Spell Checking

Spell checking requires the deployment of several server-side components onto a J2EE compatible application server (e.g. Jetty, or Apache Tomcat).

The following server-side components are required to enable spell checking:

Component	File	Description
Allowed Origins	ephox-allowed-origins.war	Supplies configuration for server components to communicate with your application.
Spell Checking	ephox-spelling.war	Spell checking service for JavaScript editors.

This guide will help you set up the Spelling server-side components, and show you how to use them in conjunction with editor clients. The steps required are:

1. Install a Java application server (or use existing)
2. Deploy server-side components
3. Create a configuration file and configure the allowed origins service
4. Pass the configuration file to the Java application server
5. Restart the Java application server
6. Set up editor client instances to use the spelling service

1. Install a Java Application Server

Server-side components require a Java Application Server to run.

If you don't already have a Java application server installed you can easily install Tomcat or Jetty with their default settings. These are both simple, open source Java application servers and they're easy to install and configure. The editor SDK supports both of these platforms. You can find the instructions and download links below. For the later setup, it's also helpful if you note any domain name and port number you specify during installation of the web application server.

Jetty	download	instructions
Tomcat	download	instructions

Memory Requirement

Please ensure that you configure your Java Server (Tomcat/Jetty etc) with a minimum of 4GB.
Please refer to the Services Troubleshooting page if you require instructions on how to explicitly define how much RAM will be allocated to your Java server.

2. Deploy Server-side Components

You'll need to ensure you deploy all of the WAR files packaged with the SDK:

- ephox-allowed-origins.war
- ephox-spelling.war

The easiest way to deploy these files is to simply drag and drop them into the webapps directory of your Tomcat/Jetty server (or equivalent folder of another Java application server), and then restart the server.

More information on deploying components/applications:
Deploying applications with Tomcat 6.0
Deploying applications with Jetty

3. Create a configuration file and configure the allowed origins service

It is recommended that you use a plain text editor (eg: gedit, vim, emacs, notepad etc) when creating or editing the `application.conf`. Do not use editors like Evernote as there is a good chance of smart quotes being used where plain quotes should be used and this will cause the services to fail.

Services requires a configuration file named `application.conf` to be referenced by the application server.

The SDK comes packaged with an example configuration file (`examples/application.conf`) that can be used as a reference guide. You can use this example file (after modifying it with your settings). We recommend that you make a backup of the file before editing it.

The allowed-origins configuration element will need to be specified in order for the spelling server-side component to work.

allowed-origins

This element configures the allowed-origins service which allows all server-side components to communicate with specified domains.

The `origins` attribute must list all the domains that instances of the editor will be hosted on. Only requests from the listed origins will be processed by the server-side components. Requests from any other domains will be rejected.

The `url` attribute defines the location of the allowed-origins service.

element	`allowed-origins`	Stores CORS setup information
attribute	`origins`	An array of strings representing the domains allowed to communicate with the services. Note*: Be sure to* include the protocol (https or http) and any required port number (eg:8080) in the string.
attribute	`url`	This string is a concatenation of two values: String 1: The URL location of the allowed-origins service String 2: The API to access in the service (`/cors`).

Entering Origins

The origins are matched by protocol, hostname and port. So you may need a combination of all three, depending on which browser /s you use. If you are serving the editor and services from http://localhost & port 80, then the list of origins should have an entry for "http://localhost" and any other servers with ports, like so:

```
ephox{
    allowed-origins{
        origins=["http://localhost", "http://any-other-servers:port"]
        url = "http://localhost/ephox-allowed-origins/cors"
    }
}
```

This only applies to port 80 because this being the default http port, browsers omit it when talking to the server. For every other port and hostname, the recommended setting is to make one entry with the port and one without the port. This is because different browsers behave differently with regards to the Origin header. So the config file should resemble:

```
ephox{
    allowed-origins{
        origins=["http://hostname", "http://hostname:1234"]
        url = "http://hostname:1234/ephox-allowed-origins/cors"
    }
}
```

Ensure that you have the right protocol specified, and for more examples see the section below. If you experience issues, please use the Troubleshooting guide (in the Tip below) and you should be able to see if the browser sends a different origin to the one that you have specified. Both must match for the services to work.

Depending on your configuration and the browser you use, you may need to specify the port number as well when listing the origin. If you observe that requests are failing with services not being available, it may be because the port number is required. Refer to troubleshooting guide - section titled Investigating Using the Browser's Network Tools

Example

allowed-origins Example

```
ephox {
        allowed-origins {
                origins = [ "http://myserver", "http://myserver:8080", "http://myotherserver",
"http://myotherserver:9090", "https://mysecureserver" ]
                url = "http://myserver:8080/ephox-allowed-origins/cors"
        }
}
```

4. Pass the configuration file to the Java application server

You'll need to reference the configuration file created in Step 3 as a parameter passed to the JVM running the services. Once the server has been configured to use the file, restart the server.

Note

If the path to your *`application.conf`* file has spaces in it, you must ensure you prefix each white space with an escape character (\).

Example: " -Dephox.config.file=/config/file/location/with/white\ space/`application.conf`"

The following examples demonstrate how to reference `application.conf` for Tomcat or Jetty instances.

- Tomcat (Unix)
- Tomcat (Windows)
- Jetty (simple configuration)
- Jetty (automatic configuration for services launching on system start-up)

Tomcat (Unix)

Make/edit a script at `/tomcat/install/directory/bin/setenv.sh`

Ensure the file contains a single line, like (this must be the absolute path as before):

Example setenv.sh

```
CATALINA_OPTS=" -Dephox.config.file=/config/file/location/application.conf"
```

Tomcat (Windows):

Setting configuration varies based on installation of tomcat. You can choose one of following options to complete configuration setting.

Option A. Installation though Windows Command line (CMD) with tomcat binary file:

Make/edit a script at `DRIVE:\tomcat\install\directory\bin\setenv.bat (This file might not exist for some versions of tomcat, create it in` *`bin`* `folder)`

The file should contain a single line:

Example setenv.bat

```
set CATALINA_OPTS= -Dephox.config.file=DRIVE:\config\file\location\application.conf
```

After setting up configuration, you can launch tomcat by startup.bat or catalina.bat

Option B. Windows Service Installer:

1. From start menu, open Monitor Tomcat and see popping up window.
2. Go to Java tab and see Java Options.
3. In Java Options, insert new line, which is

```
-Dephox.config.file=DRIVE:\config\file\location\application.conf
```

4. Then you can go to General tab to start Tomcat.

Jetty (simple configuration):

You can specify your `application.conf` as a parameter to this command, along with other jvm parameters:

Jetty

```
java -jar /jetty/install/directory/start.jar -Dephox.config.file="/config/file/location/application.conf"
```

Jetty (automatic configuration for services launching on system start-up)

Edit /etc/default/jetty and add the line:

```
JETTY_OPTS=" -Dephox.config.file=/config/file/location/application.conf"
```

Edit /opt/jetty/start.ini and add the line:

```
" -Dephox.config.file=/config/file/location/application.conf"
```

The first new lines of the file should read:

```
#===========================================================
# Jetty start.jar arguments
# Each line of this file is prepended to the command line
# arguments # of a call to:
# java -jar start.jar [arg...]
#===========================================================
" -Dephox.config.file=/config/file/location/application.conf"
```

5. Restart the Java application server

Once you have created a configuration file, configured the allowed origins service, and passed the configuration file to the Java application server you must restart the Java application server.

6. Set up editor client instances to use the spelling service

With the above steps completed you can now direct the editor client instances to use the server-side spelling component.

Set the `spelling.url` configuration property to the URL of the deployed server side spelling component. This URL is provided to you by your Java application server.

Example

Textbox.io Client Config

```
var config = {
        spelling : {
                url: 'http://yourspelling.server.com/ephox-spelling/' // Spelling service URL
        }
};

var editor = textboxio.replace('#id', config);
```