ANNEX I
Technical description of the Europass Binder

INTRODUCTION AND OVERALL DESCRIPTION

According to the Europass Decision, "*Citizens will be entitled:*

– *to complete through the Internet their Europass-CV and any other Europass document which does not need to be issued by authorised bodies,*

– *to establish, update and remove links between their Europass-CV and their other Europass documents,*

– *to attach any other supporting documents to their Europass documents,*

– *to print totally or partially their Europass and its annexes, if any."*

Still some development is needed to implement the notion of Europass being a single framework and acting as a Transparency integrator.

The Europass Binder will be the new edition of the Europass on-line CV/LP editor, being advanced in terms of usability, quality, graphics, speed, responsiveness and intuitiveness. It will:

– Provide a common, single screen to perceive all of Europass instruments (cv, lp, em, ds, cs) as bound together;

– reuse any data that are common amongst the instruments (e.g. First/Last name, etc.);

– provide the possibility to preview and download the Europass Binder en bloc (e.g. in a single PDF file) or separately.

This will result in one reusable PDF + XML Europass document, combining both a graphical layout (PDF) and database capabilities (XML).

The prototype of the Europass binder is available at http://europassd.cedefop.europa.eu/binder/index.jsp. The Feasibility study you can find in the Annex 1: "Europass binder study", below.

Europass Binder Study

Table of Contents

1. WHAT IS THE EUROPASS BINDER

1.1. Functional Requirements for the implementation of a Europass Binder

1.1.1. The File Format

The idea behind the **Europass Binder** instrument is to allow the citizens to group, store, organize, link and finally print all their Europass documents. Initially, the plan was to study the possibility to extend the Europass Namespace so that it can contain all the 5 Europass documents and to propose an XML based implementation of the Europass Binder.

This initial idea was abandoned during this study and after some discussions, for the following reasons:

- The Europass documents are **not** all **produced** and managed by the citizens: actually only the ELP and the ECV are. The citizens rely on organisations to give them some of the documents, which apart from the National Europass Centres cannot be forced to produce appropriate structured documents.
- At the time of this study, **not** all Europass documents are **structured**: the Europass Diploma Supplement and Certificate Supplement are not structured and even their printable form may vary a lot from country to country. A specification for these documents would be needed along with its wide acceptance from the organisations producing these documents …
- Some Europass documents such as the Europass Mobility Experience wear **digital signatures** and cannot be altered by the citizen. Thus, there are little to no benefits in having a unified namespace for all documents and the possibility to share information between them.

A common way to group and organize heterogeneous and non structured files is to use an archive file format such as ZIP, RAR or anything else. The requirements of the file format used are the following:

- It should allow **compression** in order to make the archive more portable
- It should support **tree-like structures** based on folders and files
- It should support **any type of files** or at least: PDF, XML, images, Microsoft Word and ODT.

The requirements for the file containing the table of contents are the following:

- It should contain **entries** corresponding to the **folders** of the archive
- It should contain **entries** corresponding to the **files** of the archives
- Each entry should at least bear a **name**, a **mime-type**, a **size** and a **multilingual label**
- File entries and folder entries should be **nested**

1.1.2. The Management Tool

The second main part of this study concerns the implementation of an instrument (a management tool) allowing to manage a Europass Binder. The functional requirements for this tool are the following:

- Be able to create a new binder or upload an existing one

- Manage the documents of contained in this binder: add, remove, update or rename documents in the various sections of the Binder
- Link the various documents, for example link a Mobility Experience document to a Work Experience in the CV
- Configure a printout: make a selection of documents to be added to the printout, order them and specify how they should be added to the final document (inline or as attachments).
- Produce the final printout
- Integrate existing Editors (at least for the CV and the ELP) in order to allow creation and modification of the document directly from the Binder Interface
- Integrate existing Web Services (at least for the CV and ELP) in order to allow direct conversion of XML files in PDF.

Apart from these functional requirements concerning the Europass Binder itself, there are also some requirements common to all Europass instruments such as: full support for multilingual interface, HR-XML integration, bridges with other systems such as Job portals …

1.2. Technical requirements and constraints for the implementation of a Europass Binder

1.2.1. The file format

The solution that we propose in this study relies on the usage of an archive file with a predefined tree-like structure and a table of contents pointing to the different documents contained in the archive and allowing to give them multilingual human readable names. This idea is commonly used in several existing file formats such as ODT, JAR or WAR files.

The ZIP file format is a natural candidate for this kind of needs as it supports all the needed functionalities and is very widely used. The main idea here is to pre-create a folder for each type of document that can be added to the Europass Binder (covers, CV, ELP, DS and CS) and to add a specific file at the root of the archive containing the table of contents. This table of contents can be written in XML according to a very simple specification (like a DTD). The complete specification of this file is given in the next following paragraphs of this study. If needed, the extension of this file may even be changed from ".zip" to something else such as ".eb" for example.

1.2.2. The Management Tool

In this study we will propose a web interface for the Europass Binder management tool mainly because all the Europass Instruments are web based and also because a web interface was asked for this study. However, we could easily imagine a standalone application for the Europass Binder management tool with internet connectivity in case Web Services or Web Resources are needed.

The technical requirements for this tool are what is commonly understood by "Web 2.0" application: fast, responsive, without browser reloads, AJAX based (Asynchronous Javascript And XML) …

Apart from these "classical" requirements, some of the functional requirements presented in the previous paragraphs have non-trivial technical implications:

- For example, the integration of the existing or shortly coming offline Europass Editors have implications on the editors themselves, which cannot be considered as a part of this study.
- The integration of Web Services at least for the Europass CV and ELP (also for other documents if they existed) implies the usage of a SOAP implementation on server side. This could make the overall tool much heavier but also much more powerful.

In the following paragraphs of this study, we will give a detailed description of the application's architecture as well as the tools and libraries used to achieve the requested result.

2. TECHNOLOGICAL AND TECHNICAL BACKGROUND

2.1. The file format

The **Europass Binder** as we see it in this study is first of all a file format allowing to store and organize all the Europass documents of a citizen. The file format can be described in 2 parts: the first concerns the ZIP file and its structure and the second concerns the table of contents.

Along with this study we provide an example binder file: *binder.zip* that can be used with the prototype application once it is installed.

2.1.1. The ZIP file structure

The Binder archive file **must have** the following structure:

```
binder.zip
      |_ Covers
      |     |_ EuropassCoverPage_Blue_en_GB.pdf
      |     |_ EuropassCoverPage_White_en_GB.pdf
      |     |_ …
      |_ Curriculum Vitae
      |     |_ CVTemplate_en_GB.doc
      |     |_ CV-82670.pdf
      |     |_ …
      |_ Language Passport
      |     |_ ELPTemplate_en_GB.doc
      |     |_ LP82680.pdf
      |     |_ …
      |_ Mobility Experience
      |     |_ MobExamples_en_GB.pdf
```

```
|     |_ …
|_ Certificate Supplement
|     |_ CSExamples_el_GR.PDF
|     |_ …
|_ Diploma Supplement
|     |_ DSupplementExamples_fr_FR.pdf
|     |_ …
|_ toc.xml
```

Please note that the files in *italics* are given as examples.

As we can see from the previous diagram, an empty binder archive file is composed of:

- 5 folders corresponding to the 5 Europass instruments and **named after them**. Each of these folders **should** only contain documents corresponding to the concerned Europass Instruments
- 1 folder named "**Covers**" which **should** only contain the 2 Europass Covers.
- 1 file named "**toc.xml**" containing the tables of contents formatted in XML.

2.1.2. The Table Of Contents file format

The specification of the table of contents file format is proposed in the form of the following DTD:

```
<!DOCTYPE toc [
    <!ELEMENT toc (folder+,file*)>
    <!ATTLIST toc id ID #FIXED "toc">
    <!ELEMENT folder (label+, file*)>
    <!ATTLIST folder id ID #REQUIRED>
    <!ATTLIST folder name CDATA "">
    <!ELEMENT file (label+)>
    <!ATTLIST file id ID #REQUIRED>
    <!ATTLIST file name CDATA "">
    <!ATTLIST file size CDATA "">
    <!ATTLIST file content-type CDATA "">
    <!ELEMENT label (#PCDATA)>
    <!ATTLIST label lang CDATA "en">
]>
```

This DTD specifies that the table of contents XML file (toc.xml):

- **must** contain a root element named “toc”
- this root element **must** have unique attribute named “id” with a fixed value “toc”
- under this root element:
    - there **must** be at least one element named “folder”
    - and there **may** be one or more elements named “file”
- the “folder” element **must** have an attribute named “id”
- the “folder” element **must** have an attribute named “name” containing the folder name

- the “file” element **must** have on attribute named “id”
- the “file” element **must** have an attribute named “name” containing the file name
- the “file” element **may** have a “size” attribute containing the size of the file in bytes
- the “file” element **may** have a “content-type” attribute containing the mime type of the file

- the “file” and “folder” element **must** contain at least on element named “label”, which contains a multilingual label for the corresponding file or folder used for display purposes in the management tool
- the “label” element **must** have an attribute named “lang” with a default value “en”, which marks the language version of the label

This clearly represents the typical tree-like structure corresponding to the contents of the binder ZIP archive.

2.2. The Web application

The prototype web application implemented with this study is in a certain way a classical Java + JSP application, which can be deployed on any Tomcat server. It is contained in a folder named “Binder Application” along with instructions allowing to install it in a file named “Instructions.doc”.

The management tool is a “Web 2.0” application using AJAX technologies for the communication with the server side code. It is mainly composed of one JSP file, which uses the Ext JS library in order to dynamically build the interface. The AJAX calls needed in order to communicate with the server are implemented using the DWR library. These two libraries are described in greater detail in the following sections of the study.

2.2.1. Overall Architecture

Several solutions were investigated in order to implement a completely offline Web 2.0 application, which would use the internet only for Web Services and / or Web Resources. Unfortunately, the available technologies allowing to store data on the browser are either not mature or not available on all browsers and platforms. For example:

- Google Gears is a browser plug-in allowing to store application data into a local relational database accessible in Javascript from the Web pages. Unfortunately,

Google Gears is only in beta version at the moment and cannot be used for a production system.

- Adobe Air is a runtime allowing to develop Flash applications and to integrate them with server-side components and code (through Flex for exmaple). It has a very rich feature-set allowing to implement Rich Internet Applications (RIAs). Unfortunately, the Adobe Air runtime is not available in a stable version for the Linux platforms.
-

Apart from not being mature yet, these technologies also imply the usage and the installation of external runtimes or plug-ins. For very widespread applications such as the Europass instruments, we think that it is preferable to stick to more standard solutions.

Our implementation for the Europass Binder prototype relies on Javascript and AJAX technologies on the client side and use the server-side for the temporary storage of the user data. The server-side of this application is session based, which means that it uses a web session scope to store some of the user data.

However, because of the nature of the documents present in a Europass Binder (heterogeneous and mostly unstructured) we have chosen to rely on the file system of the server to temporary store the user data.

The following diagram presents the different Europass Documents, their possible file types and their structured nature:

CV	XML, PDF, PDF+XML, Microsoft Doc or ODT	Maybe structured
ELP	XML, PDF, PDF+XML, Microsoft Doc or ODT	Maybe structured
EM	PDF + XML	Structured
DS	could be anything ...	Not structured
CS	could be anything ...	Not structured
Covers	PDF	Not structured

2.2.2. Europass Binder Creation

When a Europass binder is created or uploaded, a folder is created and named with a unique identifier:

- In the case of a new Binder the directory structure and the toc.xml file are created
- In the case of an uploaded Binder the contents of the archive are “unzipped” in the created folder

The only information that is kept in the user session is the unique name of the folder where the files are stored. From now on, all user’s actions will result into changes in the file system, for example:

- New Europass documents are uploaded on the file system
- Document removals result in files being deleted from the file system

- The contents of the "toc.xml" files are always kept synchronized with the contents of the file system

2.2.3. Europass Binder printout

When a Europass Binder document is printed, the user is asked to specify the content disposition of each document in the print out. This content disposition is used in the PDF generation in order to integrate the different Europass documents into the final binder printout. These documents:

- may be integrated inline or as attachments in case of PDF or images
- or must be integrated as attachments in case of any other file type

The following diagram presents the different Europass Document file types and how they can be integrated in the final printout:

Europass XML	Inline with Web Service or attachment
Europass PDF+XML	Inline or attachment
PDF	Inline or attachment
Image	Inline or attachment
Other	Attachment

When the user session expires (usually due to a timeout) the folder containing all the user data is erased.

3. TECHNOLOGIES AND TOOLS USED

3.1. Ext JS

"*Ext JS is a cross-browser JavaScript library for building rich internet applications.*"
From http://www.extjs.com

Ext JS is a Javascript toolkit allowing to write advanced web applications using the fool power of Javascript on modern browsers. It contains various widgets such as trees, grids, forms and form fields and a rich component model allowing to create custom widgets by extending existing ones. It is fully AJAX (Asynchronous Javascript And XML) compatible and offers various possibilities of integration with server-side technologies.

3.2. Direct Web Remoting (DWR)

"*DWR is a Java library that enables Java on the server and JavaScript in a browser to interact and call each other as simply as possible.*
DWR is Easy Ajax for Java"
From http://directwebremoting.org/

DWR is a server-side framework allowing to remote Java Code in Javascript with as little work as declaring a Java class and its remoted methods in an XML configuration files. Java classes and methods remoted with DWR are accessible through a similar Javascript programming interface without the need of writing any code. DWR is also highly customizable and extensible thanks to pluggable implementation that can be injected in the framework through its configuration.

3.3. Integration

The quality of the implementation and the design of these 2 libraries has made it easy for us to implement the necessary "glue code" to allow their optimal integration. For example, we have implemented a DWR Proxy allowing to automatically remote Ext JS Data Grids with DWR and a DWR Tree Loader allowing to automatically remove Ext JS Trees with DWR.

4. PROTOTYPE

4.1. Functionalities covered

The prototype delivered with this study covers the following functionalities:

- Creation of a Europass Binder
- Upload of an existing Europass Binder
- Management of a Europass Binder, in each section:
  - Upload a new document
  - Update an existing document (replace)
  - Rename a document
  - Delete a document
  - View a document
- Configure a printout
  - Add a document in the printout (drag from the tree, drop in the grid)
  - Remove a document form the printout
  - Reorder documents in the printout (drag and drop in the grid)
  - Configure document disposition (inline or attachment)
- Print a binder in PDF

4.2. Remaining functionalities

Some functionality was not implemented for this prototype:

- Multilingualism: the infrastructure is ready, all the interface labels are stored into variables and translations may be applied quite easily. The table of contents supports multilingual labels for all files and folders.
- Links: there was no time to implement this functionality during this study mainly because it involves changes in the generation procedure of the documents themselves.
- Document Editors: the infrastructure is ready for the CV and the ELP. The integration cannot be completed with the existing online tools.
- Integration with generation web services: there wasn't time to implement the integration of the Europass CV and ELP Web Service for this prototype. We think that a pluggable mechanism should be implemented in order to allow the possible usage of Web Services to retrieve Europass Documents produced by external organizations based on an unique document id (and a username and password): a good example of this could be the Europass Mobility System where a web service could generate and serve Europass Mobility Experience documents based on the experience code and the username and password of the holder.