|  |
| --- |
| Armatiek BV |
| XSLWeb Quick Start |
| Web application framework for XSLT developers |

|  |
| --- |
| Maarten Kroon, Arjan Loeffen  31-12-2014 |

Table of Contents

[1 Introduction to XSLWeb 2](#_Toc407778762)

[1.1 Introduction 2](#_Toc407778763)

[1.2 Request XML 2](#_Toc407778764)

[1.3 Response XML 2](#_Toc407778765)

[1.4 Request dispatcher XSLT stylesheet 2](#_Toc407778766)

[1.5 Caching 3](#_Toc407778767)

[1.6 XSLWeb web applications 4](#_Toc407778768)

[1.6.1 webapp.xml 4](#_Toc407778769)

[1.6.2 Jobs 4](#_Toc407778770)

[1.7 Development and production mode 4](#_Toc407778771)

[1.8 Logging 5](#_Toc407778772)

[1.9 Caching 5](#_Toc407778773)

[1.10 XSLT extension functions 5](#_Toc407778774)

[1.10.1 Built in extension functions 5](#_Toc407778775)

[1.10.2 Session functions 5](#_Toc407778776)

[1.10.3 Webapp functions 5](#_Toc407778777)

[1.10.4 Context functions 6](#_Toc407778778)

[1.10.5 EXPath File 6](#_Toc407778779)

[1.10.6 EXPath HTTP Client 8](#_Toc407778780)

[1.10.7 Base64 8](#_Toc407778781)

[1.10.8 Log 8](#_Toc407778782)

[1.10.9 Email 9](#_Toc407778783)

[1.10.10 Serialization 9](#_Toc407778784)

[1.10.11 UUID 9](#_Toc407778785)

[1.10.12 Custom extension functions 9](#_Toc407778786)

[2 Download and installation 10](#_Toc407778787)

[2.1 Single executable jar with embedded application server and packaged JRE. 10](#_Toc407778788)

[2.2 Single executable jar with embedded application server without JRE. 10](#_Toc407778789)

[2.3 Web application archive (.war) 10](#_Toc407778790)

# The fundamentals of XSLWeb

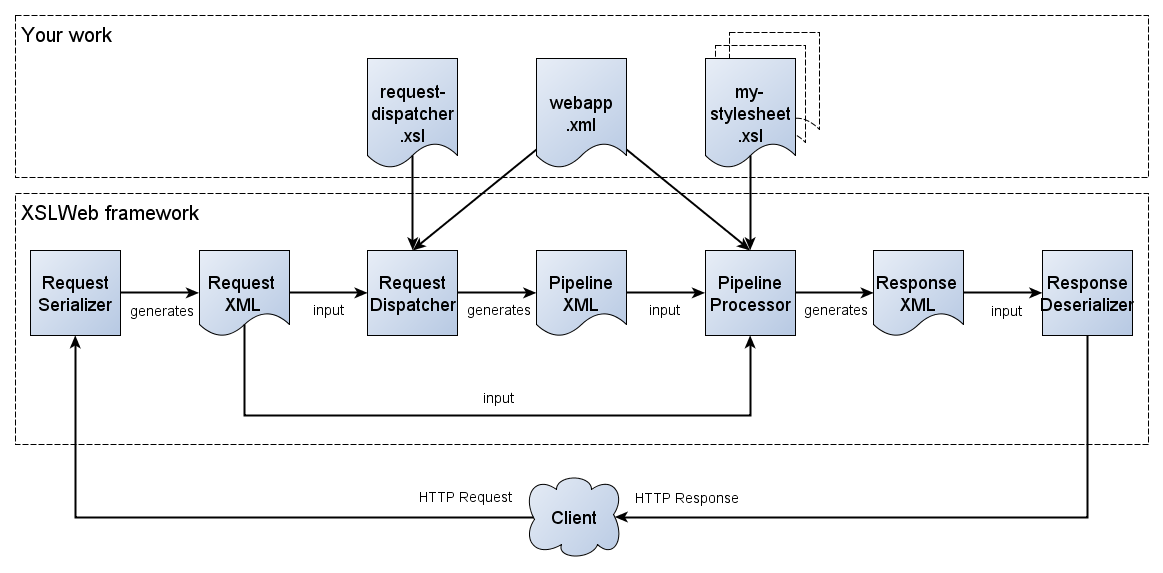
## Introduction

XSLWeb is an open source and free to use web development framework for XSLT developers. With XSLWeb, XSLT developers can develop both web applications (websites) and web services. In essence, an XSLWeb web application is a set of XSLT stylesheets that transform an XML representation of the HTTP request (the *Request XML*) to an XML representation of the HTTP response (the *Response XML*).

Which specific XSLT stylesheet (or pipeline of XSLT stylesheets) must be executed for a particular HTTP request is governed by another XSLT stylesheet, the *request dispatcher stylesheet*.

Further configuration of an XSLWeb web application can be specified in an XML configuration document called *webapp.xml*. An XSLWeb server can contain multiple web applications.

Diagram X shows the flow of a HTTP request within XSLWeb:



* A HTTP request is sent from a client (for instance a web browser).
* The HTTP request is serialized by the Request Serializer of XSLWeb to a XML representation (the Request XML). All information of the request is preserved.
* The Request XML is the input of a XSLT transformation using the webapp specific stylesheet *request-dispatcher.xsl* that you write. The output of this transformation is a pipeline specification, in the simplest form specifying the path to a XSLT stylesheet that will be used to transforming the Request XML to the Response XML.
* The pipeline specification is

## Request XML

The Request XML is a XML representation (or *XML serialization*) of the HTTP Request. The Request XML conforms to the XML Schema *«xslweb-home»/config/xsd/xslweb/request.xsd*, and contains the following information:

* The request characteristics: auth-type, character-encoding, content-length, context-path, content-type, local-addr, local-name, local-port, method, path, path-info, path-translated, protocol, query-string, remote-addr, remote-host, remote-port, remote-user, requested-session-id, request-URI, request-url, scheme, server-name, server-port, servlet-path, webapp-path, is-secure, is-requested-session-id-from-cookie, is-requested-session-id-from-url and is-requested-session-id-valid.
* HTTP headers
* Request parameters
* Request body
* Request attributes
* File uploads
* Session information
* Cookies

See XXX for an example of a Request XML document.

## Response XML

The Response XML is a XML representation (XML serialization) of the HTTP Response. The Response XML must conform to the XML Schema *«xslweb-home»/config/xsd/xslweb/response.xsd*, and contains the following information:

* HTTP headers
* Response body
* Session information
* Cookies

See XXX for an example of a Response XML document

## Request dispatcher XSLT stylesheet

The task of the XSLT stylesheet *request-dispatcher.xsl* is to determine which XSLT stylesheet (or pipeline of XSLT stylesheets) must be used to transform the Request XML to the Response XML. The input of the request dispatcher transformation is therefore the Request XML so it has all information available to determine which XSLT stylesheet should process the request further. The output of the request dispatcher transformation is a pipeline definition that must conform to the XML Schema *«xslweb-home»/config/xsd/xslweb/pipeline.xsd*.

Afbeelding

Examples:

Below is an example of a very basic request dispatcher stylesheet that generates a valid pipeline for the HTTP request *http://my-domain/my-webapp/hello-world.html*:

<xsl:stylesheet  
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
 xmlns:req="http://www.armatiek.com/xslweb/request"  
 xmlns:pipeline="http://www.armatiek.com/xslweb/pipeline"  
 version="2.0">  
   
 <xsl:template match="/req:request[req:path = '/hello-world.html']">  
 <pipeline:pipeline>  
 <pipeline:transformer   
 name="hello-world"   
 xsl-path="hello-world.xsl"   
 log="true"/>  
 </pipeline:pipeline>  
 </xsl:template>  
   
</xsl:stylesheet>

The following example uses the request parameter *lang* in the request *http://my-domain/my-webapp/hello-world.html?lang=en* to determine the stylesheet:

<xsl:stylesheet  
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
 xmlns:req="http://www.armatiek.com/xslweb/request"  
 xmlns:pipeline="http://www.armatiek.com/xslweb/pipeline"  
 version="2.0">  
   
 <xsl:template match="/req:request[req:path = '/hello-world.html']">  
 <xsl:variable  
 name="lang"  
 select="req:parameters/req:parameter[@name='lang']/value[1]"/>  
 <pipeline:pipeline>  
 <pipeline:transformer name="hello-world"  
 xsl-path="{concat('hello-world-', $lang, '.xsl')}"/>  
 </pipeline:pipeline>  
 </xsl:template>  
   
</xsl:stylesheet>

Caching

The output of a pipeline can be cached by providing an optional attribute *cache-key* on the element *pipeline:pipeline*. The purpose of caching the output is to gain performance; a response that can be served from cache will be returned quicker (no transformations are necessary), and will decrease the load on the server.   
The output of the pipeline will by default be cached for one minute. To cache the output longer or shorter than that, an extra optional attribute *cache-timeout* can be provided specifying the timeout in seconds. A third optional attribute *cache-scope* can be used to specify whether the output should be cached and reused by all users of the web application (“webapp”), or for a specific user (“user”). The output is cached persistently on disk, so it will still be available after restarting XSLWeb.

Serialization

The way the result of the transformation pipeline is serialized to XML, XHTML, HTML or text is by default determined by the serialization attributes of the element *xsl:output* in the last XSLT stylesheet of the pipeline (the attributes “method”, “encoding”, “indent”, “omit-xml-declaration” and so on). In case the output of the pipeline should not be XML, or XHTML, HTML or text, a serializer element can be added to the pipeline. XSLWeb provides two serializers, one for JSON and one for PDF.

Logging

## Caching

## XSLWeb web applications

One XSLWeb installation can contain multiple web applications. A web application can be added under the folder *«xslweb-home»/webapps* and has the following minimal folder structure:

my-webapp/  
 lib/  
static/  
xsl/  
 request-dispatcher.xsl  
 my-stylesheet.xsl  
 webapp.xml

The folder *my-webapp* can have any name you like. This name will also be the name in the url (TODO). The folder *lib* can contain any custom XSLT extension functions you have developed (in Java) and 3rd party libraries they depend on (see TODO). The folder *static* can contain all the static files you use in your web application, like images, css stylesheets and javascripts. The folder *xsl* contains the XSLT stylesheet *request-dispatcher.xsl* and at least one pipeline XSLT stylesheet that transforms Request XML to Response XML. The file *webapp.xml* contains further configuration of your web application.

### webapp.xml

The file *webapp.xml* contains the configuration of your web application. It must conform to the XML Schema *«xslweb-home»/config/xsd/xslweb/xslweb-webapp.xsd*, and contains the following configuration items:

* *Title*: The title of your web application
* *Description*: The description of your web application
* *Development-mode*: TODO
* *Resources*: The definition of requests to static files that should not be processed by request dispatcher but should be served straight away.
* *Parameters*: The definition of webapp specific configuration parameters that are passed as stylesheet parameters to every XSLT transformation.
* *Jobs*: The definition of scheduled jobs (see TODO).

### Jobs

A job in webapp.xml defines a pipeline that is (repeatedly) executed at a certain time using a [cron](http://en.wikipedia.org/wiki/Cron) expression.

Example:

<job>  
 <name>MyJob</name>   
 <uri>job/my-job</uri>   
 <!— Execute at 10:15am on the 15th day of every month: -->  
 <cron>0 15 10 15 \* ?</cron>   
</job>

## Development and production mode

In webapp.xml a web application can be configured to run in *development mode* or *production mode*. The differences between development and production mode are:

* In development mode, compiled XSLT stylesheets are not cached. That means that for every request all stylesheets in the pipeline are reread from disk and recompiled and therefore changes will be visible immediately. In production mode, stylesheets are compiled and cached the first time they are used. However, in production mode, changes in stylesheets will automatically be detected by the file alteration monitor and the complete web application will be reloaded. So there is no need to restart the application server when deploying stylesheets in production mode. The file alteration monitor will also detect and pick up changes in the webapp.xml configuration file and plugin extension function library jars.
* In development mode, the output of a pipeline is not streamed directly to the client (e.g. the browser) but instead buffered until the complete transformation is finished. If an error occurs during the execution of the pipeline, the error message and stack trace are sent to the client, making it easier to debug the error. If an error occurs in production mode, only a HTTP status code 500 (internal server error) is sent to the client (that is, if the response is not already committed by the application server).
* In development mode a pipeline step can be configured to log its (intermediate) output to the log file *«xslweb-home»/logs/pipeline.log*, by specifying *log=”true”* on the pipeline step. In production mode all logging of the output of pipeline steps is disabled.

## Logging

## Caching

## XSLT extension functions

### Built in extension functions

XSLWeb contains a set of readily available extension functions. To use these extension functions in your XSLT stylesheets you only have to declare the namespace.

### Session functions

Namespace: *http://www.armatiek.com/xslweb/session*

Functions:

get-attribute($name as xs:string) as item()\*

set-attribute($name as xs:string, attr as item()\*) as xs:boolean

### Webapp functions

Namespace: *http://www.armatiek.com/xslweb/functions/webapp*

Functions:

get-attribute($name as xs:string) as item()\*

set-attribute($name as xs:string, attr as item()\*) as xs:boolean

get-cache-value($cache-name as xs:string, $key-name as xs:string) as item()\*

set-cache-value($cache-name as xs:string, $key-name as xs:string, attrs as item()\*, duration as xs:integer) as xs:boolean

### Context functions

Namespace: *http://www.armatiek.com/xslweb/functions/context*

Functions:

get-attribute($name as xs:string) as item()\*

set-attribute($name as xs:string, attr as item()\*) as xs:boolean

### EXPath File

EXPath File is a standard file system API for XPath. It defines extension functions to perform file system related operations such as listing, reading, writing, copying and moving files or directories. It has been designed to be compatible with XQuery 1.0 and XSLT 2.0, as well as any other XPath 2.0 usage. The API is described [here](http://expath.org/spec/file).

Namespace: *http://expath.org/ns/file*

Functions:

exists($path as xs:string) as xs:boolean

is-dir($path as xs:string) as xs:boolean

is-file($path as xs:string) as xs:boolean

last-modified($path as xs:string) as xs:dateTime

size($file as xs:string) as xs:integer

append($file as xs:string, $items as item()\*) as xs:boolean

append($file as xs:string,  
 $items as item()\*,  
 $params as element(output:serialization-parameters)) as xs:boolean

append-binary($file as xs:string,  
 $value as xs:base64Binary) as xs:boolean

append-text($file as xs:string,  
 $value as xs:string) as xs:boolean

append-text($file as xs:string,   
 $value as xs:string,  
 $encoding as xs:string) as xs:boolean

append-text-lines($file as xs:string,  
 $values as xs:string\*) as xs:boolean

append-text-lines($file as xs:string,  
 $lines as xs:string\*,  
 $encoding as xs:string) as xs:boolean

copy($source as xs:string, $target as xs:string) as xs:boolean

create-dir($dir as xs:string) as xs:boolean

create-temp-dir($prefix as xs:string, $suffix as xs:string) as xs:string

create-temp-dir($prefix as xs:string,   
 $suffix as xs:string,  
 $dir as xs:string) as xs:string

create-temp-file($prefix as xs:string, $suffix as xs:string) as xs:string

create-temp-file($prefix as xs:string,   
 $suffix as xs:string,  
 $dir as xs:string) as xs:string

delete($path as xs:string) as xs:boolean

delete($path as xs:string, $recursive as xs:boolean) as xs:Boolean

list($dir as xs:string) as xs:string\*

list($dir as xs:string, $recursive as xs:boolean) as xs:string\*

list($dir as xs:string,  
 $recursive as xs:boolean,  
 $pattern as xs:string) as xs:string\*

move($source as xs:string, $target as xs:string) as xs:Boolean

read-binary($file as xs:string) as xs:base64Binary

read-binary($file as xs:string, $offset as xs:integer) as xs:base64Binary

read-binary($file as xs:string,  
 $offset as xs:integer,  
 $length as xs:integer) as xs:base64Binary

read-text($file as xs:string) as xs:string

read-text($file as xs:string, $encoding as xs:string) as xs:string

read-text-lines($file as xs:string) as xs:string\*

read-text-lines($file as xs:string, $encoding as xs:string) as xs:string\*

write($file as xs:string, $items as item()\*) as xs:boolean

write($file as xs:string,   
 $items as item()\*,   
 $params as element(output:serialization-parameters)) as xs:boolean

write-binary($file as xs:string,   
 $value as xs:base64Binary) as xs:boolean

write-binary($file as xs:string,  
 $value as xs:base64Binary,  
 $offset as xs:integer) as xs:boolean

write-text($file as xs:string, $value as xs:string) as xs:boolean

write-text($file as xs:string,  
 $value as xs:string,  
 $encoding as xs:string) as xs:boolean

write-text-lines($file as xs:string,   
 $values as xs:string\*) as xs:boolean

write-text-lines($file as xs:string,  
 $values as xs:string\*,  
 $encoding as xs:string) as xs:boolean

name($path as xs:string) as xs:string

parent($path as xs:string) as xs:string?

path-to-native($path as xs:string) as xs:string

path-to-uri($path as xs:string) as xs:anyURI

resolve-path($path as xs:string) as xs:string

dir-separator() as xs:string

line-separator() as xs:string

path-separator() as xs:string

temp-dir() as xs:string

### EXPath HTTP Client

EXPath HTTP Client is a standard HTTP client interface for XPath 2.0. It defines one extension function to perform HTTP requests and handle responses, and has been designed to be compatible with XQuery 1.0 and XSLT 2.0, as well as any other XPath 2.0 usage. The API is described [here](http://expath.org/spec/http-client).

EXPath HTTP Client provides a lot more functionality that XSLT’s document() function:

* Execution of other HTTP methods (POST, HEAD, PUT, DELETE etc), making it possible to use both SOAP and REST based web services.
* Requesting text or even binary documents.
* Authentication (Basic and Digest).
* Specify HTTP headers in the request and read the HTTP headers of the response.
* Execute requests to HTML pages and parse them as well-formed XML.

Namespace: *http://expath.org/ns/http-client*

Functions:

send-request($request as element(http:request)) as item()+

send-request($request as element(http:request)?,  
 $href as xs:string?) as item()+

send-request($request as element(http:request)?,  
 $href as xs:string?,  
 $bodies as item()\*) as item()+

### Base64

Namespace: *http://www.armatiek.com/xslweb/functions/context*

Functions:

encode($str as xs:string) as xs:string

decode($str as xs:string) as xs:string

### Log

Namespace: *http://www.armatiek.com/xslweb/functions/log*

Functions:

log($level as xs:string, $message as item()\*) as xs:boolean

log($level as xs:string,   
 $message as item()\*,  
 $params as element(output:serialization-parameters)) as xs:boolean

Where $level is one of “ERROR”, “WARN”, “INFO” or “DEBUG”.

### Email

Namespace: *http://www.armatiek.com/xslweb/functions/email*

Functions:

send-email($email as element(email:email)) as xs:boolean

TODO add example or link to

### Serialization

Namespace: *http://www.armatiek.com/xslweb/functions/serialize*

Functions:

serialize($nodes as node()\*, $options as node()) as xs:string

### UUID

Namespace: *http://www.armatiek.com/xslweb/functions/uuid*

Functions:

uuid() as xs:string

### Custom extension functions

It is also possible to write your own custom XPath extension functions in Java and add them to an XSLWeb web application. These extension functions must be integrated extension functions that use the full interface of Saxon version 9.6 (see <http://www.saxonica.com/documentation9.5/extensibility/integratedfunctions/ext-full-J.html>)

The compiled jar of a custom extension function together with any libraries that the function depend on can be placed in the folder *«web-app»/lib*. There is no need to restart the application server, XSLWeb will detect the jars and will load and register the extension function automatically.

# Download and installation

XSLWeb 1.0 can be downloaded in three distributions:

1. Single executable java library (.jar) with embedded application server (Tomcat 7) and packaged Java JRE (Java 7).
2. Single executable java library (.jar) with embedded application server (Tomcat 7) but without Java JRE.
3. Web application archive (.war)

## Single executable jar with embedded application server and packaged JRE.

This distribution is the easiest to install but also the biggest to download. If you have little or no experience with Java and quickly want to try out XSLWeb this is the distribution you want.

Installation:

1. Download <http://www.armatiek.nl/xslweb/xslweb-1.0-exec-jre.zip>
2. Unzip the zip in a directory of your choice
3. Run the batch file run-xslweb.bat (Windows) or the shell script run-xslweb.sh (Linux/Mac).

## Single executable jar with embedded application server without JRE.

This distribution is also easy to install and relatively small to download. A precondition is that you already have a Java 7 JRE or JDK installed, and that java can be run from .

Installation:

1. Download <http://www.armatiek.nl/xslweb/xslweb-1.0-exec-no-jre.zip>
2. Unzip the zip in a directory of your choice

Run the batch file *run-xslweb.bat* (Windows) or the shell script *run-xslweb.sh* (Linux/Mac).

## Web application archive (.war)

This distribution is a standard J2EE web application that can be installed on any Java application server that supports Servlet Spec 3.0 (Tomcat 7+, Jetty 8+, Glassfish 3+, JBoss AS 6.x/7.x etc.). There is one extra configuration setting that must be done and that is that a Java system property *xslweb.home* must be specified thet points to the XSLWeb home directory, the directory containing the directory *webapps*.

JSON

XSL-FO

JDBC

Web services