

## Adding a servlet filter

This is a very simple example of a servlet filter. In this case, we're serving CF files from another location on disk, which is specified by the `cfsrc` parameter. This is basically a hack for the servlet container to be able to find out where files are that Railo already knows about through a mapping, as well as being able to specify default file (like `index.cfm` or `default.cfm`) when the request is a directory. It's pretty specific to what I needed, but is easily customizable.

Eventually I'll add some more details, but hopefully this is enough to get an idea of how to write a servlet filter.

### CFMappingFilter.java

```
package servlet.filter;

import java.io.IOException;
import java.io.File;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletRequest;

/* ----- */
public class CFMappingFilter implements Filter {
    private String cfsrc, defaultFileNames;

    public void init(FilterConfig filterConfig) {
        cfsrc = filterConfig.getInitParameter("cfsrc");
        defaultFileNames = filterConfig.getInitParameter("defaultFileNames");
    }

    /* ----- */
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException,
        ServletException {
        String path = ((HttpServletRequest) request).getServletPath();
        String contextPath = ((HttpServletRequest) request).getContextPath();
        if (cfsrc != null) {
            File requestedFile = new File(cfsrc + path);
            System.out.println("initwith:"+requestedFile.getPath());
            if (requestedFile.exists()) {
                System.out.println("exists");
                if (requestedFile.isDirectory() && !path.endsWith("/")) {
                    System.out.println("redir");
                    HttpServletResponse resp = (HttpServletResponse) response;
                    HttpServletRequest req = (HttpServletRequest) request;
                    resp.sendRedirect(req.getContextPath() + path + "/");
                }
            }
            else if (requestedFile.isDirectory()) {
                File defaultFile = getDefaultFile(requestedFile.getPath());
                System.out.println("getDefault:"+defaultFile.getPath());
                if (defaultFile.exists()) {
                    request.getRequestDispatcher(defaultFile.getPath().replace(cfsrc,"")).forward(request, response);
                }
                else {
                    chain.doFilter(request, response);
                }
            }
        }
        else {
            //System.out.println("go for:" + contextPath + requestedFile.getPath().replace(cfsrc,""));
        }
    }
}
```

```

        request.getRequestDispatcher(requestedFile.getPath().replace(cfsrc, "")).forward(request, response);
    }
} else {
    chain.doFilter(request, response);
}
}
}

private File getDefaultFile(String directory) {
String[] files = defaultFileNames.split(" ");
for (int i = 0; i < files.length; i++) {
    File defaultFile = new File(directory + File.separator + files[i]);
    System.out.println("looking..." + defaultFile.getPath());
    if (defaultFile.exists()) {
        return defaultFile;
    }
}
return new File("");
}

public void destroy() {
}
}
}

```

## Add this to WEB-INF/web.xml

```

CFMappingFilter
servlet.filter.CFMappingFilter

```

```

cfsrc
${src.dir}

```

```

defaultFileNames
index.cfm default.cfm

```

```

CFMappingFilter
/*

```

Compile the java file, and add it to your WEB-INF/classes folder, then start the servlet container.

And now your servlet container can find files that are in the /src directory!

I personally do the compiling and adding to WEB-INF through an Ant script, and for completeness (hah!), here's a snippet:

```

FileServlet
File Servlet for simple files
railo.loader.servlet.FileServlet
2

```

```

FileServlet
/

```

```
CFMappingFilter
servlet.filter.CFMappingFilter
```

```
cfsrc
${src.dir}
```

```
defaultFileNames
index.cfm default.cfm
```

```
CFMappingFilter
/*
```

```
]]>
```

You'd need to change the classpath of the javac task to point at your servlet containers lib folder (this example uses jetty-runner.jar, which has all the other jars inside it).

As you can see, I'm also using Railo's file filter to let it serve out images and whatnot. Railo's file filter alone was /almost/ all I needed, with this additional one, I now have things set up in a way that works for the process it was needed for.

Happy times!