Inventory of Open Source Software in use at European Public Services

September 2021

TRASYS INTERNATIONAL

Study Authors

This study was carried out for the European Commission in 2021 by Trasys Consulting

Contract: Specific Contract n°479 using Framework Contract DI/07624 - ABC IV Lot 3

Request: Funded by ISA² 2020 Sharing and Re-Use Action (2016.31), Open Source Solutions for European Public Services

European Commission

The study was managed by Saranjit Arora (external PM & member of OSPO) and Miguel Diez-Blanco (Commission PM & OSPO Lead) from DIGIT B.3.002. It was reviewed by Gijs Hillenius (OSPO), Evangelos Tsavalopoulos (Head of Sector) and others within DIGIT B.3.

Disclaimer

EUROPEAN COMMISSION

Directorate-General for Informatics
Directorate DIGIT
Unit B3 — DIGIT.B3 Reusable Solutions

E-mail: DIGIT-OSPO@ec.europa.eu

TABLE OF CONTENTS

TABLE OF FIGURES

1. DELIVERABLE OVERVIEW

This deliverable describes the updated process applied to define and develop the inventory of Open Source Software (OSS) in use in the contacted European public organisations, and the resulting software inventory.

The purpose of creating the inventory was to facilitate the identification of **critical software**, software which is in danger of becoming unsupported, thus proving to be a risk to the organisations that use it, often unknowingly.

The scope of the inventory has been determined by the availability of data sources identified in the initial phase of the project. Due to limited data collection, at this stage this inventory can only serve as a partial inventory.

This document describes the various steps through which the inventory has been created, from the treatment of the input data, to the detailed analysis of software information, including the software rating. The various steps of the development of the inventory have been described under a business criticality perspective linked to the use of the inventoried software within the European public organisations, and under a sustainability perspective, determined according to the metrics developed in the context of the EU-FOSSA Pilot Project.

Finally, the present document analyses the inventory results, providing an overview, elaborated on the raw inventory data, of the weight, distribution, and typology of the inventoried Open Source Software. Furthermore, it also analyses the OSS from the point of view of its sustainability, with particular focus on the defined critical software shortlist.

2. SOFTWARE INVENTORY PROCESS

The process to build the software inventory is composed of five distinct phases:

1. Treatment of input data
2. Open Source Software screening and grouping
3. Evaluation of software through criticality criteria and definition of the critical Open Source Software shortlist
4. Evaluation of the critical software through sustainability criteria
5. Identification of dependencies for critical software items

The above listed five phases are explained in details in the following sections.

2.1. Treatment of input data

The input for the inventory process were exported files coming from the various data sources, provided in .csv format and or .xlsx format from Microsoft Excel.

A format template was provided to the identified and contacted stakeholders in order to ensure coherence and usability of the received data. However, some stakeholders didn't use the provided template and, in some cases, the files received have been manually manipulated in order to be usable and to be loaded for Extract, Transform and Load jobs (ETL jobs)

The input files received in .csv format were transformed into .xlsx format in order to ensure uniformity in the format loaded for the ETL jobs. Consequently, the files were transformed in .xlsx with the format "software name; number of users", with the first row containing only the relevant field names.

Once the data received is in the inventory database, the software list is sorted by descending order of occurrences.

2.2. Open Source Software screening and grouping

The full list of software, resulting from the previous phases, has been manually screened in order to identify the Open Source items. The screening has been based on the operator's knowledge and on Google searches.

Furthermore, raw inventory data have been elaborated to group all items pertinent to a certain software package and to eliminate multiple items referring to the same application in order to provide a picture where the use of major software packages can be clearly identified. More specifically:

- Entries such as the ones listed below have been grouped as "Firefox", with the relevant number of instances summed, as they correspond to different installations.
    - FIREFOXPORTABLE
    - Mozilla Firefox ESR, Portable Edition
    - Mozilla Firefox, Portable Edition {Beta}
    - Mozilla Firefox, Portable Edition

    - FirefoxDeveloperEdition
    - Firefox
    - Firefox Nightly
    - Mozilla Firefox Developer Edition, Portable
    - Mozilla Firefox, Portable Edition 2nd Profile
    - Firefox (Mac)
    - Firefox Developer Edition

- Entries such as the ones below have not been counted towards the sum above as they correspond to the same installation. In other words, they are considered as extensions to the core software and thus their number of instances is ignored.
    - Plugin Hang UI for Firefox
    - Firefox Helper
    - Firefox Software Updater
    - Plugin Container for Firefox

The elaboration has been done manually to obtain a shortlist of software items by instances for each software type.

More details are provided in the following Annexes, as part of the present deliverable:

- "Annex_1_OSS_categorisation (only those above 20 instances).xlsx"
- "Annex_2_Complete list of software (including those below 20 instances).xlsx"
- "Annex_3_Open Source Software by system type.xlsx"
- "Annex_4_Top Open Source Software by software type.xlsx"
- "Annex_5_Top Open Source Software dependencies.xlsx"
- "Annex_6_OSS shortlist ranked by Business Criticality Index.xlsx"

2.3. Evaluation of software through criticality criteria and definition of the critical Open Source Software shortlist

This stage includes the data management tasks, which run all in parallel in order to produce a well-structured database. It is worth to mention that the experience and the knowledge acquired during the execution and performance of the previous project are of a great advantage for the project team.

The detailed steps are:

- **ETL procedure**
  After the design of the target data model to define the structure under which the inventory will be stored, a routine is developed to process the raw data and import them to the initial inventory database.
- **Identification of OSS**
  This identification step only includes the software with significant instances (e.g. >=20).
- **Grouping of OSS**
  This step is based on the previous categorisation task (i.e. FirefoxA, FirefoxB ➔ Firefox). A process, including manual visual checks, to group the uncategorised software correctly, has been developed and followed.

- **Enrichment data**
  The data have been enriched with known attributes (i.e. relevant to security flag, user interface) from previous work done.
- **Analysis of the Open Source inventory and implementation of criticalities**
  The latter step includes the rating of the software based on the number of instances, the security level and the existence or not of a user interface.

In more detail, the three criticality criteria applied to assess the OSS are:

1. **Number of instances**

   Rationale: The more a software is deployed, the more it impacts the infrastructure and/or the user base, and the more damage a vulnerability could cause.

   Rating: Normalisation on a scale of 0 to 1, based on the most common software (max instances).

2. **Exposure to users**

   Rationale: A vulnerability in a component exposed to the end user (i.e. that offers an interface to the end users) increases the risk of an exploit attacking the software. This criteria only applies to data centre infrastructure, since workstation users have a direct login to their machines.

   Rating: A binary rating (exposed to users = 1, non-exposed to users = 0).

3. **Relation with security**

   Rationale: A vulnerability in components related to security aspect may increase the damage due to an exploit. Examples of security-related software are the solutions meant to secure communication, to manage authentication, to manage processes and permissions, etc.

   Rating: A binary rating (security-related = 0.5, not security-related = 0).

The total score provided by the sum of the three above scores is then normalised on a scale ranging from 0 to 1 (dividing by 2.5, i.e. the sum of the highest values of the three criteria).

In that way, the "Business Criticality Index" is created and applied. "Annex_6_OSS shortlist ranked by Business Criticality Index.xlsx" shows the calculation of the Business Criticality Index of the top critical items.

It is worth to mention that, although a lot of manual work is needed, especially in the cases of new software, the data management stage should be as automated and parametric as possible.

A mechanism that will detect and explore only the new software and will also function as a first stage of control can also be useful for the respective future projects.

With regard to the "number of instances", the latter is calculated based on the number of deployments. As a result of the above evaluation, Open Source items, with most occurrences, possibly security related and possibly facing a large user base, are considered as critical.

However, an additional manual input has been included in order to increase the ranking of some of the software. The choice of this approach has been made for the following reasons:

- The software is massively used in public institution,

- The software is leader for communities,
- The software is leader for linux, or,
- The software is considered as best integration tool.

Further to the evaluation of software through criticality criteria, it has also been decided to include in the top 30 list, the software that should be included in the latter based on previous experience and knowledge of the software themselves. In addition, for the assessment of the top 30 software, a file received by one of the stakeholder, has been excluded from the list given that the number of instances provided in that specific file was overpowering all the other received files.

2.4. Evaluation of the critical software through sustainability criteria

The shortlisted items resulting from the analysis and judgmental approach described in paragraph 2.3 have been submitted to manual investigation in order to be rated against the sustainability metrics.

Such metrics have been calculated automatically through an Excel file, namely, "D05.02 Assessment of 30 items in the inventoried software against the criticality mechanism", in which, once the parameters used to define the metrics are entered, the latter are automatically calculated according to the provided methodology.

The parameters used for the calculation of sustainability metrics have been looked for in Openhub.net, Wikipedia.org, Github.com, in the specific community websites of each software items, in their relevant documentation and in their related bug trackers.

Due to the length of the full names of the metrics, their IDs (M1 to M34) have been used instead in the database.

2.5. Identification of dependencies for critical software items

Within the critical software shortlist, the items have undergone a dependency identification process. The information of the dependencies can only be found in Linux systems. In Linux, when a software package is installed, the package manager resolves a list of dependencies that are downloaded and installed on the fly, whereas on Windows, the installed software packages contain all their dependencies.

Finding the dependencies of a Linux package can be done, on any RedHat-like machine, using this command:

```
yum deplist httpd | grep provider
```

where "httpd" is the name of the package to get dependencies from.

3. CREATION OF PENTAHO DASHBOARDS AND EXTRACTION OF INVENTORY REPORTS

Inventory data can be navigated and analysed through Pentaho dashboards. These are collections of other content components displayed together with the goal of providing a centralised view of Key Performance Indicators (KPIs) and other business data movements.

This section gives an overview of the structure of the Pentaho dashboarding features, to help DIGIT manage and edit the dashboards.

Credentials to connect to the Pentaho User Console (on the docker host, the url would be http://localhost:8080/pentaho) are:
Login = **admin**
Password = **password**

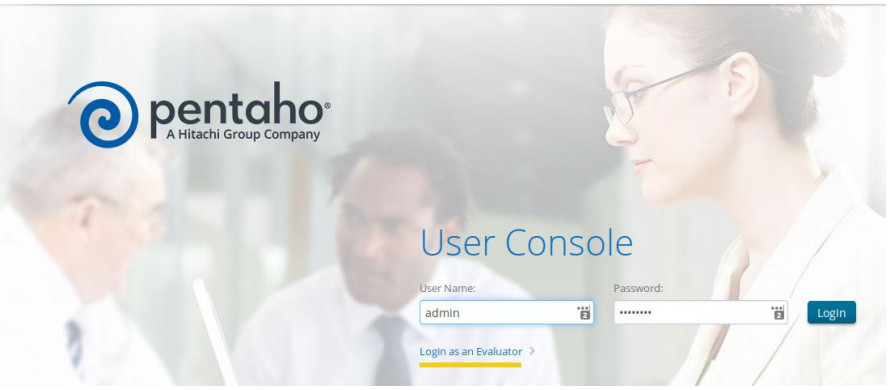


Figure 1: Pentaho User Console

To edit an existing dashboard, click on "Browse files" and go to the folder "Public" ➔ OSSEPS:

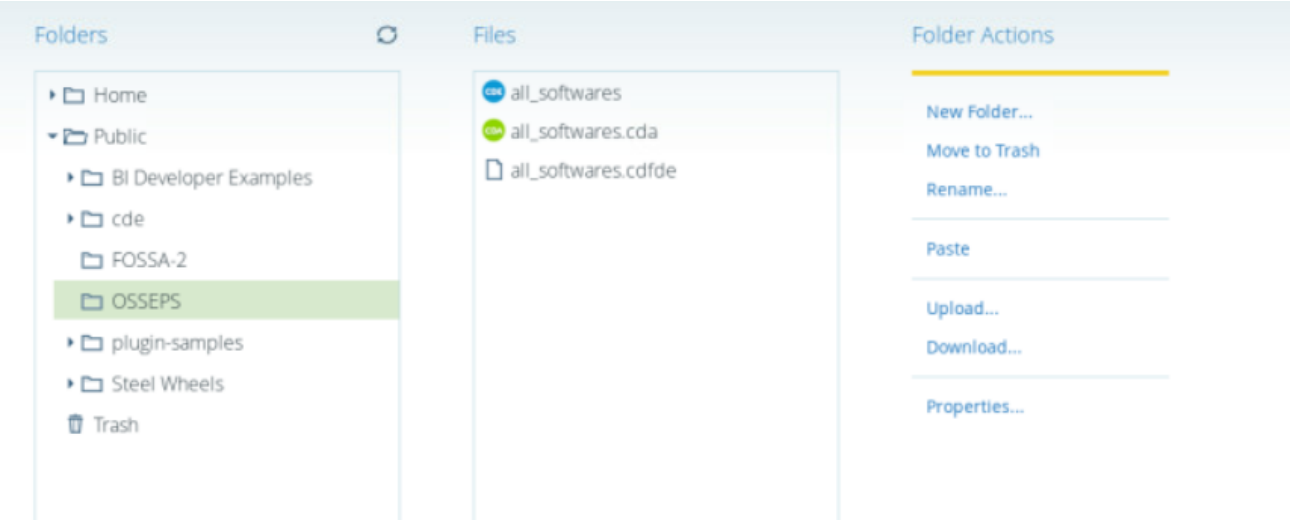


Figure 2: Pentaho files

Select the Community Dashboard Editor (CDE) file from the dashboard to edit, and click "edit" in the right panel. This opens the CDE. The editor is split in 3 tabs (on the right hand): Layout, Components, and Datasources.

Figure 3: Community Dashboard Editor (CDE)

The "Datasources" tab appears as follows:

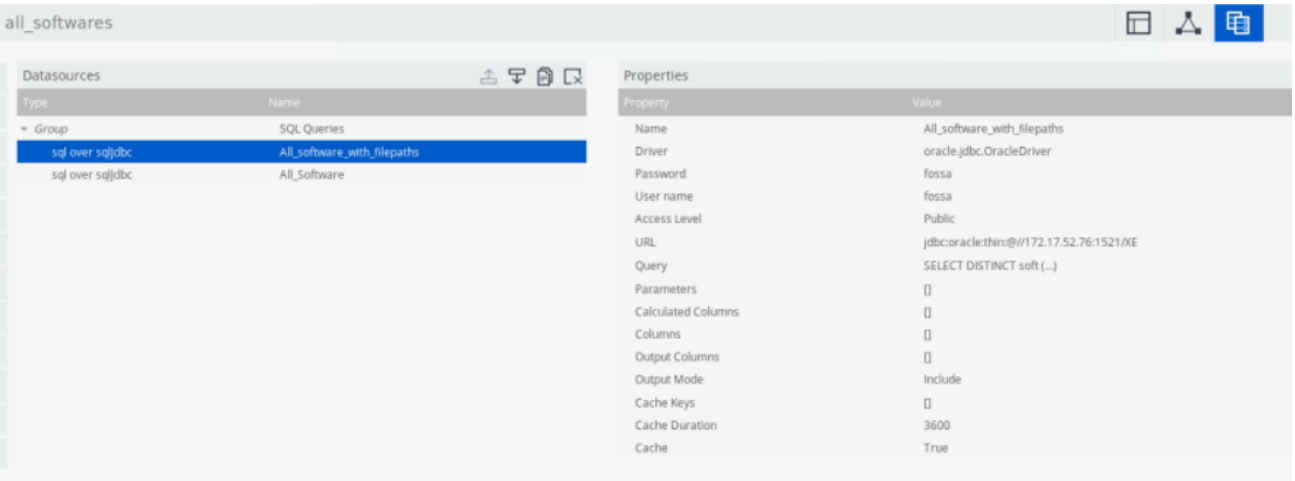


Figure 4: Pentaho CDE "Datasources" tab

The settings to access the (Oracle) database are shown in the capture above. The "Query" setting handles the SQL request to access the data.

Select the CDA file from the dashboard to edit and click "edit" in the right panel. Then click on "Preview".

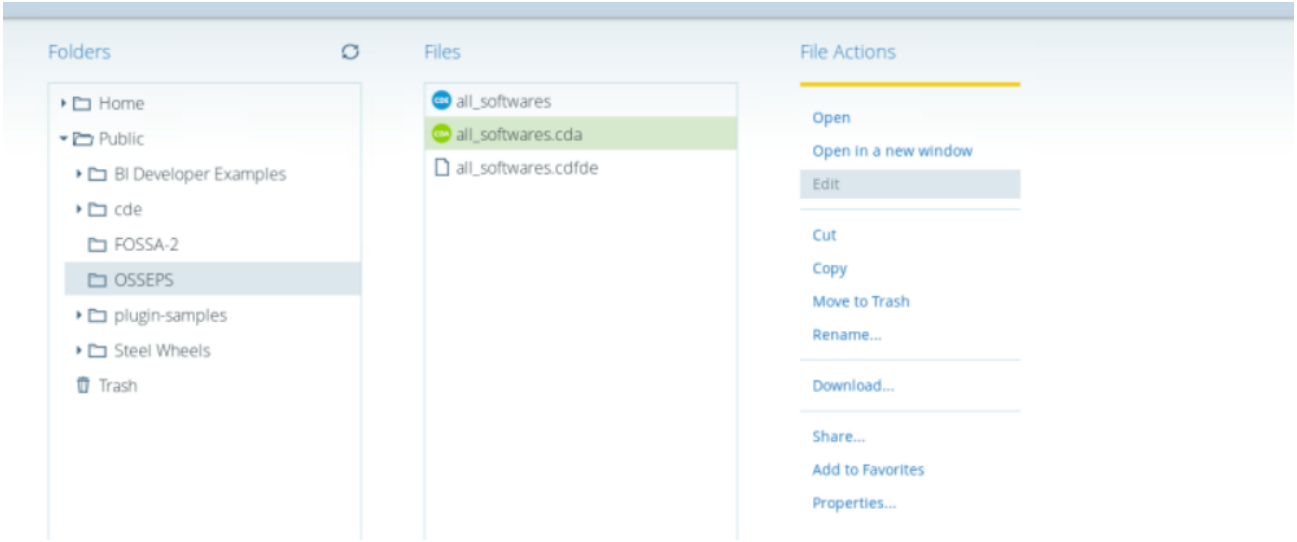


Figure 5: Editing the CDA file

The final output is the table below with the results of the query:

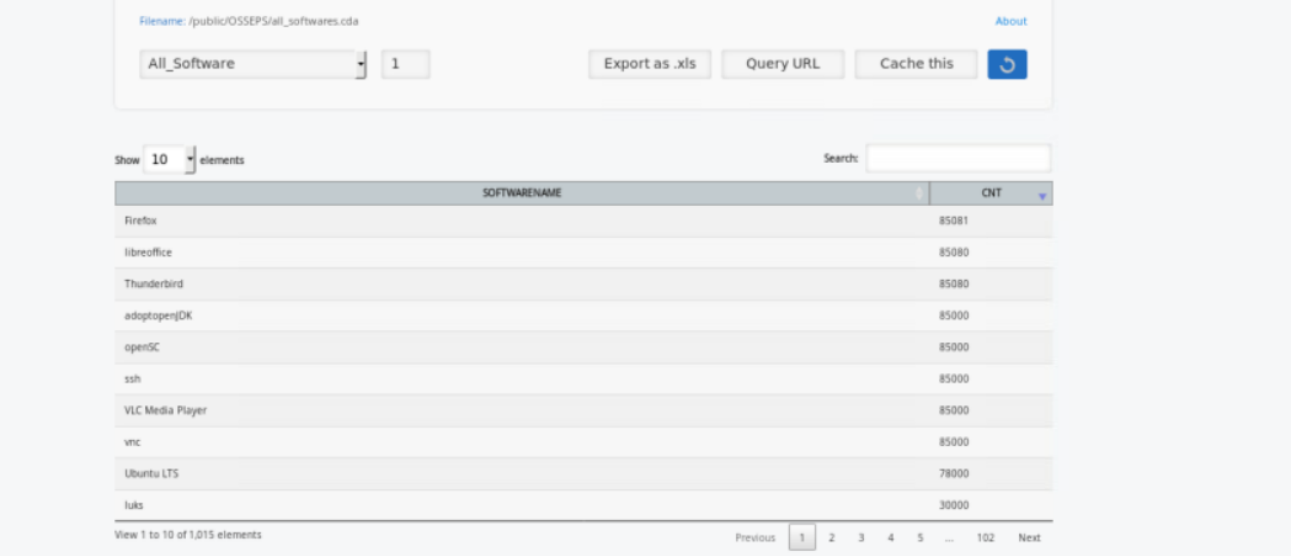
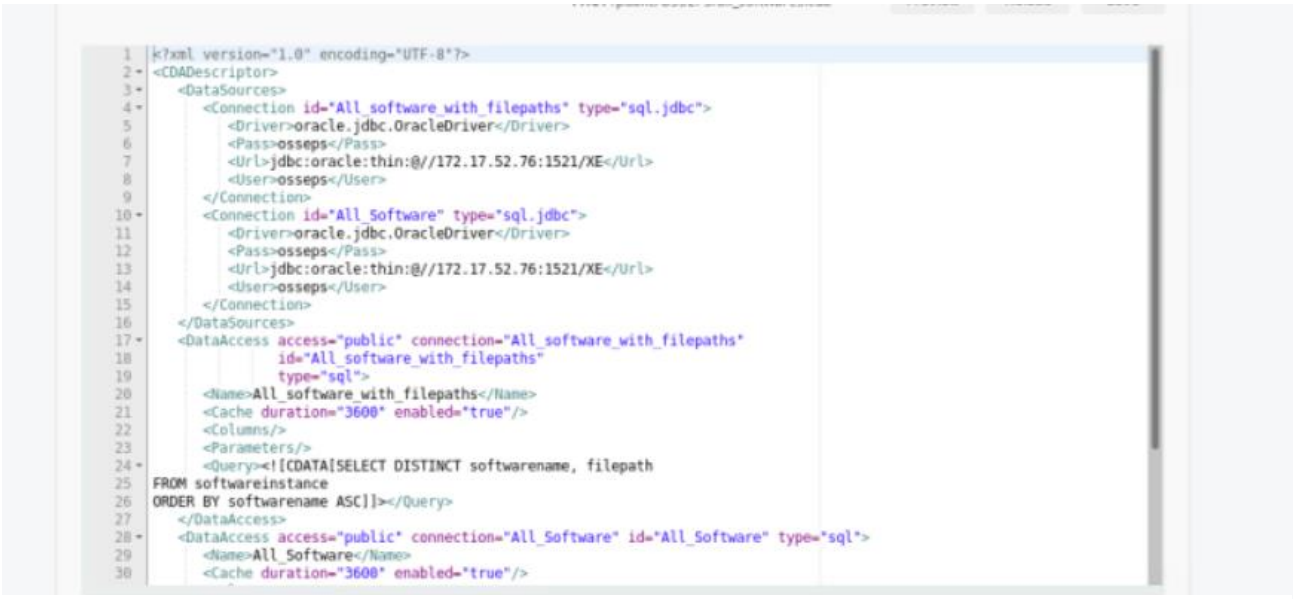


Figure 6: Final output of the results



```
<?xml version="1.0" encoding="UTF-8"?>
<CDADescriptor>
   <DataSources>
      <Connection id="All_software_with_filepaths" type="sql.jdbc">
         <Driver>oracle.jdbc.OracleDriver</Driver>
         <Pass>osseps</Pass>
         <Url>jdbc:oracle:thin:@//172.17.52.76:1521/XE</Url>
         <User>osseps</User>
      </Connection>
      <Connection id="All_Software" type="sql.jdbc">
         <Driver>oracle.jdbc.OracleDriver</Driver>
         <Pass>osseps</Pass>
         <Url>jdbc:oracle:thin:@//172.17.52.76:1521/XE</Url>
         <User>osseps</User>
      </Connection>
   </DataSources>
   <DataAccess access="public" connection="All_software_with_filepaths"
               id="All_software_with_filepaths"
               type="sql">
      <Name>All_software_with_filepaths</Name>
      <Cache duration="3600" enabled="true"/>
      <Columns/>
      <Parameters/>
      <Query><![CDATA[SELECT DISTINCT softwarename, filepath
FROM softwareinstance
ORDER BY softwarename ASC]]></Query>
   </DataAccess>
   <DataAccess access="public" connection="All_Software" id="All_Software" type="sql">
      <Name>All_Software</Name>
      <Cache duration="3600" enabled="true"/>
```

Figure 7: Final output of the query

4. ANALYSIS OF TOP SOFTWARE BASED ON THE CRITICALITY INDEX

An in-depth analysis has been performed on the defined critical software shortlist on the basis of the Business Criticality Index.

The analysis is related to the sustainability and evaluated through 34 metrics. At this stage of the project, it is important to highlight that, for the metrics related to the area of performance (e.g. time spent in code reviews and time to resolve tickets) limited information have been found. Such lack of information has resulted in reducing the average rating of the project in that area.

The inventory analysis has also covered the dependencies for each software.

The following components have more than 1 dependency upon the shortlisted items (extract from "Annex_5_Top Open Source Software dependencies.xlsx" mentioned in section 2.2):

Components	Number of Dependencies
Tomcat	67
Debian	61
OBS Studio	44
KeeFox	38
FireFox	38
Rudder	36
Chromium	33
exodus-privacy (standalone)	31
ClamAV	27
Syslog-ng	24

Figure 8: Number of dependencies per software

The analysis shows a relative fragmentation of the dependencies, apart from Glibc and Bash, which relates to the software shown below (extract from "Annex_5_Top Open Source Software dependencies.xlsx"):

Components	Number of dependencies
zlib	6
glibc	6
bash	5
glib2	5
libssl.so.10	4
libX11	4
gtk2	4
log4j	3
libx11-6	3
libstdc++6	3
gdk-pixbuf2	3
libXrender	3
rtld(GNU_HASH)	3
freetype	3
atk	3

Figure 9: Number of dependencies per component

Based on the results of our analysis, we can conclude that Tomcat is one of the software with the most dependencies. Finally, it seems that glib and zlib components remain as well at the top list of dependant software.

5. APPENDIX – ABBREVIATIONS AND ACRONYMS

WP	Work Package
DLV	Deliverable
csv	Comma-Separated Values (file format)
CMDB	Configuration Management Data Base
ETL	Extract, Transform, Load
OSS	Open Source Software
CDE	Community Dashboard Editor
SQL	Structured Query Language