Solr Plugins](#)
  - [Customizing the War](#)
  - [web.xml Hacks](#)
    - [Customizing the Path Prefix Solr Intercepts](#)
    - [Hardcoding Solr Home](#)
  - [Embedding Solr](#)

## Hacking On Solr

### Source Repository

The Solr website contains [Information about the Lucene/Solr Source Code SVN Repository](#)


There are currently two active development branches: "branches/branch\_4x/" which all 4.x releases are derived from, and "trunk" which will be [nightly builds](#). There are [Nightly Builds](#) for both.

## Hacking With Solr

### Solr Plugins

The first thing to know if you are interested in hacking and customizing Solr, is that you may not need to customize the source at all. Solr has a fairly extensive number of [Plugin Hooks](#) that allow you to add custom functionality for a variety of purposes by developing against some simple APIs, and then use your plugins by placing your jar in a special "lib" directory and referring to them by name in your configuration file.

### Customizing the War

 [Solr3.4](#) Note that the ant build.xml files changed in Solr 3.4+, so these instructions would need to be altered for earlier versions.

If you want to know how to add a custom plugin and let the plugin replace the web.xml. Create your plugin similar the existing plugins in the contrib directory. Create a build.xml file and add this to the build.xml file. You will need to change the build.xml to fit your needs but this build.xml is showing you post-compile executions. What it is supposed to do is to replace the web.xml file with a custom-web.xml file and to place a compiled jar file from the plugin in to the lib directory so that solr will include it in the solr.war file.

```
<?xml version="1.0"?>
<project name="solr-customplugin" default="default">
  <property name="contrib.has.webapp" value="true" />
  <import file="../contrib-build.xml"/>
  <!-- Add the compiled jar to our war file -->
  <target name="add-to-war">
    <echo message="CUSTOM: Cleaning ${common-solr.dir}/lib/${fullnameever}.jar from the build " />
    <delete failonerror="false" file="${common-solr.dir}/lib/${fullnameever}.jar" />
    <echo message="CUSTOM: Copying ${common-solr.dir}/dist/${fullnameever}.jar to ${common-solr.dir}/lib" />
    <copy file="${common-solr.dir}/dist/${fullnameever}.jar" todir="${common-solr.dir}/lib"/>
    <delete failonerror="false" file="${common-solr.dir}/webapp/web/WEB-INF/web.xml" />
    <copy file="web-custom.xml" tofile="${common-solr.dir}/webapp/web/WEB-INF/web.xml" />
  </target>
  <description>Build file for CUSTOM Search specifics</description>
  <!-- Add this whenever we want to build the example -->
  <target name="example" depends="common-solr.dist"/>
</project>
```

Source : <http://www.nickveenohof.be/blog/adding-custom-plugin-solr>

### web.xml Hacks

Some behavior of Solr can be modified just by changing the [web.xml](#)

### Customizing the Path Prefix Solr Intercepts

If you are wiring Solr into a larger web application which controls the web context root, you will probably want to mount Solr under a path prefix (app.war with /app/solr mounted into it, for example). You will need to specify a prefix init-param for the SolrDispatchFilter to do this. See the comments in the web.xml for details.

## Hardcoding Solr Home

The Solr Home can be hardcoded using JNDI

```
<env-entry>
  <env-entry-name>solr/home</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>/put/your/solr/home/here</env-entry-value>
</env-entry>
```

## Embedding Solr

In addition to using Solr as a web based application, the SolrJ API provides the ability to embedded the core Solr functionality into any Java application using [EmbeddedSolrServer](#).