XSLWeb 3.0

Web application development framework for XSLT and XQuery developers

Quick Start Guide

Maarten Kroon
Arjan Loeffen

Februari 2018

Table of Contents

1 The fundamentals of XSLWeb

1.1 Introduction

XSLWeb is an open source and free to use web development framework for XSLT and XQuery developers. It is based on concepts similar to frameworks like Cocoon and Servlex, but aims to be more easily accessible and pragmatic.

Using XSLWeb, XSLT/XQuery developers can develop both web applications (dynamic websites) and web services. In essence, an XSLWeb web application is one or more XSLT stylesheets (version 1.0, 2.0 or 3.0) or XQueries (version 1.0, 3.0 or 3.1) 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 XQuery (or pipeline of XSLT stylesheets and XQueries) must be executed for a particular HTTP request is governed by another XSLT stylesheet, the *request dispatcher stylesheet*.

After every XSLT transformation step, an optional validation pipeline step (XML Schema or Schematron) can be added to validate the result of the previous transformation step.

During transformations, data sources can be accessed using a library of built-in extension functions that provide HTTP communication (for example to consume REST or SOAP based web services), file and directory access, relational database access and so on.

The result of a transformation pipeline can be serialized to XML, (X)HTML or plain text format and using specific serializer pipeline steps to JSON, ZIP files, PDF, Postscript or RTF (using XSL:FO and Apache FOP).

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

Diagram 1 shows the flow of a HTTP request to a HTTP response within XSLWeb:
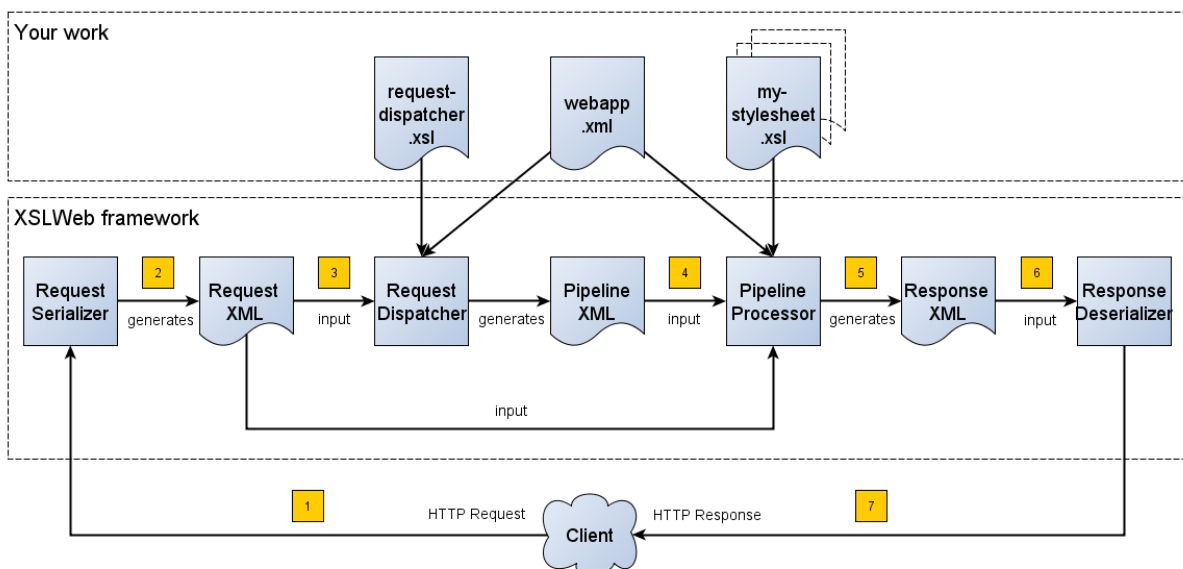

Diagram 1

1. A HTTP request is sent from a client (a web browser or webservice client).
2. The HTTP request is serialized by the Request Serializer to a Request XML document. All information of the request is preserved in the XML.

3. The Request XML is the input of the Request Dispatcher, which transform the Request XML using the webapp specific XSLT stylesheet *request-dispatcher.xsl*. The output of this transformation is a pipeline specification, in the simplest form only specifying the path to a XSLT stylesheet that will be used to transforming the Request XML to the Response XML. This specification could also contain a pipeline of multiple XSLT transformations and XML Schema or Schematron validations.
4. The pipeline specification is the input for the Pipeline Processor, which reads the Pipeline XML and executes the pipeline transformation and validation steps. The input for the first transformation in the pipeline is the same Request XML as was used as input for the Request Dispatcher.
5. The Pipeline Processor executes your pipeline of XSLT stylesheets, XQueries and validations. The last transformation in the pipeline must generate a Response XML document which conforms to the schema *«xslweb-home»/config/xsd/xslweb/response.xsd*.
6. The Response XML is then passed on to the Response Deserializer, which interprets your Response XML and converts it to a HTTP response, which is sent back to the client, a web browser of webservice client (7).

1.2   The Request XML

The Request XML is an 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 properties: 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
- File uploads
- Session information
- Cookies

See *Appendix A: Request XML example* for an example of a Request XML document.

1.3   The Response XML

The Response XML is a XML representation (or *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 *Appendix B: Response XML example* for an example of a Response XML document

1.4   The Request dispatcher XSLT stylesheet

The task of the XSLT stylesheet *request-dispatcher.xsl* is to dynamically generate the pipeline specification that is then used to process the Request XML and convert it to the Response XML. The input of the request dispatcher transformation is the Request XML so it has all information available to generate the correct pipeline. The output of the request dispatcher transformation is a pipeline specification that must conform to the XML Schema *«xslweb-home»/config/xsd/xslweb/pipeline.xsd*.

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="3.0">

  <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. This *lang* parameter is also passed to the stylesheet as a stylesheet parameter:

```
<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="3.0">

  <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-', $lang, '.xsl')}">
        <pipeline:parameter
          name="lang"
          uri=http://www.armatiek.com/xslweb/my-namespace
          type="xs:string">
          <pipeline:value>
            <xsl:value-of select="$lang"/>
          </pipeline:value>
        </pipeline:parameter>
      </pipeline:transformer>
    </pipeline:pipeline>
  </xsl:template>
```

```
</xsl:stylesheet>
```

A pipeline consists of:

- One or more of the following *transformation* pipeline steps:
    - *transformer*: transforms the input of the pipeline step using an XSLT version 1.0, 2.0 or 3.0 stylesheet.
    - *query*: processes the input of the pipeline step using an XQuery version 1.0, 3.0 or 3.1 query.
    - *transformer-stx*: transform the input of the pipeline step using a STX (Streaming Transformations for XML) version 1.0 stylesheet.
- Zero or more of the following *validation* pipeline steps:
    - *schema-validator*: validates the input of the step using an XML Schema version 1.0.
    - *schematron-validator*: validates the input of the step using an ISO Schematron schema.
- Zero or one of the following *serialization* pipeline step
    - *json-serializer*: serializes XML output to a JSON representation.
    - *zip-serializer*: serializes a XML ZIP specification to an actual ZIP file.
    - *resource-serializer*: serializes a text or binary file to the response.
    - *fop-serializer*: serializes XSL-FO generated in a previous pipeline step to PDF using the Apache FOP XSL-FO processor.

See chapter *2 Pipeline steps* for a more in depth overview of these different pipeline steps.

The output of the pipeline can be cached by specifying extra attributes on the *<pipeline:pipeline/>* element, see chapter *7 Response caching*.

In development-mode, the output of the (intermediate) transformation steps can be logged to a log file, see chapters *5 Development mode and production mode* and *6 Logging*.

1.5   The pipeline transformation stylesheets

The result of the request dispatcher stylesheet is a pipeline specification containing one or more transformation, query, validation or serialization steps. The input of the first stylesheet or query in the pipeline is the Request XML, the output of the last stylesheet in the pipeline must conform to the Response XML schema.

XSLWeb extends the standard XSLT/XPath 1.0, 2.0 and 3.0 functionality in a number of ways:

- XSLWeb provides a number of built-in XPath extension functions that you can use to read and write files and directories, execute HTTP requests, access the Request, Response and Context, Session and WebApp objects, log messages, send e-mails and so on, see chapter *11: XPath extension function library*.
- Other pipelines can be called from within a stylesheet and the result of this nested pipeline can be used or embedded in the calling stylesheet by passing a URI that starts with the scheme "xslweb://" to the standard XSLT *document()* function, see chapter *4 Nested pipelines*.
- URLs that are passed to XSLT's *document()* or XQuery's *doc()* function and must be proxied through a proxy server can be provided with two extra request parameters: *proxyHost* and *proxyPort*.

- Within every transformation a number of standard stylesheet parameters is available, see chapter *3: Stylesheet parameters*.

1.6   Web applications

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

```
my-webapp/
  lib/
  static/
  xsl/
    request-dispatcher.xsl
    my-stylesheet.xsl
  xquery/
    my-query.xq
  stx/
    my-stylesheet.stx
  xsd/
    my-xml-schema.xsd
  sch/
    my-schematron.sch
  webapp.xml
```

This web application can be accessed by using the following uri:

http://<domain>:<port>/<xslweb-context-path>/my-webapp

If you use the jar distribution of XSLWeb with default parameters, this uri will be:

http://localhost:8080/my-webapp

Out of the box, XSLWeb contains four web applications, "documentation", "examples", "my-webapp" and "ROOT". The ROOT web application is used for Uri's that doesn't contain a reference to a web application, for instance:

http://localhost:8080

The folder *my-webapp* can have any name you like (provided it doesn't contain spaces or other strange characters). The folder *lib* can contain any custom XPath extension functions you have developed in Java and 3rd party libraries they depend on, see section *11.1.22 Custom extension functions*. The folder *static* contains all static files you use in your web application, like images, css stylesheets and javascript files. 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 folders *xsd* and *sch* can contain XML Schema or Schematron validation specifications. The file *webapp.xml* contains further configuration of your web application.

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*: see chapter *5 Development mode and production mode*.

- *Resources*: The definition of requests to static files that should not be processed by the request dispatcher (but should be served straight away) and the duration these resources should be cached by the browser (default 4 hours).
- *Parameters*: The definition of webapp specific configuration parameters that are passed as stylesheet parameters to every XSLT transformation, see chapter *3 Stylesheet parameters*.
- *Jobs*: The definition of scheduled jobs, see chapter *8 Job scheduling*.
- *Data sources*: the definition of JDBC data sources.
- *FOP configurations*: configurations for the Apache FOP serialization step (see section 2.3.4).

See *Appendix* C: Webapp XML example for an example of a webapp.xml configuration.

2   Pipeline steps

2.1   Transformation pipeline steps

The following transformation pipeline steps are available:

- transformer
- query
- transformer-stx

2.1.1   transformer step

The transformer step transforms the input of the pipeline step using an XSLT version 1.0, 2.0 or 3.0 stylesheet.

Example:

```
<pipeline:pipeline>
  <pipeline:transformer name="my-xsl-step" xsl-path="my-stylesheet.xsl"/>
</pipeline:pipeline>
```

2.1.2   query step

The query step processes the input of the pipeline step using an XQuery version 1.0, 3.0 or 3.1 query.

Example:

```
<pipeline:pipeline>
  <pipeline:transformer name="my-xquery-step" xquery-path="my-query.xq"/>
</pipeline:pipeline>
```

See also example 27 of the *examples* webapp

2.1.3   transformer-stx step

The transformer-stx step transforms the input of the pipeline step using a STX (Streaming Transformations for XML) version 1.0 stylesheet.

Example:

```
<pipeline:pipeline>
  <pipeline:transformer-stx name="my-stx-step" stx-path="my-stylesheet.stx"/>
</pipeline:pipeline>
```

See also example 28 of the *examples* webapp

These steps could be combined in a pipeline as follows:

```
<pipeline:pipeline>
  <pipeline:transformer-stx name="my-stx-step" stx-path="my-stylesheet.stx"/>
  <pipeline:transformer name="my-xsl-step" xsl-path="my-stylesheet.xsl"/>
  <pipeline:query name="my-xquery-step" xquery-path="my-query.xq"/>
</pipeline:pipeline>
```

2.1.4   Extensions

XSLWeb extends the standard functionality of the transformation steps in a number of ways:

- XSLWeb provides a number of built-in XPath extension functions to the *transformer* and *query* steps (not the *transformer-stx* step) that you can use to read and write files and directories, execute HTTP requests, access the Request, Response and Context, Session and

WebApp objects, log messages, send e-mails and so on, see chapter *11: XPath extension function library*.

- Other pipelines can be called from within a stylesheet or query and the result of this nested pipeline can be used or embedded in the calling stylesheet/query by passing a URI that starts with the scheme "xslweb://" to the standard XSLT/XQuery *document()* function or the STX *stx:process-children* element, see chapter *4 Nested pipelines*.
- Within every transformation of query a number of standard stylesheet parameters is available, see chapter *3: Stylesheet parameters*.

2.2   Validation pipeline steps

XSLWeb supports the XML validation of the output of a transformation pipeline step by adding a validation pipeline step after the transformation step.

The following validation pipeline steps are available:

- schema-validator
- schematron-validator

2.2.1   schema-validator step

The schema-validator step validates the output of the previous step using an XML Schema version 1.0.

```
<pipeline:pipeline>
  <pipeline:transformer name="my-transformation " xsl-path="my-transormation.xsl"/>
  <pipeline:schema-validator
    name="schema-validator"
    xsl-param-namespace="http://www.armatiek.com/xslweb/validation"
    xsl-param-name="schema-validation-report">
    <pipeline:schema-paths>
      <pipeline:schema-path>my-schema.xsd</pipeline:schema-path>
    </pipeline:schema-paths>
  </pipeline:schema-validator>
</pipeline:pipeline>
```

The location(s) of the XML schemas can be specified in the subelements *schema-path*. These paths must be relative to the directory *«xslweb-home»/xsd.*

Any validation warnings and errors are written to the log file. If you specify the attribute *xsl-param-name* (and optional attribute *xsl-param-namespace*), a validation report (in XML format) is added as a stylesheet parameter of type document-node() to the next XSLT transformation step in the pipeline.

Validation properties (like `http://javax.xml.XMLConstants/property/accessExternalSchema`) and features (like `http://javax.xml.XMLConstants/feature/secure-processing`) can be specified in the *features* and *properties* subelements (see pipeline.xsd).

See also example 25 of the *examples* webapp.

2.2.2   schematron-validator step

The Schematron-validator step validates the output of the previous step using an ISO Schematron schema:

```
<pipeline:pipeline>
  <pipeline:transformer name="my-transformation " xsl-path="my-transormation.xsl"/>
  <pipeline:schematron-validator
    name="schematron-validator"
    schematron-path="my-schematron.sch "
    xsl-param-namespace="http://www.armatiek.com/xslweb/validation"
    xsl-param-name="schematron-validation-report">
  </pipeline:schematron-validator>
</pipeline:pipeline>
```

The location of the Schematron schema can be specified in the attribute *schematron-path*. This path must be relative to the directory *«xslweb-home»/sch.*

Any validation warnings and errors are written to the log file. If you specify the attribute *xsl-param-name* (and optional attribute *xsl-param-namespace*), the validation report (in [SVRL format](#)) is added as a stylesheet parameter of type document-node() to the next XSLT transformation step in the pipeline.

The Schematron phase can be specified using the optional attribute *phase* on the element *schematron-validator* (see pipeline.xsd).

See also example 25 of the *examples* webapp.

2.3 Serialization pipeline steps

The way the result of the transformation pipeline steps is serialized to XML, XHTML, HTML or text can be specified by the serialization attributes of the element *xsl:output* in the last stylesheet or query of the pipeline, using the attributes *method*, *encoding*, *indent*, *omit-xml-declaration* and so on.

In case the output of the pipeline should not be XML, XHTML, HTML or text, a specific serialization pipeline step can be added at the end of the pipeline. XSLWeb provides the serialization steps:

- json-serializer
- zip-serializer
- resource-serializer
- fop-serializer

2.3.1 json-serializer step

The json-serializer step serializes XML to a [JSON](#) representation. This step can be added as the last step in a pipeline like this:

```
<pipeline:pipeline>
  <pipeline:transformer
    name="my-transformation "
    xsl-path="my-transformation.xsl"/>
  <pipeline:json-serializer
    name="json-serialization"
    auto-array="false"
    pretty-print="true">
  <pipeline:json-serializer/>
</pipeline:pipeline>
```

The pipeline step supports the following attributes:

- auto-array
- auto-primitive
- multi-pi
- namespace-declarations

- namespace-separator
- pretty-print
- virtual-root-namespace
- virtual-root-name
- repairing-namespaces

See for an explanation of these properties the documentation of StAXON.

Namespace declarations can be specified by adding namespace-declaration elements under the namespace-declarations sub element of json-serializer (see pipeline.xsd).

See also example 18 of the *examples* webapp.

2.3.2   zip-serializer step

The zip-serializer step serializes an XML representation of the contents of a ZIP file to the actual file.

A ZIP serializer pipeline step can be added as the last step in a pipeline like this:

```
<pipeline:pipeline>
  <pipeline:transformer
    name="my-zip-serialization"
    xsl-path="my-zip-serialization.xsl"/>
  <pipeline:zip-serializer name="zip"/>
</pipeline:pipeline>
```

The last transformation step in the pipeline has to generate a response like the following XML:

```
<resp:response status="200">
  <resp:body>
    <zip:zip-serializer
      xmlns:zip="http://www.armatiek.com/xslweb/zip-serializer">
      <zip:file-entry
        name="file/myfile.txt"
        src="/home/john/myfile.txt"/>
      <zip:inline-entry
        name="dir1/test.xml"
        method="xml"
        encoding="UTF-8"
        omit-xml-declaration="no"
        indent="yes">
        <a>
          <b>Hello World</b>
        </a>
      </zip:inline-entry>
    </zip:zip-serializer>
  </resp:body>
</resp:response>
```

The element zip-serializer can contain two elements:

- **zip:file-entry**: a representation of a disk file that must be serialized to the zip file. The attribute "src" holds the path to the file, the attribute "name" holds the name (path) of the file in the serialized zip file.
- **zip:inline-entry**: an inline XML, HTML or text fragment that must be serialized to the ZIP file. The attribute "name" holds the name (path) of the file in the serialized zip file. Other attributes specify the serialization behavior and are the same as the attributes of xsl:output.

See also example 23 of the *examples* webapp.

2.3.3   resource-serializer step

The resource serializer returns a stored text or binary file to the client. It is typically used in scenarios where a file is dynamically generated during pipeline execution (for instance by using the extension function *exec-external()*) and then must be returned to the client in the same request. Another scenario is to create download links to static files that are located on the server outside the scope of the webapp's "static" directory.

The resource serializer pipeline step can be added as the last step in a pipeline like this:

```
<pipeline:pipeline>
  <pipeline:transformer
    name="resource-serialization"
    xsl-path="resource-serialization.xsl"/>
  <pipeline:resource-serializer name="resource"/>
</pipeline:pipeline>
```

In this example the stylesheet *resource-serialization.xsl* must generate a XML fragment containing information that the resource serializer uses to create the desired response. The stylesheet must generate an element *resource-serializer* in the namespace *http://www.armatiek.com/xslweb/resource-serializer*. On this element the following attributes can be set:

- **path**: the local path to the file that must be returned to the client.
- **content-type** (optional): the content type (mime type) that must be set on the response. If this attribute is not set, XSLWeb will try to determine the content type automatically.
- **content-disposition-filename** (optional): this attribute can be used to force the browser to display a "Save as" dialog (instead of display the file "inline"). The specified filename will be used as the default filename in de dialog.

An example of the output of the XSLT stylesheet *resource-serialization.xsl* is:

```
<res:resource-serializer
  path="webapps/examples/xsl/resource/leaves.jpg"
  content-type="image/jpg"
  content-disposition-filename="my-image.jpg"/>
```

See also example 26 of the *examples* webapp

2.3.4   fop-serializer step

The fop-serializer step serializes XSL:FO format generated in a previous pipeline step to document formats like PDF or RTF using the Apache FOP processor version 2.2.

The FOP serializer pipeline step can be added as the last step in a pipeline like this:

```
<pipeline:pipeline>
  <pipeline:transformer
    name="my-xsl-fo-serialization"
    xsl-path="my-xsl-fo-serialization.xsl"/>
  <pipeline:fop-serializer name="fop-serialization"/>
</pipeline:pipeline>
```

The last transformation step in the pipeline has to generate a response like the following XML:

```xml
<resp:response status="200">
  <resp:body>
    <fop:fop-serializer
       xmlns:fop="http://www.armatiek.com/xslweb/fop-serializer"
       output-format="application/pdf"
       config-name="default">
       <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
         <!-- Your further XSL:FO code -->
       </fo:root>
    </fop:fop-serializer>
  </resp:body>
</resp:response>
```

The fop:fop-serializer element supports the following attributes:

- **config-name**: the name of a FOP configuration in webapp.xml.
- **output-format** (optional): the output format of the serialization, like "application/pdf" (default), application/postscript, application/rtf (see the FOP class MimeConstants.java).
- **pdf-a-mode** (optional): specify a PDF/A profile:
  - PDF/A-1a
  - PDF/A-1b
  - PDF/A-2a
  - PDF/A-2b
  - PDF/A-2u
  - PDF/A-3a
  - PDF/A-3b
  - PDF/A-3u

See also example 24 of the *examples* webapp.

3 Stylesheet parameters

Every XSLT stylesheet that is executed within XSLWeb is provided with a number of stylesheet parameters:

- The configuration parameters from the parameters section in the *webapp.xml*. The parameter's local name can be given a namespace using the attribute *uri* and the type of the values can be specified using the attribute *type*. The value itself can be a sequence of atomic values.
- *config:home-dir*: the path to the XSLWeb home directory (config = http://www.armatiek.com/xslweb/configuration)
- *config:webapp-dir*: the path to the base directory of the webapp.
- *config:webapp-path*: The path in de url to the web application ("/" for the webapp *ROOT* and "/" + *webapp-name* for other webapps).
- *config:development-mode*: whether the webapp runs in development-mode or production-mode.
- The Java HttpServletRequest, HttpServletResponse and WebApp objects. These can be used in custom XPath extension functions.

Pipeline stylesheets are also provided with any parameters that are defined within the element *pipeline:transformer* in *request-dispatcher.xsl*. The parameter's local name can be given a namespace

using the attribute *uri* and the type of the values can be specified using the attribute *type*. The value itself can be a sequence of atomic values.

The parameters only have to be declared in the stylesheets (as *<xsl:param/>* elements) when they are actually used. The parameters for the Java objects doesn't have to be declared at all.

4   Nested pipelines

It is possible to call another pipeline from a stylesheet using the standard XSLT function *document()* providing an URL that starts with the scheme *xslweb*, for instance:

```
<xsl:sequence select="document('xslweb:///examples/nestedpipeline.html')"/>
```

where *examples* is the name of the webapp of the nested pipeline. The result of the nested pipeline will be available in the calling stylesheet as a document node. The nested pipeline request will follow the flow of a normal HTTP request, including the request dispatcher stylesheet. A nested pipeline call can be seen as an "internal request", it does not go through the HTTP stack.

5   Development mode 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 caching framework (using the cache attributes on the pipeline element) is disabled, so no caching is performed.
- 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 pipeline is executed. 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.
- In development mode, the generated XSLT of a Schematron schema is logged to the log file (with severity INFO).

6   Logging

Log files are stored in the directory *«xslweb-home»/logs*. This directory contains two log files, *xslweb.log* and *pipeline.log*.

Regular XSLWeb specific log messages are logged to *xslweb.log*. It's also possible to write to this log file from web application stylesheets using the XPath extension function *log:log()*, see paragraph 11.1.10.

In development mode a pipeline step can be configured to log its (intermediate) output to the log file *pipeline.log*, by specifying *log="true"* on the pipeline step.

By default the log files are rotated when they reach the size of 10Mb, and a maximum of 8 backups is retained.

XSLWeb makes use of the standard logging framework slf4j with logback. The rotation, backup and other settings can be configured in the configuration file *«xslweb-home»/config/logback.xml*.

7   Response caching

The output of a pipeline can be cached by providing optional caching attributes on the element *pipeline:pipeline* in the stylesheet *request-dispatcher.xsl*. The purpose of caching the response output is to gain performance; a response that can be served from cache will be returned quicker because no transformations are necessary and also the load on the server is decreased.

The following attributes are supported:

- *cache* (xs:boolean): specifies whether the output of the response must be cache. Default: false.
- *cache-key* (xs:string): specifies the key under which the output of the pipeline must be cached, default the concatenation of req:method, req:request-URI and req:query-string. It is only necessary to override the default mechanism if for instance the query string contains parameters that are different for every request, like with tracking software.
- *cache-time-to-live* (xs:integer): The number of seconds the output will be cached from the time it was first added to the cache. Default: 60 seconds.
- *cache-time-to-idle*: (xs:integer): The number of seconds the output will be cached from the last time it was actually used. Default": 60 seconds.
- *cache-scope* (xs:string): One of "webapp" or "user". It specifies whether the output should be cached and reused by all users of the web application ("webapp"), or for a specific user ("user"). Default "webapp".
- *cache-headers* (xs:boolean): Specifies whether XSLWeb should automatically provide the HTTP response cache headers: *ETag*, *Last-Modified* and *Expires*. It supports conditional GET. Because browsers and other HTTP clients have the expiry information returned in the response headers, they do not even need to request the page again. Even once the local browser copy has expired, the browser will do a conditional GET. Default: false.

XSLWeb uses the standard caching framework Ehcache to support its caching (see http://ehcache.org). More advanced configuration properties can be specified in the Ehcache specific configuration file *«xslweb-home»/config/xslweb-ehcache.xml*, like for instance how many responses should be cached in memory and how many on disk. See the ehcache documentation for further details.

N.B. Response caching is only enabled in production mode, see chapter *5 Development mode and production mode*.

8 Job scheduling

When you want to execute a pipeline (repeatedly) on a certain moment without user interaction, you can use the job scheduling functionality of XSLWeb. The jobs can be defined and scheduled in the *webapp.xml* configuration file, for example:

```xml
<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>
  <concurrent>false</concurrent>
</job>
```

The elements have the following meaning:

- *name*: the name of the scheduled job. Used in log files.
- *uri*: the Uri of the request to a pipeline within the current webapp. This internal request will follow the same flow of a normal HTTP request, including the request dispatcher stylesheet. The Uri does not contain the name of the webapp.
- *cron*: the cron expression which is a string comprising five or six fields separated by white space that represents a set of times to execute the job (see http://en.wikipedia.org/wiki/Cron#CRON_expression).
- *concurrent*: specifies whether or not the job can run concurrently with other jobs.

9 User authentication

You can implement (BASIC) user authentication by performing the following steps:

- Include the stylesheet *«xslweb-home»/xsl/system/authentication/basic/basic-authentication.xsl* in your *request-dispatcher.xsl* stylesheet.
- Implement the function *auth:must-authenticate($request as element(request:request)): xs:boolean*. In this function you can determine whether *$request* must be authenticated or not.
- Implement the function *auth:get-realm(): xs:string*. This function must return the authentication realm.
- Implement the function *auth:login($username as xs:string, $password as xs:string): element()?*. This function must authenticate $username with $password and return an empty sequence if the authentication failed or an element() containing the user profile if authentication succeeded. This element must have the name *authentication* and a subelement *ID*. The element *data* can be filled with arbitrary data you will need in subsequent requests.
- This element will be stored by XSLWeb in the user's session object under the name *xslweb-userprofile* so it will be available in subsequent requests.

N.B. DIGEST or other authentication methods are not yet supported. BASIC (and DIGEST) authentication is only secure if HTTPS is used!

10 Global configuration properties

In *«xslweb-home»/config/xslweb.properties* two global properties can be set:

- **xslweb.trustallcerts**: specifies if all SSL certificates must be trusted when XSLWeb connects to an external HTTPS server.

- **xslweb.parserhardening**: specifies if the Xerces XML parser must be configured to resist [XML External Entity (XXE)](#) attacks.

11 XPath extension function library

11.1.1 Built in extension functions

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

11.1.2 Response functions

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

Functions:

```
add-cookie(element(response:cookie)) as xs:boolean?

add-date-header($name as xs:string, $value as xs:dateTime) as xs:boolean?

add-int-header($name as xs:string, $value as xs:integer) as xs:boolean?

add-header($name as xs:string, $value as xs:string) as xs:boolean?

encode-redirect-url($url as xs:string) as xs:string

encode-url($url as xs:string) as xs:string

is-committed() as xs:boolean

set-buffer-size($size as xs:integer) as xs:boolean?

set-status($status as xs:integer) as xs:boolean?
```

See example 5 how to use the response functions to set cookies.

11.1.3 Session functions

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

Functions:

```
attribute-names() as xs:string*

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

invalidate() as xs:boolean?

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

set-max-active-interval($interval as xs:integer) as xs:boolean?
```

See example 7 how to use the session functions to set and get session attributes.

11.1.4 Webapp functions

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

Functions:

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

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

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

```
set-cache-value($cache-name as xs:string,
                $key-name as xs:string,
                $attrs as item()*,
                $time-to-idle as xs:integer,
                $time-to-live as xs:integer) as xs:boolean?
```

See example 7 how to use the webapp functions to set and get webapp attributes, and example 14 how to use the caching functions.

11.1.5 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?
```

See example 7 how to use the context functions to set and get session attributes.

11.1.6 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. The API is described [here](#).

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
```

The structure of *element(output:serialization-parameters)* is described in [XSLT and XQuery Serialization 3.0.](#) See example 10 how to use some of the EXPath File functions.

11.1.7 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. The API is described [here](#).

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 consume both SOAP and REST based web services.
- Request 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()+
```

See examples 11 and 21 how to use some of the EXPath HTTP Client function.

11.1.8 Base64

Namespace: *http://www.armatiek.com/xslweb/functions/base64*

Functions:

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

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

11.1.9 Execute external processes

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

Functions:

```
exec-external(
  $command-line as xs:string,
  $args as xs:string*,
  $exit-value as xs:integer?,
  $time-out as xs:integer?
  $async as xs:boolean) as xs:integer
```

Where $command-line is the path to the executable, $args a sequence of arguments to the application, $exit-value the exit code that is considered as success, $time-out the time in milliseconds after an asynchronous process is killed and $async indicates if the process must be started asynchronous or not.

11.1.10   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". The structure of *element(output:serialization-parameters)* is described in XSLT and XQuery Serialization 3.0. See example 15 how to use some of the log functions.

11.1.11   Email

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

Functions:

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

See example 12 how to use the send-email example and an example of the structure of *element(email:email)*.

11.1.12   Serialization

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

Functions:

```
serialize($nodes as node()*, $options as element(output:serialization-
parameters)?) as xs:string
```

The structure of *element(output:serialization-parameters)* is described in XSLT and XQuery Serialization 3.0.

See example 11 how to use the serialize function.

11.1.13 Cache

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

Functions:

Remove a cache entry from the response output cache:

```
remove($cache-key as xs:string) as xs:boolean?
```

11.1.14 Image processing

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

Functions:

Resizes an image and optionally convert it to another format:

```
scale(
  $source as xs:string,
  $target as xs:string,
  $format-name as xs:string,
  $target-size as xs:integer) as xs:boolean?
```

Where $source is the path or url to the source image, $target the path to the scaled image, $format-name the name of the target format (like png, gif, jpg) and $target-size the maximum image width or height of the scaled image.

11.1.15 Input/Output

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

Functions:

Registers a temporary file or directory that will automatically be deleted after the pipeline has executed.

```
register-temp-file($path as xs:string) as xs:boolean?
```

11.1.16 Utilities

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

Functions:

Remove supplied document from memory pool so it will be released by the Java garbage collector:

```
discard-document($document-node()) as document-node()
```

Parse a XML string to a document node:

```
parse($serialized-xml as xs:string) as document-node()
```

11.1.17 Zip

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

Functions:

Zip a file on path $source to a new file on path $target:

```
zip($source as xs:string, $target as xs:string) as xs:boolean?
```

Unzip a file on path or url $source to a new file on path $target:

```
unzip($source as xs:string, $target as xs:string) as xs:boolean?
```

See also section **Error! Reference source not found. Error! Reference source not found.**.

11.1.18 UUID

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

Functions:

Generate a universally unique identifier:

```
uuid() as xs:string
```

11.1.19 JSON
(Experimental)

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

Functions:

```
serialize-json($items as item()*) as xs:string

parse-json($json as xs:string) as document-node()?
```

See also section **Error! Reference source not found. Error! Reference source not found.**.

11.1.20 Script
(Experimental)

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

Functions:

```
invoke($script as xs:string,
       $function-name as xs:string,
       $arg1 as anyAtomicType*,
       $arg2 as anyAtomicType*,
       $arg3 as anyAtomicType*,
       $arg4 as anyAtomicType*,
       $arg5 as anyAtomicType*,
       $arg6 as anyAtomicType*,
       $arg7 as anyAtomicType*,
       $arg8 as anyAtomicType*) as anyAtomicType*
```

Executes a function named *$function-name* in the Javascript code *$script*. The Javascript function must have the following signature:

```
function function-name(context, webapp, request, response, arg1, arg2,
arg3, arg4, arg5, arg6, arg7, arg8)
```

The first argument of the Javascipt function is the XSLWeb Context object, the second the XSLWeb WebApp object, the third the Java EE HttpServletRequest object and the fourth argument the Java EE HttpServletResponse object. The $arg* parameters of the *invoke* extension function must correspond to the fifth and higher arguments of the Javascript function and are all optional. The arguments are converted to Javascript arrays. Only sequences of atomic types can be used as arguments. The result of the Javascript function must be a primitive type or Javascript array containing primitive types. Within the Javascript, Java objects can be instantiated by using *JavaImporter* (see example 20).

XSLWeb uses the Mozilla Rhino scripting engine on Java 7 (see [documentation](#)) and the Oracle Nashorn scripting engine on Java 8 (see [documentation](#)).

See example 20 how to use some of the script extension functions.

11.1.21   SQL

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

Functions:

```
sql:close($connection as java.sql.Connection
          or $resultset as java.sql.ResultSet) as xs:boolean?

sql:commit($connection as java.sql.ResultSet) as xs:boolean?

sql:execute-query($connection as java.sql.Connection,
                  sql as xs:string) as java.sql.ResultSet

sql:get-connection($name as xs:string,
                   $username as xs:string?,
                   $password as xs:string?,
                   $readonly as xs:boolean?,
                   $autocommit as xs:boolean?) as java.sql.Connection

sql:get-next-row($resultset as java.sql.ResultSet) as xs:anyAtomicType*

sql:rollback($connection as java.sql.Connection) as xs:boolean?

sql:resultset-to-node($resultset as java.sql.ResultSet) as element()
```

The SQL extension functions make use of the [Java JDBC](#) framework. You can place the JDBC driver for your database in the directory *«xslweb-home»/common/lib* (restart required). Then you can define a datasource in the *datasources* section of the application's *webapp.xml*, for instance for a [H2](#) database:

```xml
<datasource>
  <name>my-datasource</name>
  <driver-class>org.h2.Driver</driver-class>
  <jdbc-url>jdbc:h2:file://${webapp-dir}/database/my-database.mv</jdbc-url>
  <property name="user">sa</property>
</datasource>
```

With the extension function *sql:get-connection("my-datasource")* a database connection can be retrieved from the connection pool. This connection then can be used to execute one or more queries.

The connection pool is implemented using the Java framework [c3p0](#). The c3p0 specific properties that can be used in the datasource definition in *webapp.xml* are described [here](#). De default c3p0 properties can be configured in the configuration file *«xslweb-home»/config/c3p0-config.xml*.

Connections and resultsets can be explicitly closed by using the extension function *sql:close()*. The *sql:close()* function on a connection will also return the connection to the connection pool. Connections and resultsets that are not closed that way will be implicitly closed at the end of the pipeline, and connections will be returned to the connection pool automatically.

JDBC drivers are available for most relational database systems (like MySQL, Oracle, PostgreSQL, MSSQL Server, Sybase, Cloudscape and Firebird), but also for non-relational database datasources like CSV files and LDAP directory services (untested).

See example 22 how to use some of the sql extension functions.

11.1.22   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 9.7.

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.

11.1.23   Extension functions with side effects

A number of the extension functions described in previous sections perform a certain task and thereby change the state of something outside the stylesheet, like write or log to a file, send an e-mail etc. These functions don't have any return information and should have an empty sequence as their return type. In XSLWeb, the return type of these functions is actually declared as *xs:boolean?* The reason is that in that case the Saxon XSLT optimizer cannot ignore these functions, because they could add something to the result tree (a boolean value). In reality, these functions never return this boolean value and always return an empty sequence. Therefore it is safe to do something like:

```
<xsl:sequence select="log:log('INFO', 'Hello World!')"/>
```

without having to worry that something is written to the result tree.

12 Download, install and run XSLWeb

The sources of XSLWeb can be found on GitHub. XSLWeb is licensed under the Apache License version 2.0.

Compiled binaries of XSLWeb 3.0 can be downloaded in two distributions:

1. As a .zip (Windows) or .tgz (Linux/OSX) archive, containing a single executable java library (.jar) with embedded application server (Apache Tomcat 7).
2. As a web application archive (.war)

12.1 The single executable jar distribution

This distribution contains the XSLWeb home directory and a single executable java library (xslweb.jar) with embedded application server (Tomcat 7). This distribution is the easiest to install and run and is most suitable to try out and develop applications in XSLWeb.

12.1.1 Download

You can download the archives from the following locations:

- http://www.armatiek.nl/downloads/xslweb/xslweb-3.0.0-jar.zip (Windows)
- http://www.armatiek.nl/downloads/xslweb/xslweb-3.0.0-jar.tgz (Linux/OSX).

12.1.2 Install

Extract the archive to a directory of your choice. You will need the Java Runtime Environment (JRE) version 1.8 or higher. At a command line, check your Java version like this:

```
$ java -version
```

```
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
```

The output will vary, but you need to make sure you have version 1.8 or higher. If you don't have the required version, or if the java command is not found, download and install the latest version from Oracle at http://www.oracle.com/technetwork/java/javase/downloads/index.html.

12.1.3 Run

The embedded jar version of XSLWeb can be started from the directory where the archive was extracted using the following command (one line):

```
java -Dxslweb.home=./home -cp xslweb.jar;home/config
org.apache.tomcat.maven.runner.Tomcat7RunnerCli
```

The following additional arguments are supported:

```
-ajpPort <ajpPort>                      ajp port to use
-clientAuth                             enable client authentication for
                                        https
-D <arg>                                key=value
-extractDirectory <extractDirectory>    path to extract war content,
                                        default value: .extract
-h,--help                               help
-httpPort <httpPort>                    http port to use
-httpProtocol <httpProtocol>            http protocol to use: HTTP/1.1 or
                                        org.apache.coyote.http11.Http11Nio
                                        Protocol
-httpsPort <httpsPort>                  https port to use
-maxPostSize <maxPostSize>              max post size in bytes to use
-keyAlias <keyAlias>                    alias from keystore for ssl
-loggerName <loggerName>                logger to use: slf4j to use slf4j
                                        bridge on top of jul
-obfuscate <password>                   obfuscate the password and exit
-resetExtract                           clean previous extract directory
-serverXmlPath <serverXmlPath>          server.xml to use, optional
-X,--debug                              debug
```

Now you can open a browser and go to the address http://localhost:8080. A web page with the text "It works!" should appear. From here you can go to the examples and the documentation.

12.2 The Web Application Archive (.war) distribution

This distribution contains the XSLWeb home directory and a web application archive (.war) and is most suitable to run XSLWeb in a production setting. The web application archive is a standard J2EE web application that can be installed on any Java application server that supports Servlet Spec 3.0 (Tomcat 7+, TomEE 1.6+, WebLogic, Jetty 8+, Glassfish 3+, JBoss AS 6.x/7.x etc.).

12.2.1 Download

You can download the archives from the following locations:

- http://www.armatiek.nl/downloads/xslweb/xslweb-3.0.0-war.zip (Windows)
- http://www.armatiek.nl/downloads/xslweb/xslweb-3.0.0-war.tgz (Linux/OSX)

12.2.2 Install

Extract the archive to a directory of your choice. This directory will contain the directory xslweb-3.0 containing a directory called *home* and a file called *xslweb.war*. The installation of a war is application

server specific, so please consult the manual of your server for that. Regardless of which application server is used, two settings are essential:

1. The home directory of XSLWeb must be specified using a Java System Property called *xslweb.home*.
2. The path *«xslweb-home»/config* must be added to the Java classpath.

12.2.3 Run

Start your application server. Open a web browser and go to the address:

http://localhost:<port>/xslweb

where port is the port your application server runs on. A web page with the text "It works!" should appear. From here you can go to the examples and the documentation.

12.3 Install XSLWeb as a Windows service or Linux/OSX daemon

One way to install XSLWeb as a Windows service or Linux/OSX daemon is to use YAJSW (Yet Another Java Service Wrapper). This installation involves the following steps:

1. Execute the steps described in sections 12.1.1 and 12.1.2.
2. Download YAJSW version 12.11+ from https://sourceforge.net/projects/yajsw/files/ and extract the archive in the directory created in step 1.
3. Create YAJSW wrapper config file *wrapper.xslweb.conf* and place it in YAJSW's *conf* folder. You can use the example below. You may need to modify this example to:
   o Specify the location of java (wrapper.java.command)
   o Specify the installation directory of XSLWeb (wrapper.working.dir)
   o Change the amount of memory available to XSLWeb from 1024m (for example, 512m or 2048m)

```
# YAJSW configuration for XSLWeb

wrapper.java.command=C:/ProgramData/Oracle/Java/javapath/java.exe

wrapper.working.dir=D:\\xslweb-3.0

wrapper.java.app.mainclass=org.apache.tomcat.maven.runner.Tomcat7Run
nerCli

wrapper.java.classpath.1 = xslweb.jar
wrapper.java.classpath.2 = home/config

wrapper.java.additional.1 = -Xmx1024m
wrapper.java.additional.2 = -Dfile.encoding=utf-8
wrapper.java.additional.3 = -Dxslweb.home=./home
wrapper.java.additional.4 = -
Dorg.terracotta.quartz.skipUpdateCheck=true

wrapper.ntservice.name= XSLWEB_3_0
wrapper.ntservice.displayname= XSLWeb 3.0
wrapper.ntservice.description= XSLWeb 3.0
wrapper.ntservice.starttype=DELAYED_AUTO_START
```

```
wrapper.console.loglevel=INFO
wrapper.logfile=${wrapper.working.dir}/home/logs/wrapper-xslweb.log
wrapper.logfile.maxsize=10m
wrapper.logfile.maxfiles=10

wrapper.on_exit.0=SHUTDOWN
wrapper.on_exit.default=RESTART
```

You can add any optional arguments mentioned in section 12.1.3 as *wrapper.app.parameter.NN.*

After you have created the wrapper configuration file:

1. Open a command prompt as administrator (Windows) or shell (Linux/OSX)
2. Navigate to the YAJSW folder
3. Install the service:
   ```
   java -jar wrapper.jar --install conf\wrapper.xslweb.conf
   ```
4. Start the service:
   ```
   java -jar wrapper.jar --start conf\wrapper.xslweb.conf
   ```

XSLWeb is now running as a service/daemon, and will start automatically when the operating system starts. Open a web browser and go to the address:

http://localhost:8080

A web page with the text "It works!" should appear. From here you can go to the examples and the documentation.

If you make changes to the configuration you can follow this sequence:

1. Stop the service:
   ```
   java -jar wrapper.jar --stop conf\wrapper.xslweb.conf
   ```
2. Remove the service:
   ```
   java -jar wrapper.jar --remove conf\wrapper.xslweb.conf
   ```
3. Make your changes to the wrapper or application configuration.
4. Install the service:
   ```
   java -jar wrapper.jar --install conf\wrapper.xslweb.conf
   ```
5. Start the service:
   ```
   java -jar wrapper.jar --start conf\wrapper.xslweb.conf
   ```

13 Support for Saxon PE (Professional) and EE (Enterprise Edition)

The downloadable binaries of XSLWeb contain the open source Home Edition (HE) of the Saxon XSLT processor. You can build a version of XSLWeb that contains Saxon PE or EE by following these steps:

- Install Java 1.8+, maven 2.2.1+ and make a clone of the git repository https://github.com/Armatiek/xslweb.git.
- Purchase a PE or EE license from http://www.saxonica.com/.
- Download the PE of EE package from http://www.saxonica.com/download/SaxonPE9-8-0-8J.zip or http://www.saxonica.com/download/SaxonEE9-8-0-8J.zip
- Extract and register the jars in your local Maven repository using the following commands:
  - o PE:

- ▪ `mvn install:install-file -Dfile=saxon9pe.jar -DgroupId=net.sf.saxon -DartifactId=Saxon-PE -Dversion=9.8.0-8 -Dpackaging=jar`
  - ▪ `mvn install:install-file -Dfile=saxon9-icu.jar -DgroupId=net.sf.saxon -DartifactId=Saxon-PE-icu -Dversion=9.8.0-8 -Dpackaging=jar`
  - ▪ `mvn install:install-file -Dfile=saxon9-sql.jar -DgroupId=net.sf.saxon -DartifactId=Saxon-PE-sql -Dversion=9.8.0-8 -Dpackaging=jar`
  - o EE:
    - ▪ `mvn install:install-file -Dfile=saxon9ee.jar -DgroupId=net.sf.saxon -DartifactId=Saxon-EE -Dversion=9.8.0-8 -Dpackaging=jar`
    - ▪ `mvn install:install-file -Dfile=saxon9-icu.jar -DgroupId=net.sf.saxon -DartifactId=Saxon-EE-icu -Dversion=9.8.0-8 -Dpackaging=jar`
    - ▪ `mvn install:install-file -Dfile=saxon9-sql.jar -DgroupId=net.sf.saxon -DartifactId=Saxon-EE-sql -Dversion=9.8.0-8 -Dpackaging=jar`

- Place your purchased license file *saxon-license.lic* in <<xslweb.home>>/config
- Build XSLWeb with the maven profile "Saxon-PE" or "Saxon-EE":
  - o `mvn -PSaxon-PE clean install`
    or
  - o `mvn -PSaxon-EE clean install`

14 Appendix A: Request XML example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="http://www.armatiek.com/xslweb/request">
  <character-encoding>UTF-8</character-encoding>
  <content-length>-1</content-length>
  <context-path>/xslweb</context-path>
  <local-addr>127.0.0.1</local-addr>
  <local-name>127.0.0.1</local-name>
  <local-port>8080</local-port>
  <method>GET</method>
  <path>/log/log.html</path>
  <path-info>/examples/log/log.html</path-info>
  <path-translated>D:\webapps\xslweb\examples\log\log.html</path-translated>
  <protocol>HTTP/1.1</protocol>
  <remote-addr>127.0.0.1</remote-addr>
  <remote-host>127.0.0.1</remote-host>
  <remote-port>55451</remote-port>
  <requested-session-id>D5984A4C38D09BE74C04F1D89022AE90</requested-session-id>
  <request-URI>/xslweb/examples/log/log.html</request-URI>
  <request-url>http://localhost:8080/xslweb/examples/log/log.html</request-url>
  <scheme>http</scheme>
  <server-name>localhost</server-name>
  <server-port>8080</server-port>
  <servlet-path/>
  <webapp-path>/examples</webapp-path>
  <is-secure>false</is-secure>
  <is-requested-session-id-from-cookie>true</is-requested-session-id-from-cookie>
  <is-requested-session-id-from-url>false</is-requested-session-id-from-url>
  <is-requested-session-id-valid>true</is-requested-session-id-valid>
  <headers>
    <header name="host">localhost:8080</header>
    <header name="connection">keep-alive</header>
    <header
name="accept">text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;
q=0.8</header>
    <header name="user-agent">Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36</header>
    <header name="referer">http://localhost:8080/xslweb/examples</header>
    <header name="accept-encoding">gzip, deflate, sdch</header>
    <header name="accept-language">nl-NL,nl;q=0.8,en-US;q=0.6,en;q=0.4</header>
    <header name="cookie">JSESSIONID=D5984A4C38D09BE74C04F1D89022AE90</header>
  </headers>
  <parameters>
    <parameter name="country">
      <value>US</value>
    </parameter>
    <parameter name="states">
      <value>AZ</value>
      <value>CA</value>
    </parameter>
  </parameters>
  <!-- If this request was a file upload POST request:
  <file-uploads>
    <file-upload>
      <file-path>C:\Users\John\AppData\Local\Temp\48226ce5-7bba-4986-8d1f-
c4a8f34638cf\MyDocument1.docx</file-path>
      <field-name>file1</field-name>
      <file-name>MyDocument1.docx</file-name>
      <content-type>application/vnd.openxmlformats-
officedocument.wordprocessingml.document</content-type>
```

```xml
      <size>177032</size>
    </file-upload>
  </file-uploads>
  -->
  <session>
    <creation-time>2015-01-06T13:06:04.925+01:00</creation-time>
    <id>D5984A4C38D09BE74C04F1D89022AE90</id>
    <last-accessed-time>2015-01-06T14:36:04.909+01:00</last-accessed-time>
    <max-inactive-interval>1800</max-inactive-interval>
    <is-new>false</is-new>
  </session>
  <cookies>
    <cookie>
      <max-age>-1</max-age>
      <name>JSESSIONID</name>
      <is-secure>false</is-secure>
      <value>D5984A4C38D09BE74C04F1D89022AE90</value>
      <version>0</version>
    </cookie>
  </cookies>
</request>
```

15 Appendix B: Response XML example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<resp:response
  xmlns:resp="http://www.armatiek.com/xslweb/response"
  status="200">
  <resp:headers>
    <resp:header name="Pragma">no-cache</resp:header>
    <resp:int-header name="Expires">0</resp:int-header>
    <resp:date-header
      name="Last-Modified">2006-04-10T13:40:23.83-05:00</resp:date-header>
  </resp:headers>
  <resp:session max-active-interval="1800">
    <resp:attributes>
      <resp:attribute name="msg">
        <item type="xs:string">Hello World</item>
        <item type="node()">
          <msg>Hello World!</msg>
        </item>
      </resp:attribute>
    </resp:attributes>
  </resp:session>
  <resp:cookies>
    <resp:cookie>
      <resp:comment>Comment 1</resp:comment>
      <resp:domain>localhost.com</resp:domain>
      <resp:max-age>-1</resp:max-age>
      <resp:name>cookie-1</resp:name>
      <resp:path>/examples</resp:path>
      <resp:is-secure>false</resp:is-secure>
      <resp:value>cookie-1-value</resp:value>
      <resp:version>0</resp:version>
    </resp:cookie>
  </resp:cookies>
  <resp:body>
    <html xmlns="http://www.w3.org/1999/xhtml">
      <head>
        <title>Hello World!</title>
      </head>
      <body>
        <h1>Hello World</h1>
      </body>
    </html>
  </resp:body>
</resp:response>
```

16 Appendix C: Webapp XML example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<webapp
  xmlns="http://www.armatiek.com/xslweb/webapp"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.armatiek.com/xslweb/webapp
../../config/xsd/xslweb/webapp.xsd">

  <title>XSLWeb examples</title>
  <description>XSLWeb examples</description>
  <development-mode>true</development-mode>

  <!-- Resources to serve straight away: -->
  <resources>
    <resource pattern="/favicon.ico" media-type="image/x-icon"/>
    <resource
      pattern="/(styles|images)/.+\.png"
      media-type="image/png"
      duration="P7DT0H0M0S"/>
    <resource pattern="/(styles|images)/.+\.gif" media-type="image/gif"/>
    <resource pattern="/(styles|images)/.+\.(jpg|jpeg)" media-type="image/jpg"/>
    <resource pattern="/scripts/.+\.js" media-type="text/javascript"/>
    <resource pattern="/styles/.+\.css" media-type="text/css"/>
    <resource pattern="/downloads/.+\.docx?" media-type="application/msword"/>
  </resources>

  <!-- Stylesheet parameters: -->
  <parameters>
    <parameter
      name="hostname"
      uri="http://www.armatiek.com/xslweb/functions/email"
      type="xs:string">
      <value>smtp.googlemail.com</value>
    </parameter>
    <parameter
      name="port"
      uri="http://www.armatiek.com/xslweb/functions/email"
      type="xs:integer">
      <value>465</value>
    </parameter>
    <parameter
      name="username"
      uri="http://www.armatiek.com/xslweb/functions/email"
      type="xs:string">
      <value>MYUSERNAME</value>
    </parameter>
    <parameter
      name="password"
      uri="http://www.armatiek.com/xslweb/functions/email"
      type="xs:string">
      <value>MYPASSWORD</value>
    </parameter>
    <parameter
      name="use-ssl"
      uri="http://www.armatiek.com/xslweb/functions/email"
      type="xs:boolean">
      <value>true</value>
    </parameter>
  </parameters>
```

```xml
    <!-- Scheduled job definitions: -->
    <jobs>
      <job>
        <name>WriteTimeJob</name>
        <uri>execute-writetime-job.html</uri>
        <!-- Execute every 60 seconds: -->
        <cron>0/60 * * * * ?</cron>
        <concurrent>true</concurrent>
      </job>
    </jobs>

    <datasources>
      <datasource>
        <name>datasource-worldcup</name>
        <driver-class>org.h2.Driver</driver-class>
        <jdbc-url>jdbc:h2:file://${webapp-dir}/xsl/relational-
database/worldcup.mv</jdbc-url>
        <property name="user">sa</property>
      </datasource>
    </datasources>

</webapp>
```