

# High Performance XML/XSLT Transformation Server

Design document

Zixun Lu (luzi), Shuai Peng (pengs), Elijah Voigt (voigte)

CS 461 | CS Senior Capstone | Group #40

Fall 2016

February 15, 2017

## **Abstract**

The design document for the XML/XSLT document transformer “XZES40-Transformer”, as well as its UI, web-api, and installation packages.

## CONTENTS

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Purpose . . . . .	4
1.2	Scope . . . . .	4
1.3	Development Time-line . . . . .	4
1.4	Summary . . . . .	5
1.5	Issuing organization . . . . .	5
1.6	Change history . . . . .	5
<b>2</b>	<b>References</b>	<b>5</b>
<b>3</b>	<b>Glossary</b>	<b>5</b>
<b>4</b>	<b>Component Overview</b>	<b>6</b>
<b>5</b>	<b>Document Transformer</b>	<b>7</b>
5.1	Transformer . . . . .	7
5.1.1	Context . . . . .	8
5.1.2	Composition . . . . .	8
5.1.3	Dependencies . . . . .	8
5.1.4	State Dynamics . . . . .	8
5.1.5	Interactions . . . . .	8
5.1.6	Interfaces . . . . .	8
5.2	Document class . . . . .	9
5.2.1	Context . . . . .	9
5.2.2	Dependencies . . . . .	9
5.2.3	Interactions . . . . .	9
5.2.4	Resources . . . . .	9
5.2.5	Interfaces . . . . .	9
5.3	Cache . . . . .	9
5.3.1	Context . . . . .	10
5.3.2	Composition . . . . .	10
5.3.3	Logical . . . . .	10
5.3.4	Information . . . . .	10
5.3.5	State Dynamics . . . . .	10
5.3.6	Interactions . . . . .	11
5.3.7	Algorithms . . . . .	11
5.3.8	Resources . . . . .	11
5.3.9	Dependencies . . . . .	11
5.3.10	Interfaces . . . . .	11
5.4	Parallel Computation . . . . .	12

		3
	5.4.1	Context . . . . . 12
	5.4.2	Dependencies . . . . . 12
	5.4.3	Interactions . . . . . 12
	5.4.4	Algorithms . . . . . 12
	5.4.5	Interfaces . . . . . 13
<b>6</b>	<b>Daemon</b>	<b>13</b>
	6.0.1	Context . . . . . 13
	6.0.2	Composition . . . . . 13
	6.0.3	Interfaces . . . . . 13
<b>7</b>	<b>User Interface</b>	<b>13</b>
	7.1	Web-API . . . . . 13
	7.1.1	Context . . . . . 13
	7.1.2	Composition . . . . . 14
	7.1.3	Dependencies . . . . . 14
	7.1.4	Interfaces . . . . . 14
	7.1.5	Resources . . . . . 15
	7.2	Website . . . . . 15
	7.2.1	Context . . . . . 15
	7.2.2	Composition . . . . . 15
	7.2.3	State Dynamics . . . . . 15
	7.2.4	Interactions . . . . . 16
	7.2.5	Interfaces . . . . . 16
	7.3	CLI . . . . . 16
	7.3.1	Context . . . . . 16
	7.3.2	Composition . . . . . 16
	7.3.3	Resources . . . . . 16
	7.3.4	Interfaces . . . . . 16
	7.4	Example Use-cases . . . . . 16
<b>8</b>	<b>System Requirements</b>	<b>18</b>
	8.1	Installation Packages . . . . . 18
	8.1.1	Context . . . . . 18
	8.1.2	Resources . . . . . 18
	8.2	OS-API . . . . . 18
	8.2.1	Logical . . . . . 18
	8.2.2	Interactions . . . . . 18
	8.2.3	Interfaces . . . . . 18
	8.3	Performance Benchmark . . . . . 19
	8.3.1	Context . . . . . 19
	8.3.2	Resources . . . . . 19

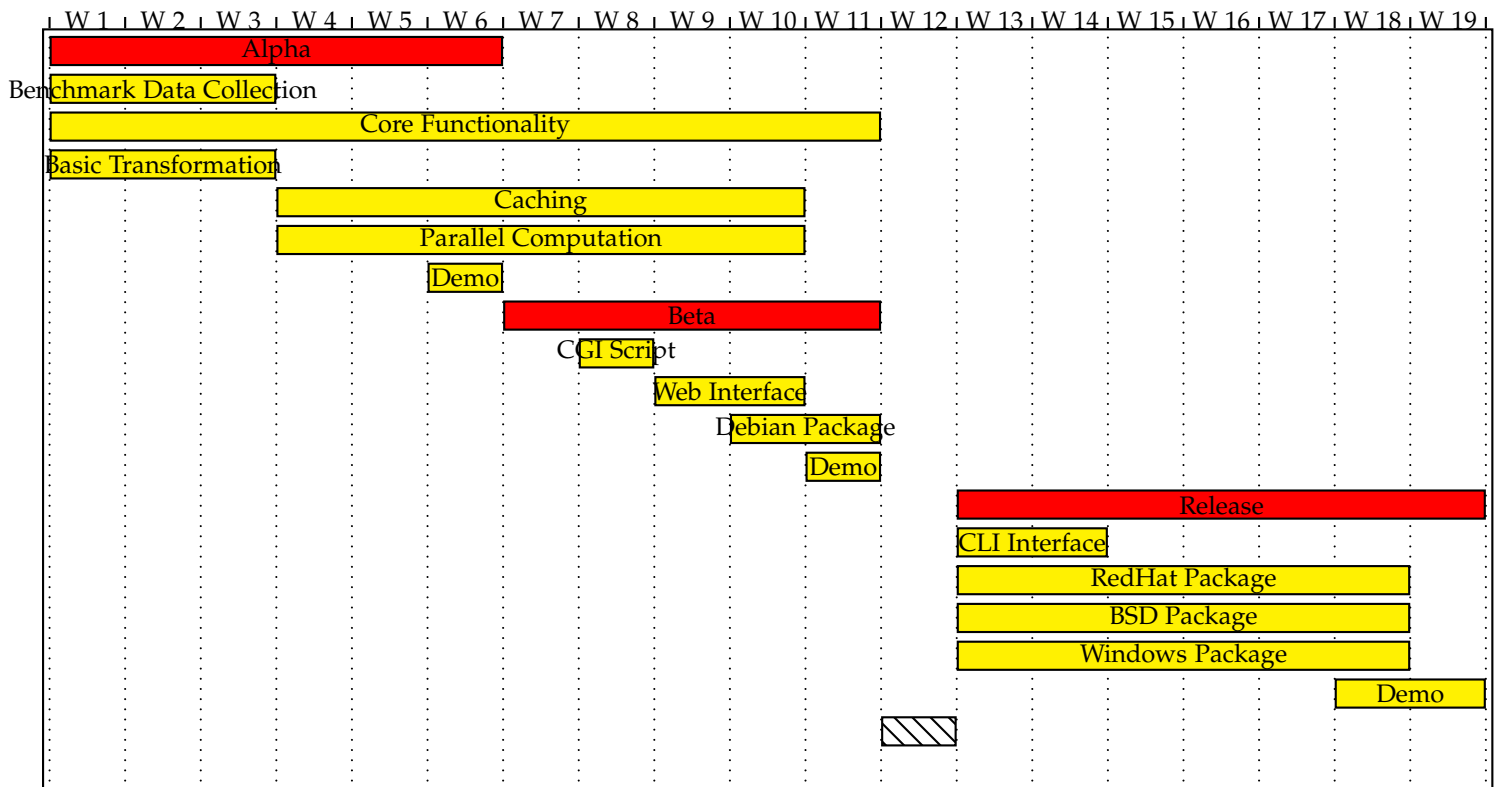


Figure 1: Development Gantt Chart Timeline.

<b>9</b>	<b>Design Rationale</b>	<b>19</b>
9.1	Cache Decisions . . . . .	19
9.2	Web-API Decisions . . . . .	19
9.3	Packaging Decisions . . . . .	20
9.4	Bench-marking decisions . . . . .	20

## 1 INTRODUCTION

### 1.1 Purpose

The purpose of this document is to outline the entirety of the design of the XZES40 application for the purposes of referencing during development and for communication with the project sponsor(s).

### 1.2 Scope

The scope of this document is to outline in necessarily complex terms how the XZES40-Transformer application will be developed.

### 1.3 Development Time-line

The following Gantt chart outlines the projected time-line for development. This starts at the first week of January (winter-term week 1), and goes through project completion in June (end of spring term).

Revision	Date
Working Draft	2016-11-17

Figure 2: Change History Table

## 1.4 Summary

XZES40-Transformer is an application which transforms one XML with one XSLT document for the purposes of data transformation, like that of a spreadsheet program like Excel. The uses for this range widely from business to scientific to personal uses. For most needs, an XML/XSLT document transformer needs to be able to handle a high volume of requests by a wide variety of clients. To address these needs the XZES40 team is building an Open Source XML/XSLT document transformer with key optimizations built in to improve the document transformation process; helping businesses, institutions, and individuals get more done in a day. [2] [5]

## 1.5 Issuing organization

This document has been issued by Oregon State University and the Apache Software Foundation through the OSU 2016-2017 CS Capstone class.

## 1.6 Change history

## 2 REFERENCES

### REFERENCES

- [1] *Apache Xalan*. "http://xalan.apache.org: Apache Software Foundation.
- [2] *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. <https://www.w3.org/TR/2008/REC-xml-20081126/>: W3C Organization.
- [3] *HTTP Server Project*. <https://httpd.apache.org/>.
- [4] *The Document Object Model*. <https://xerces.apache.org/xerces2-j/dom.html>: The Apache Software Foundation.
- [5] *XSL Transformations (XSLT) Version 1.0*. <https://www.w3.org/TR/1999/REC-xslt-19991116>: W3C Organization.

## 3 GLOSSARY

**Apache** An application for sending and receiving HTTP requests on a remote host.. 10, 11, 16

**API** Application Programming Interface. Connects the Apache server to upload or return the files.. 4, 5, 10, 11, 15

**BSD** Berkeley Software Distribution. A popular UNIX operating system.. 13–15, 17

**CentOS** A popular Linux operating system based on Red Hat Enterprise Linux.. 13, 14, 17

**CGI** Common Gateway Interface. A way for a script on a local host to be run remotely via a web-server like Apache.. 10, 11, 16

**CLI** Command Line Interface. A user interface to a computer's operating system.. 10, 12, 13, 16, 17

**Debian** A popular Linux operating system.. 13, 17

- DOM** A cross-platform and language-independent application programming interface that treats XML document as a tree structure.. 4–8
- FOSS** Free and Open Source Software. Software which is developed and maintained for free and by a community.. 17
- FPM** Effing Package Management. A software package creator which targets multiple Unix-like operating systems package managers.. 15, 17
- GUI** Graphical Interface. A user interface to a computer application.. 11
- Hash-map** Hash map based implementation of the Map interface.. 6
- HTTP** Hypertext Transfer Protocol. XZES40-Transformer interact with remote clients.. 10, 11, 13
- Linux** A popular UNIX-like operating system.. 14, 15, 17
- MD5** A complex hashing algorithm used to securely verify data is consistent.. 7
- OS** Operating System. Software which runs other software.. 13, 15
- OS-API** Operating System API. It will be the program's interface with the operating system.. 15
- Python** A program scripting language used for a wide variety of purposes from scientific applications to dynamic websites.. 10, 11, 13, 16
- Struct** A complex data type declaration.. 6
- UI** User Interface. Exposes some functionality, usually referring to one on a computer, to the user through some way of interaction either mouse, keyboard, or combination of the two.. 10
- Unicode** Provides an unique number for every character.. 6
- UNIX** A family of Operating Systems encompassing Linux, BSD, and MacOS.. 13, 17
- URI** Universal Resource Indicator. Also called a url, this is the path a client needs to use to identify a web-api endpoint.. 10
- Web-API** Web API. Connects web application by Apache server to upload or return the files.. 9–12, 16
- Web-UI** Web UI. A user interface which uses a browser to render its content.. 12
- Windows** A very popular non-unix operating system.. 13–15
- WIX** Windows Installer XML. A Windows software packaging program.. 15
- Xalan C++** A library used to XML transformation in C/C++.. 5, 16
- Xerces C++** A library used to perform XML transformation in C/C++.. 6, 16, 17
- XML** Extensible Markup Language. The human-readable data format used and processed by our application.. 3–6, 10–13, 16
- XSLT** Extensible Stylesheet Language. The human readable format used to transform documents in our application.. 3–6, 10–13, 16

## 4 COMPONENT OVERVIEW

The following is a list of **required** project components ordered by importance:

Component	Owner
Document Transformer	Elijah C. Voigt
Website Interface	Shuai Peng
Web API	Elijah C. Voigt
Document Cache	Shuai Peng
Daemon Process	Elijah C. Voigt
Parallel Document Transformation	Zixun Lu
Benchmarking	Zixun Lu
Debian Software Package	Zixun Lu

Figure 3: Required project components ordered by importance

Component	Owner
CLI Interface	Elijah C. Voigt
CentOS Linux software package	Elijah C. Voigt
Windows software package	Shuai Peng
FreeBSD software package	Elijah C. Voigt

Figure 4: Stretch goal project components ordered by importance

## 5 DOCUMENT TRANSFORMER

This section outlines the views and viewpoints related to the Document Transformer. These components relate to core functionality of the application:

- 1) Taking input XML and XSLT documents
- 2) Parsing them into DOM objects
- 3) Transforming those into a new XML document
- 4) Returning the final product to a user

This section does not necessarily outline how users interact with the application, for that one should see the User Interface section (7).

### 5.1 Transformer

As the name suggests, the Transformer is the core of the XZES40 Document Transformer, which is the core of application as a whole. At a high level this component takes two documents, an XML and XSLT document, and returns a transformed XML document. The rest of the application is built around this component; all other parts of the application depend on or work toward this feature.

### 5.1.1 Context

The Transformer provides the core functionality of the application by transforming input documents into output documents.

All users of the application use the Transformer indirectly by way of using it to transform their input documents into output documents. The Transformer component is not be directly exposed, however it is accessible via the Web API (7.1).

### 5.1.2 Composition

The Transformer's functionality is outlined in the following steps:

- 1) Receive the input XML XSLT files.
- 2) Create a Document object out of the input files. This will either encode the input files as DOM objects or fetch the pre-compiled objects from the cache. 5.3 5.2
- 3) Pass the Document's parsed contents to the [1] **transform** method.
- 4) Encode the transformed document back to a text file and return this document.

These steps will each be held in a thread spawned by the long-running daemon process. 6

### 5.1.3 Dependencies

The Transformer directly depends on the following internal application components for the following reasons:

- The Document class (5.2) is used to create a transformable DOM object from an input file. [4]
- The Document Cache (5.3) is used to store and retrieve parsed and transformed documents. While this cache isn't strictly necessary for document transformation, it is used to speed up the process drastically.

### 5.1.4 State Dynamics

The Transformer deals heavily with state dynamics in two ways: transforming documents and storing/retrieving cached documents.

#### 5.1.4.1 Transforming documents

The Transformer does not directly handle transforming documents, this task is delegated to the Xalan C++ library. [1] The transformer takes two or more documents as input: an XML DOM file and an XSLT object. The input documents are transformed by the Xalan C++ library and the resulting document is eventually returned by the writer (??) via the API (7.1).

#### 5.1.4.2 Cached documents

The Transformer also operates with the cache (5.3). Input files are raw files in the cache. XSLT documents are primarily cached as they change very little and are re-used very often.

### 5.1.5 Interactions

Few components directly call the Transformer, however the transformer depends heavily on communicating with the cache. This is done by using a "cache" object. Documents are retrieved by using a "get" method and added to the cache with a "set" method. More information can be found in the Cache section of this document (5.3).

### 5.1.6 Interfaces

The Transformer has one programmer-facing interface, the "transform\_documents( filepaths )" function. This can be used for testing, mocking, or implementation purposes.



#### 5.1.6.1 DOM object transform\_documents( filepaths )

- This function takes as argument the path to the XML file, the path to XSLT document, the output destination for the output file.
- As output it writes a transformed XML and returns a status code.

## 5.2 Document class

The Document class gets as input an XML or XSLT file prepares it to be transformed. It does this by either compiling it into a DOM objects and adding it to the cache or fetching the already compiled object from the cache.

### 5.2.1 Context

The Document class is a major component of the application. It receives as input an XML or a XSLT file, it then parses and store or just retrieves the DOM representation of that file. The user will not use this function directly, but it executed by the Transformer (??).

### 5.2.2 Dependencies

The Document class also depends on the Cache to store and retrieve parsed objects.

### 5.2.3 Interactions

- 1) The class obtains a Unicode formatted XML or XSLT file-path.
- 2) The class checks if the file has been parsed and stored in the Cache, and does not re-parse the file if it is in the Cache.
- 3) The class generates a DOM object via the Xerces C++ library.
- 4) The class stores the DOM object to the Cache.

### 5.2.4 Resources

The Xerces C++ library will be used during the parsing to generate a DOM object from the input file.

### 5.2.5 Interfaces

The following function declaration is used for the parser method:

#### 5.2.5.1 class Document(source\_file\_path)

. This constructor receives an XML file or a XSLT file, and then parses it into a DOM object which is stored in the Cache.

In parsing and storing the object, the class has methods which read the file contents, hashes the contents of the file, and uses this as a key when inserting data into the Cache.

## 5.3 Cache

The Cache is a core feature of the XZES40 application. The Cache speeds up document transformation by storing and retrieving previously parsed documents.

In practice the Cache will operate much the same as a Hash-map does, storing data at a location given a key which can also be used to retrieve the data.

### 5.3.1 Context

The program heavily depends on the Cache. The Cache can store, delete, and retrieve DOM object from the in-memory cache

#### 5.3.1.1 Storing data

is the major element of the Cache.

The Cache stores data from the user, primarily parsed DOM data to avoid re-compiling files.

#### 5.3.1.2 Deleting data

provides the ability to remove items from the Cache for whatever reason.

#### 5.3.1.3 Retrieving data

allows users to fetch information stored in the Cache given the object's key.

### 5.3.2 Composition

Below are few of the components making up the Cache.

- The Cache stores input data in a Struct along with with the last time it was accessed and they key used to access the data.
- The Cache can retrieve the data via searching the associated key.
- The Cache can delete data corresponding with a key.
- The Cache can dump its contents to disk periodically.
- The Cache can load its data form an input file. This along with the previous item is used to recover from a system failure.

### 5.3.3 Logical

XZES40 handles object as a special structure. The Cache saves that object in the following format.

#### 5.3.3.1 Key

This is a hash value associated with a cached object. This value is for retrieving the data from the Cache.

#### 5.3.3.2 Content

The content is the parsed or transformed DOM object.

```
struct node {
    dom data;
    string key;
    date last_used;
}
```

### 5.3.4 Information

If an object is not in the Cache, the Cache stores the DOM object into the in-memory cache. Objects are removed from the cache via the Cache class' delete method.

### 5.3.5 State Dynamics

The Cache handles the following state changes in the following ways.

- When the Cache starts it allocates a block of memory for storage.

- If an item is being set the cache ignores previously existing data. It is the developers duty to ensure important data is not being overwritten.
- When data is being read the state of that item in cache is assumed not to change. If the item is not found it returns an empty object.
- When an item is deleted it returns a SUCCESS status if the object existed and is not deleted, and a FAILURE status if the object did not exist in the cache before the call.

### 5.3.6 Interactions

The parser sends a parsed DOM object to the Cache. The Cache will search the files if it exists in the memory. If the file exists in the memory, the Cache retrieves the DOM object from the Cache and returns to the Document class. If the file does not exist in the memory, the Document class continues to process the file, and stores the parsed file into the Cache.

### 5.3.7 Algorithms

The Cache can be divided into five major functions. The first function is **set**, second is **get**, third is **dump**, fourth is **load**, and fifth is **delete**.

#### 5.3.7.1 Set

stores an object at a location in the allocated memory block based on the MD5 hash of the "key" parameter given.

#### 5.3.7.2 Get

This function retrieves the DOM object from the cache by the key of the parsed file.

- 1) If there is a key exist in the cache, **return** the DOM object at that location.
- 2) If the key does not exist in the cache, **return** an empty storage struct object.

#### 5.3.7.3 Delete

This function deletes the DOM object from the cache by the key of the parsed file.

- 1) The Delete receives a key to delete the data it's located at.
- 2) If there is a key exist in the cache, the Delete deletes this DOM object in the cache, and **return** that there is successfully delete.
- 3) If the key is not in the cache, **return** is no DOM object inside of cache.

### 5.3.8 Resources

The Cache needs a large memory allocation to function optimally.

### 5.3.9 Dependencies

The Cache depends on the Document and the Transformer to populate it. The Document class sends a file to the Cache for checking if the file exists in the Cache. The Document class also can send a parsed DOM object for storing into the Cache if the file does not exist in the Cache. The Cache may either return a DOM object to the Transformer, or the cache returns null if the DOM object is not in the Cache.

### 5.3.10 Interfaces

Below are the major interfaces for the cache component of XZES40-Transformer.

#### 5.3.10.1 void document\_cache( dom\_object file )

This checks the DOM object if it exist in the Cache.

- If it's not in the cache, return fail.
- If it is in the cache, return true.

#### 5.3.10.2 dom\_object store\_cache( dom\_object file )

This receives object file and store it into the memory.

#### 5.3.10.3 dom\_object delete\_cache( dom\_object file )

This deletes data from the Cache.

- If the DOM object exists in the cache, return true.
- If the DOM object does not exist in the Cache, return error.

#### 5.3.10.4 dom\_object read\_cache( dom\_object file )

This retrieves data from the cache.

- If the DOM object exists in the cache, return the DOM object.
- If the DOM object does not exist in the Cache, return error.

## 5.4 Parallel Computation

In addition to the Cache component of XZES40-Transformer (5.3), the application will also carry out certain computations in parallel to further leverage the computing resources available to it and compile documents even faster.

### 5.4.1 Context

The Parallel Computation component of the application carries out the following operations in parallel to speed up documentation transformations.

#### 5.4.1.1 Document Parsing

will be carried out in parallel. These operations are logically independent so they can be carried out simultaneously without affecting data integrity.

#### 5.4.1.2 User Requests

will also be carried out in Parallel, handled by Apache.

### 5.4.2 Dependencies

The Parallel Computation component of the application will be carried out at the high level by Apache delegating parsing jobs, see the Web-API section for more information 7.1. the application will also carry out parallel computation internally (C++) using the MPI library.

### 5.4.3 Interactions

Our MPI-based thread computing will operate mostly autonomously, except when putting data into and fetching data from the application Cache (5.3). With the exception of interacting with the Cache each parsing thread will not interact with other internal components of the application.

### 5.4.4 Algorithms

One major concern with handling a cache by multiple threads and processes in parallel is avoiding data corruption. This is not a problem we have yet solved and further revisions of this document will elaborate on how we will handle this dilemma.

### 5.4.5 Interfaces

When documents are being parsed each parsed document will spawn its own thread. This functionality is in the Document class component of the application in section 5.2.

## 6 DAEMON

### 6.0.1 Context

Because the application needs to be available continuously, and because it needs to store objects in an in-memory Cache, the application is daemonized. This means that it continues to run in a suspended state when it is not actively performing document transformations. When it does receive an incoming request it spawns a thread, does the transformation, and closes the thread. This allows multiple document transformations to happen in parallel conveniently.

### 6.0.2 Composition

The daemon is composed of two main components:

- A parent thread which will wait for a signal to handle a request. This signal is accompanied with some context for what task needs to be performed, i.e., a document transformation.
- A child thread is spawned for each incoming signal. The child process exits at the end of its transformation after returning the transformed document or an error which is propagated up through the parent process to the signaling process.

### 6.0.3 Interfaces

The interface to the daemon is the same as that of the daemon. A CLI on the host running the daemon will accept an XML and XSL document as input, pass these to the daemon, and respond with an error or transformed document.

## 7 USER INTERFACE

The Document Transformer is fine and great, but without a user interface it's not useful. The following two sections, the Web-API (7.1), Website interface (7.2), and CLI (7.3) together outline the ways users will interact with the system described in the previous section.

### 7.1 Web-API

The Web-API is the standardized interface between the user interfaces and an instance of the application running on a host, communicating over HTTP.

#### 7.1.1 Context

The users for the Web-API are those who use the application from the CLI (7.3) and (Website 7.2). Both interfaces interact with the XZES40-Transformer host via standard HTTP request methods. But nobody should use the API directly as a UI is much easier than crafting an HTTP POST request.

### 7.1.2 Composition

The Web-API is composed of the following components:

- 1) An Apache runs on the remote host.[3]
- 2) The server manages a Python CGI script which handles accepting requests and sending responses.
- 3) The Python script calls the XZES40-Transformer application locally, passing input documents from a POST request and sending response files via the CGI interface.

### 7.1.3 Dependencies

The Web-API internally depends on an XZES40-Transformer binary which accepts an input XML file, and input XSLT file, and writes a transformed file to disk in a predictable location. The binary should also exit with predictable exit codes to communicate any errors or successes.

### 7.1.4 Interfaces

The Web-API communicates with remote clients via a standard HTTP API. Below are the requests a client can send and the possible responses:

#### 7.1.4.1 POST /api/

This is a request sent to the API endpoint (/api/) with the intention of getting two input files transformed into a new document. This can respond in the following ways.

**200 OK** Means the transformation was successful. This response includes a body containing the transformed file and a URI to re-download the response file.

**400 USER ERROR** Means that the user sent malformed documents. This can include a document which does not follow the XML/XSLT standard to a document which does not have a readable character encoding. This response includes a body containing an appropriately specific error.

**500 SERVER ERROR** Means that the server experienced an internal error while processing the request. This includes fatal XZES40-Transformer errors. This response includes a body containing an appropriately specific error.

The POST request is a request with a Form containing input documents in fields titled XML and XSLT for an XML 1.0 formatted document and an XSLT 1.0 formatted document respectively.

#### 7.1.4.2 GET /api/

This is a request sent to the API endpoint (/api/) with the intention of getting a status of the server. This can respond in the following ways:

**200 OK** Means the Web-API endpoint is active and functioning correctly.

**404 NOT FOUND** Means that the Web-API endpoint is not configured correctly or the user is accessing a page which is not available.

**503 SERVICE UNAVAILABLE** Means that the Web-API is setup correctly, but the application on the remote host is not operating correctly.

The GET request is an empty GET request to the servers "/api/" endpoint.

#### 7.1.4.3 Python Interfaces

The following interfaces are used for implementing the above HTTP interfaces.

**process\_request(http-request req)**

This method is used to receive an HTTP API request. It does this by reading the request header and deciding based on that to carry out one of the above responses. In the case of a POST request it uses an “exec” call to the local XZES40-Transformer application binary and responds with the error / output file of that application.

### 7.1.5 Resources

The Web-API has the following external dependencies:

- The **Apache** is used to process HTTP requests by running the Python CGI script.
- **Python 2.7+** is the programming language the Web-API is written in.
- **cgi** is a Python library for processing web requests in Python.
- **cgitb** is a Python library for developing with the cgi library.
- **mod\_wsgi** is an Apache module for interacting with Python.
- **mod\_python** is another Apache module for interacting with Python.

They should be installed via a system package manager or the Python package manager (whichever is appropriate) for the Web-API to operate correctly.

## 7.2 Website

The website interface is the major user interface for the XZES40 application. Website interface is a GUI interface for user.

### 7.2.1 Context

The Website interface is common interface for application. User can uses website to complete request with few of instructions. User can upload the XML/XSLT files that they want, and the website will give the feed back.

### 7.2.2 Composition

The web interface to access the application in a browser. Here is components of the Web-UI.

- An upload field for the **XML** file.
- An upload field for the **XSLT** file.
- A button for user send the request to server via **POST** method.
- After user sends a request to server the website will respond with a message to user. This message may be error or successfully upload or generate download link.

### 7.2.3 State Dynamics

There are few of states for website.

- If user upload bad files the website gives feedback that the files were bad.
- If user upload files and server transforming the documents the website gives the feedback that user upload files successfully.
- If user uploads files and server is down the website gives the warning that transformation service is not working.
- If user uploads good files and the server successfully transforms the documents the user is given a download link to the new file.

### 7.2.4 Interactions

The initialization status of website is waiting for user upload files. After user select files that they want upload and click the submit button. Website will send the request to server via the Web-API. There are many possible feedback as following.

- If the files uploaded are bad the website pops up a warning about the malformed files.
- If the user uploads both an XML and XSLT file the user is alerted that the request is good to go and that the website will generate a download link for them.
- If the connection is broken the website will give them a warning that user should check the connection between client and server.

### 7.2.5 Interfaces

The Web interface is XZES40 main interface for user. The web interface asks user to upload XML files and XSLT files, and gives the feedback of result. Here is the diagram for website interface.

## 7.3 CLI

The CLI is used to interact with the system via a text-based terminal/shell interface.

### 7.3.1 Context

The users of this interface are individuals who either prefer the CLI over a web interface or for testing purposes as automating tasks with the CLI is very common.

### 7.3.2 Composition

The CLI, written in Python, and will be composed of the following components.

- The main function parses command-line arguments specified below in the “Interfaces” part of this section.
- A query is built for the server including the XML and XSLT documents to the specified server.
- The query is sent to the server in a POST request.
- When the response is received either an error message is displayed to the user or a file is saved locally.

### 7.3.3 Resources

The CLI depends on the following system dependencies:

- Python 2.7+ is the programming language it will be implemented in, so a Python runtime will be necessary for the application to run.
- Requests is the standard Python library for interacting with servers over HTTP.
- A terminal and UNIX compliant shell will also be necessary to access the application via the CLI.

### 7.3.4 Interfaces

The CLI has the following options available at the command-line:

## 7.4 Example Use-cases

The following are examples of the CLI in use:

“Un-happy path” with bad host.



```
$ xzes40
  --server=<server-url[:port]> # API endpoint and port to be used.
  --xml=<input-file>.xml      # Input XML file
  --xslt=<style-sheet>.xslt   # Input XSLT file
```

Figure 5: CLI Flags. All fields encased in chevrons symbols are required. All fields with square brackets are optional.

```
$ xzes40 --server=http://xzes40.example.com:8080 \
  --xml=input-file.xml \
  --xslt=style.xslt
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Downloading transformed document to ./xzes40-transformer-2016-11-22.xml
```

Figure 6: “Happy path” use-case without specifying return file name.

```
$ xzes40 --server=http://xzes40.example.com:8080 \
  --xml=input-file.xml \
  --xslt=style.xslt \
  --output=transformed-doc.xml
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Downloading transformed document to ./transformed-doc.xml
```

Figure 7: “Happy path” use-case with return file name.

```
$ xzes40 --server=http://xzes40.example.com:8080 \
  --xml=input-file.xml \
  --xslt=style.xslt \
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Server responded with error. One of your documents is malformed.
```

Figure 8: “Un-happy path” with malformed document.

```
$ xzes40 --server=http://fakesite.com:8080 \
  --xml=input-file.xml --xslt=style.xslt \
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Unable to reach server. Is the host/port correct?
```

Figure 9: “Un-happy path” with bad host.

```
$ xzes40 --xml=input-file.xml--xslt=style.xslt
Please specify a host
```

Figure 10: “Un-happy path” with no host.

```
$ xzes40 --server=http://xzes40.example.com:8080
Please specify an input xml and/or xslt document.
```

Figure 11: “Un-happy path” with missing input document.

## 8 SYSTEM REQUIREMENTS

The XZES40-Transformer will be required to work on the Debian Linux system. Once development on Debian is completed the application will be ported to other OS platforms. Our team will release Debian, CentOS, BSD and Windows packages.

### 8.1 Installation Packages

For installation convenience the XZES40 project will provide a Linux, BSD, and Windows installation packages.

#### 8.1.1 Context

The user can directly download installation packages through the internet and install the XZES40-Transformer on their local systems. We will create the Debian package initially. After this, we will create the CentOS, BSD, and Windows packages. We will upload those packages to the internet and the users can directly download the packages in their operation system.

#### 8.1.2 Resources

The following tools will be used to create the installation packages.

### 8.2 OS-API

The OS-API will be the programs interface with the OS. It will be created to make porting the application easier.

#### 8.2.1 Logical

Any operating system specific operation will be wrapped by the API.

#### 8.2.2 Interactions

The application will interact with the host system via an OS-API. This means that all operating system specific operations (e.g., read, write, seek, etc) will be done via an API. When the application is compiled on a new target platform (e.g., Linux, BSD, Windows) a new platform API must be created for compatibility.

#### 8.2.3 Interfaces

The OS-API interface is for the developer. This interface performs operations by asking the OS-API to carry out the task and that request is translated to the system-specific system-call.

Packages	Tools	Description
Linux & BSD Packages	<ul style="list-style-type: none"> <li>• FPM</li> </ul>	<ul style="list-style-type: none"> <li>• Translates packages from one format to another</li> <li>• Allows re-use of other system's packages</li> </ul>
Windows Packages	<ul style="list-style-type: none"> <li>• WIX</li> </ul>	<ul style="list-style-type: none"> <li>• It is a open source project.</li> <li>• It is more stable and security than other tools.</li> <li>• It has steed</li> </ul>

Figure 12: Installation Packages Resources

### 8.3 Performance Benchmark

#### 8.3.1 Context

We will run test against similar application to determine how fast XZES40 should be. Throughout development we will put our application through the same paces and compare which is faster.

#### 8.3.2 Resources

We will be bench-marking these applications specifically for the following reasons.

## 9 DESIGN RATIONALE

This section explains why certain design decisions were made and “connects the pieces” of the application.

### 9.1 Cache Decisions

The reason the Cache uses an in-memory system rather than a database or a file-based cache is purely for performance reasons. Using an in-memory cache over a file-based one should yeild faster performance. As a compromise the cache is dumped to a file periodically to save the state of the cache in case the application daemon (which holds the cache) is restarted.

This may cause problems as the service may run out of memory. In an attempt to mitigate this a Garbage Collector may be built which is triggered when a certain percentage of the application's allocated memory is used, or on certain time intervals.

### 9.2 Web-API Decisions

The Web-API connects the application to the outside world in an ideally simple to implement and quick to operate fashion. The application uses Apache to handle incoming requests, these are passed to a CGI script which could be written in anything, we chose Python because it is easy to write, maintain, and is well supported.

Technology	Description
Xalan C++ CLI	<ul style="list-style-type: none"> <li>• Xalan C++ uses Xerces C++ to parse XML documents and XSLT.</li> <li>• The project provides an open source CLI program to test the project libraries.</li> <li>• Free and Open Source</li> </ul>
Altova	<ul style="list-style-type: none"> <li>• To meet industry demands for an ultra-fast processor.</li> <li>• It offers powerful, flexible options for developers including cml, python.</li> <li>• Superior error reporting capabilities include reporting of multiple errors, detailed error descriptions.</li> </ul>

Figure 13: Bench-marking Resources

Python is called by Apache which then in turn, based on the request, either returns an application status or calls the XZES40 application locally. In this way Python with Apache is a simple, well supported set of tools which expose the application to the outside world.

These tools were not necessarily chosen for their speed, and so a redesign may be necessary if they create a bottleneck in the request pipeline.

### 9.3 Packaging Decisions

This document outlines the use of FPM to create its Linux and UNIX packages. This decision was made for convenience and to allow for quick iteration on the package. In doing research there *are* other tools which can be used to create packages on CentOS, Debian, and BSD but they are very system specific and so would make iterating on the package very difficult, and updates to the software a pain to package. Using FPM we can even automate the build of packages on our UNIX-like systems.

### 9.4 Bench-marking decisions

To demonstrate the competitiveness of the XZES40-Transformer application we chose to compare the performance of this application against an FOSS tool, the Xerces C++ CLI, and a Closed Source tool, the Altova application.

This feels like a fair and balanced comparison as we hope to be competitive with Closed Source alternatives but need to ensure we are at least better than a close FOSS alternative.

Elijah C. Voigt

Zixun Lu

Shuai Peng

\_\_\_\_\_  
*Signature*

\_\_\_\_\_  
*Signature*

\_\_\_\_\_  
*Signature*

\_\_\_\_\_  
*Date*

Steven Hathaway

\_\_\_\_\_  
*Date*

\_\_\_\_\_  
*Date*

\_\_\_\_\_  
*Signature*

\_\_\_\_\_  
*Date*