Keeping Emulation Environments Portable

FP7-ICT-231954

Test description and results document for Emulation Framework

Deliverable number	D2.4
Nature	Report
Dissemination level	PU (public)
Delivery date	Due: M35 (December 2011) Actual: M36 (January 2012)
Status	Final
Work package number	WP2
Lead beneficiary	KB
Author(s)	Bart Kiers (KB) Jeffrey van der Hoeven (KB)

Document history

Revisions

Version	Date	Author	Changes
0.1	21-04-2011	Bart Kiers (KB)	Initial version
0.2	10-12-2011	Bart Kiers (KB)	Added test results
0.3	23-12-2011	Bart Kiers (KB)	Added notes from Jeffrey about feedback KEEP workshops. Made it review ready
0.4	30-12-2011	Bart Kiers (KB)	Added review comments from Winfried Bergmeyer
0.5	05-01-2012	Bart Kiers (KB)	Added review comments from Marcus Dindorf
0.6	10-01-2012	Jeffrey van der Hoeven (KB)	Adjusted template to standard deliverable output
1.0	13-01-2012	Jeffrey van der Hoeven (KB)	Finalised document
1.1	17-01-2012	Jeffrey van der Hoeven (KB)	Changed wording about KVM

Reviews

Date	Name	Result
27-12-2011	Winfried Bergmeyer (CSM)	Approved with minor changes
30-12-2011	Marcus Dindorf (DNB)	Approved with major changes

Signature/Approval

Approved by (signature)	Date

Accepted by at European Commission (signature)	Date

Executive Summary

This document describes how tests with the Emulation Framework (EF) version 1.0.0 were performed and the results of those tests. These tests were performed to:

- see if all mandatory requirements are accomplished;
- find bugs in EF release 1.0.0;
- list desired future enhancements.

Out of scope are the functionality of the emulators available in the EF and the Graphical User Interface (GUI) of the EF. The focus is on the functionality of the Core EF, Software archive and Emulator archive.

A set of tests were performed to test various aspects of the EF:

- Data integrity testing – test against corruptness of data used by the EF;
- Functional testing – test against the defined mandatory requirements by the project;
- Usability testing – collect feedback from end-users at workshops;
- Performance testing – test responsiveness of the EF;
- Stress- and volume testing – test robustness of the EF;
- Configuration testing – test behaviour when wrong parameters are set;
- Installation testing – test the software installation process.

The EF withstood all performed tests successfully. In short, the EF proved to:

- √ be reliable as it does not change the bytes of digital objects or software;
- √ meet all but one of the mandatory requirements defined by the project;
- √ satisfy the user's expectations during the workshops and user tests;
- √ have good performance in interaction with the SWA and EA;
- √ perform well under difficult circumstances such as low memory or many requests;
- √ behave well when wrong configuration entries are done;
- √ come with a solid installer.

The EF did not fully meet the requirement on the ability to run at least one KVM-based emulator. This will be solved before the end of the KEEP. Apart from that, no major shortcomings were identified.

Several useful recommendations were given during the workshops and tests:

- support for complex objects (e.g. websites, multimedia applications) by the EF;
- add language preference for emulated environment;
- give extra support to the user about the emulated environment;

- auto select the host platform (native MS Windows/Linux/MacOS);
- define separated admin and user roles;
- integrate emulator from SIMH emulation project into EF;
- start emulators and software without selecting a digital file first;
- add original software documentation and external references (web addresses) to support to the user;
- error messages are not always clear and should be improved;
- double-click on file to auto run an emulated environment;
- identification of files is not always correct, in such cases, letting the end-user provide a file format would be helpful;
- easier addition of software packages and emulators in the SWA and EA;
- possibility, or a manual, that explains how to integrate the EF with an existing repository.

The recently released version 1.1.0 of the EF already incorporates many of the recommended improvements to the EF. The final version of the KEEP EF (2.0.0) is expected to cover even more.

List of Related Documents

User Requirements Document	http://www.keep-project.eu/ezpub2/index.php?/eng/content/download/7918/39623/file/KEEP_WP2_D2.2_complete.pdf
Architectural Design Document	http://emuframework.sourceforge.net/docs/Architectural-Design-Document_1.1.pdf
Emulation Framework User Guide	http://emuframework.sourceforge.net/docs/System-User-Guide_1.1.pdf

Abbreviations

BLOB	Binary Large Object
EA	Emulator Archive
EE	Enterprise Edition
EF	Emulation Framework
GUI	Graphical User Interface
KB	National Library of the Netherlands
MB	Megabyte
NA	National Archives of the Netherlands
OS	Operating System
RAM	Random-Access Memory
SWA	Software Archive

Table of Contents

1. Introduction

This document describes how tests with the Emulation Framework version 1.0.0 were performed and the results of those tests. It focuses on the functionality of the framework and its components, rather than on the individual emulators or end-user interaction.

1.1. Objectives and scope of the tests

In May 2011, the first official public release of the EF became available[1]. Before software development started, several mandatory and optional user requirements were defined[2]. To make sure the EF meets these requirements, several tests were performed. The main objectives for doing the tests were:

- see if all mandatory requirements are accomplished;
- find bugs in EF release 1.0.0;
- list desired future enhancements.

This document can also be used as a template for tests with future releases of the EF.

Out of scope are the functionality of the emulators available in the EF and the Graphical User Interface (GUI) of the EF. The focus is on the functionality of the Core EF, Software archive and Emulator archive.

1.2. About the Emulation Framework

The Emulation Framework (EF) allows you to render digital files and computer programmes in their native environment. This offers you the potential to view these files in their intended 'look and feel', independent from current state of the art computer systems. The spectrum of potential computer platforms and applications that can be supported is practically unlimited.

Release 1.0.0 of the EF supports emulation of the x86, Commodore 64, Amiga and Amstrad CPC computer platforms. Emulation is done by using existing (open source) emulators which are carefully selected on their capability to mimic the functionality of these platforms.

The EF 1.0.0 consists of three parts:

1. Core Emulation Framework
2. Software Archive
3. Emulator Archive

The **Core EF** is the technical heart of the system, performing the workflow steps as explained before (i.e. automatic identification of file formats, selecting the required software and automatically configuring the emulation environment). For selecting software and an emulator, the Core EF interacts with the Software Archive and the Emulator Archive.

The **Software Archive** is a separate web service that contains the software (applications and operating systems) available for the EF. The download package comes with the following open source operating systems:

- FreeDOS – an open source MS DOS look-a-like operating system

[1] Emulation Framework website, available at: http://emuframework.sf.net

[2] *Requirements and design documents for services and architecture of emulation framework, D2.2a*, April 2010.

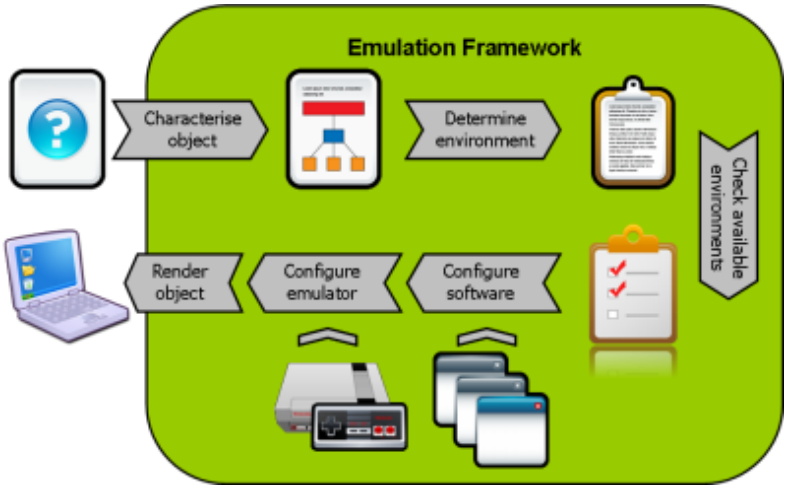
- Damn Small Linux – a small Linux kernel with limited functionality

The **Emulator Archive** is a separate web service that contains the emulators available for the EF. The download package comes with the following open source emulators:

1. Dioscuri – x86 Java-based emulator capable of running MS DOS and Linux.
2. QEMU – x86 capable of running MS Windows and Linux.
3. VICE – Commodore 64 emulator
4. UAE – Amiga emulator
5. Java CPC – Amstrad emulator
6. BeebEm – BBC Micro emulator

The Core EF, Software Archive and Emulator Archive are developed by Tessella[3] with support from the National Library of the Netherlands (Koninklijke Bibliotheek, KB)[4].

The EF is actually an automated workflow for running emulators with predefined content. It does this by following several steps. The following illustration shows which steps are taken to come from digital file to emulated computer environment.



The workflow consists of the following steps:

1. Characterise object – based on a given file (selected by a user) the EF identifies which file format it is using the tool FITS;
2. Determine environment – looks up which software and hardware is required to run the file;
3. Check available environment – matches the required environment with the best environment available in the EF;
4. Configure software – retrieves selected software from the software archive and wraps the given file into a disk image;

[3] Tessella, website available at: http://www.tessella.com

[4] National Library of the Netherlands, website available at: http://www.kb.nl

5. Configure emulator – retrieves selected emulator from the emulator archive and configures it using emulator specific templates. Attaches software and disk image containing the file;
6. Render object – launch the prepared emulation environment.

1.3. About the KEEP project

KEEP (Keeping Emulation Environments Portable) is an international research project co-funded by the European Union 7th Framework Programme. It does research into an emulation-based preservation strategy and develops several tools to support that. The consortium consists of eight organisations representing a wide range of stakeholders in Europe: cultural heritage institutes, research institutes, commercial ICT partners and the gaming industry. The project has a duration of three years and ends in February 2012.

More information can be found on the KEEP website[5].

1.4. Outline of this document

Chapter 2 describes the Test Plan which was used to gather all information necessary to plan and control the test effort for testing the EF. Chapter 3 lists the results of these tests. Chapter 4 summarises the conclusions of the individual tests and offers a couple of recommendations for future developments of the EF.

This document contains a short glossary describing uncommon terms used throughout this document and two appendices: JMeter test profile that can be used to replicate stress-, volume- and performance tests, and an overview of all system functions (methods) of the EF.

[5] KEEP project website, available at: : http://www.keep-project.eu

2. Test plan

This chapter lists all information that was needed to plan and control the tests with the EF. Tests were performed against the public software release of the EF in version 1.0.0 [6].

It describes the approach to testing the software, and was the top-level plan used to direct the test effort, and will be, in future tests.

2.1. Objectives and scope

The Test Plan for the EF covers the following objectives:

- identify the items targeted by the tests;
- identify the motivation for, and ideas behind, the covered test areas;
- outline the testing approach;
- identify the required resources;
- list the elements involved in the test activities;
- test the presence of all mandatory requirements.

Subject of the tests are:

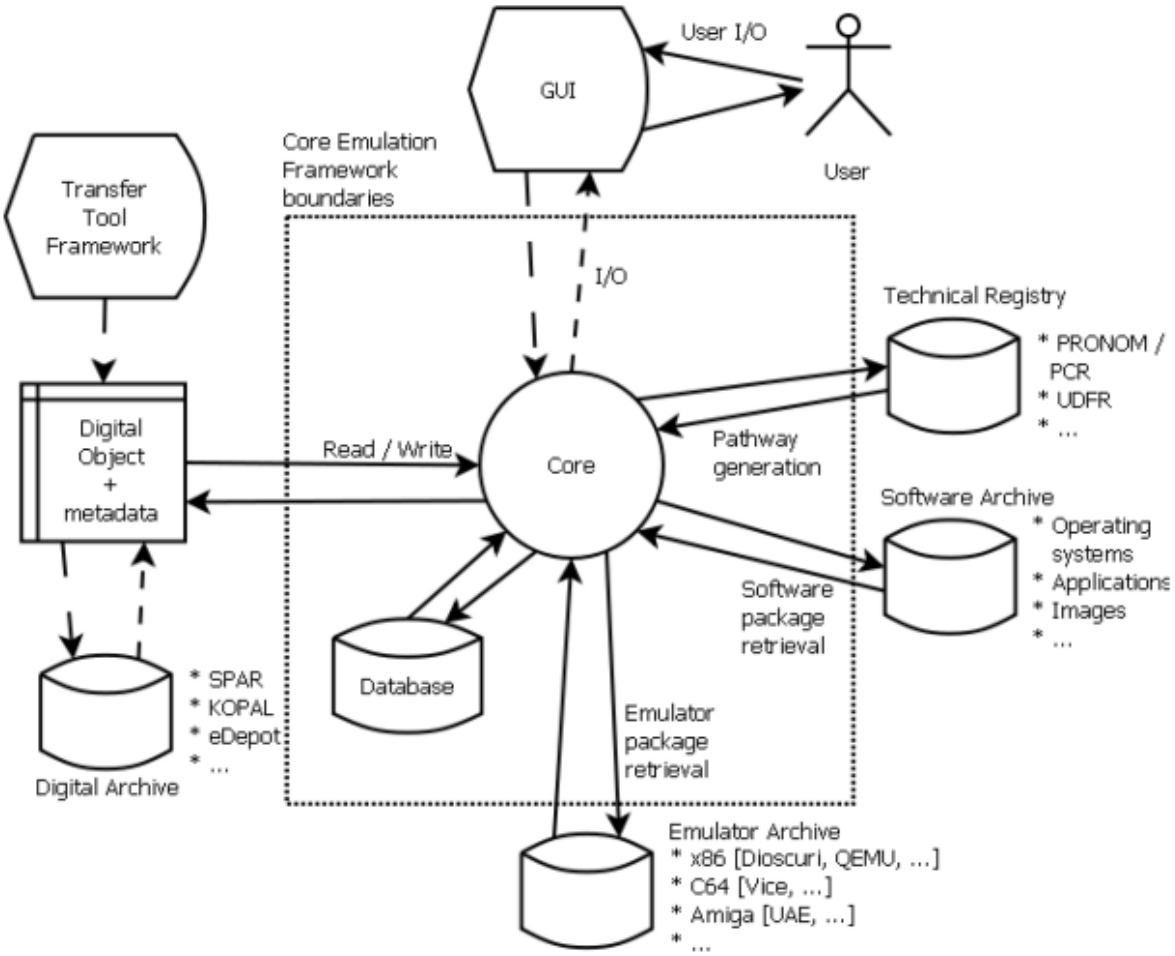
- Core functionality of the EF;
- Software Archive (SWA henceforth);
- Emulator Archive (EA henceforth).

Note: the Graphical User Interface (GUI) is not subject of the tests as this is part of WP3. Neither are the emulators managed by the EF. Instead, the focus is on the functional behaviour of the core components of the EF itself.

The relation between the core functionality and its external interfaces is given in the following architectural diagram. For more explanation, please read the Architectural Design Document[7] (D2.2b).

[6] http://sourceforge.net/projects/emuframework/files/Release_1.0.0/

[7] *Requirements and design documents for services and architecture of emulation framework, D2.2b*, April 2010.



2.2. Test Environment

The following software is used in the setup for the tests:

- KEEP Emulation Framework version 1.0.0;
- Java SE[8] version 1.6;
- Java EE 5 web services;
- H2[9] database;
- Microsoft Windows XP (client OS);
- Ubuntu 11.10 (client OS);
- Microsoft Windows 2008 Server (server OS) configured as a virtual server running in a VMware server cluster consisting of four quad-core blade servers;

Test utilities:

- Apache JMeter[10]

8 http://www.h2database.com

9 http://www.oracle.com/technetwork/java/javase

10 Apache JMeter, software available at: http://jmeter.apache.org/

- WinImage 8.1[11]
- JUnit test framework[12]
- Apache Ant[13]

The following hardware is used in the setup for the tests:

- **Client machine:** HP Compaq, Intel Core2 Quad Q9400 2.66 MHz, 4 GB RAM, NVIDIA Quadro NVS 290.
- **Server machine:** virtual server running in a VMware server cluster consisting of four quad-core HP blade servers.
- **Network:** 1 Gbit Ethernet (LAN) connection.

2.3. Test Techniques and Types

The test plan entails different types of testing:

- Data integrity testing
- Functional testing
- Usability testing
- Performance testing
- Stress- and volume testing
- Configuration testing
- Installation testing

Each of these tests are described in detail in the following sections.

2.3.1. Data Integrity Testing

Data integrity tests are performed to ensure files rendered by the EF are not manipulated in any way. It is also done to ensure that binary data stored in the H2 database, like emulators and software packages, do not differ after being extracted from the database.

Objective	Verify that no data corruption takes place after rendering a digital object, or extracting software from the database.
Tool(s) used	- Windows XP's **comp**[14] command which compares the contents of two (or more) files byte-by-byte - H2 database - WinImage, a disk imaging tool

11 WinImage, software available at: http://www.winimage.com/

12 JUnit test framework, software available at: http://www.junit.org/

13 Apache Ant, software available at: http://ant.apache.org/

Success criteria	A copy of the rendered file is byte-by-byte identically after once successfully rendered within the EF. An emulator, or software image, is byte-by-byte identically after once successfully extracted from the H2 database within the EF.

2.3.2. Functional Testing

Functional tests are performed to verify all functional requirements are successfully met. This will be accomplished through black-box testing against the defined requirements of the EF as documented in deliverable D2.2a. All functions will be tested using unit tests.

Unit testing is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application[15].

Objective	Verify all system functional requirements are met.
Tool(s) used	JUnit, Apache Ant
Success criteria	All of the mandatory requirements are implemented and successfully passed their unit tests.

2.3.3. Usability Testing

Usability tests are performed to verify the end-user's expectations of the EF's functionality and to identify features that would be beneficial to incorporate in the EF. It also verifies that all mandatory functionality in the EF is able to be operated in terms of Human Machine Interface (HMI) best-practices through a test GUI. However, the GUI itself is not part of this test.

Objective	Verify that the EF fulfils the end-user's expectations which were collected as user feedback.
Tool(s) used	Hold workshops and collect feedback from end-users. Test at the National Archives of the Netherlands with a couple of digital objects from their collection. Test at the National Library of the Netherlands (KB) with a couple of digital objects from their collection.

[14] http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/comp.mspx

[15] Definition of unit testing, by Wikipedia.org, available at: http://en.wikipedia.org/wiki/Unit_testing

Success criteria	All end-user expectations were successfully met.

2.3.4. Performance Testing

In the performance tests, response times, transaction rates, and other time-sensitive requirements are measured and evaluated. The goal of these tests is to verify if performance requirements have been achieved. Performance profiling is implemented and executed to profile and tune target-of-test's performance behaviours as a function of conditions such as workload or hardware configurations.

Objective	Evaluate performance related tests under varying workload. The following performance measurements will be tested: • response times for each of the public methods of the SWA through a web service call; • response times for each of the public methods of the EA through a web service call.
Tool(s) used	Apache JMeter
Success criteria	Met the performance requirement.

2.3.5. Stress- and Volume Testing

Stress- and volume testing is a type of performance test to understand how (or if) a system fails due to conditions at the boundary, or outside of it. For example, a large amount of requests are performed in a short time-frame from multiple clients. In case of stress testing, this typically involves low resources or competition for resources. Low resource conditions reveal how the target-of-test fails that is not apparent under normal conditions.

Objective	Determine the behaviour under a large amount of requests from different clients. Determine the behaviour under a low assignment of RAM.
Tool(s) used	Apache JMeter
Success criteria	The EF will be still operational and when problems do occur meaningful error messages will be generated by the core of the EF.

2.3.6. Configuration Testing

This type of test evaluates how the core of the EF behaves when executed with unexpected or wrong configuration settings provided.

Objective	Evaluate the behaviour of the EF, and components of it, when executed with unexpected or wrong configuration settings provided.
Tool(s) used	A locally installed version of the EF.
Success criteria	The EF, or relevant components of the EF, should be able to be terminated by the test user and produce meaningful error messages after incorrect configuration settings have been provided.

2.3.7. Installation Testing

Installation testing ensures that the deployment and installation of the EF software package works as expected.

The following will be taken into account during installation testing of the EF:

- ensure that the software can be installed under different conditions such as a clean installation, or an installation in a directory in which a copy of the EF already exists;
- evaluate the behaviour of the EF when installing on a location with insufficient disk space;
- observe what happens after installing on a location with lack of privileges;
- verify that the software can be started after installation.

Objective	Evaluate the behaviour of the EF under specific or even unexpected system environment scenarios.
Tool(s) used	The EF installer (part of EF release 1.0.0).
Success criteria	Installation is successful, or if not, exit or wait with a meaningful error message.

3. Test results

This chapter contains the test results of the tests that were outlined in chapter 2. The result of each type of test is explained here.

3.1. Data Integrity Test Results

Objective	Verify that no data corruption takes place after rendering a digital object, or extracting software from the database.
Tool(s) used	- Windows XP's **comp**[16] command which compares the contents of two (or more) files byte-by-byte - H2 database - WinImage
Success criteria	A copy of the rendered file is byte-by-byte identical after rendering. An emulator, or software image, is byte-by-byte identical after extracting it from the database.

3.1.1. Rendered digital object

After rendering a digital object using `start(FILE)`, the FILE is wrapped in a disk image which in its turn is launched together with the emulator. When terminating the emulator and extracting the FILE using WinImage, the `comp` utility was used to see if the file was still the same as the original.

Tests were performed with two types of files:

1. flat ASCII text file;
2. binary JPG image file.

Both files were emulated using Dioscuri with FreeDos and Qemu with Damn Small Linux, both part of the 1.0.0 release of the EF. In all four cases, the original file was byte-by-byte the same as the rendered file.

3.1.2. Emulator and Software package

All emulators in the EA, and all software packages in the SWA have been extracted with the following SQL queries:

[16] http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/comp.mspx

- SELECT package FROM emulatorarchive.emulators
- SELECT image FROM softwarearchive.imageblobs

Both these queries return a BLOB (a binary large object). These BLOBs were then compared to the original files located in the directories:

- EmulatorArchive/packages/emulators/
- SoftwareArchive/packages/

In all cases, there was no difference: they were 100% byte-by-byte equivalent.

3.1.3. Conclusion

Nor rendered objects, or the software- or emulator packages provided by the EF are changed in any way by the EF, or one of its components.

3.2. Functional Test Results

Objective	Verify all system functional requirements.
Tool(s) used	JUnit, Apache Ant
Success criteria	All of the Mandatory requirements are implemented and pass their unit tests.

3.2.1. Mandatory requirements

All mandatory requirements are defined in deliverable 2.2a[17].

The public API is formed by all system functions (methods) of the EF. The full list of functions can be found in appendix B of this document. The API is the public interface to be used by other developers to be able to incorporate the EF core in their own software (e.g. a GUI or technical registry).

In the following table each mandatory requirement is stated with the resulting implementation in the EF.

Requirement #	Description	Result	Requirement fulfilled
F2.1	The EF should offer an API for accepting digital object and metadata, preferably standardized.	boolean start(File file, File metadata)	yes

[17] *Requirements and design documents for services and architecture of emulation framework, D2.2a*, April 2010.

Requirement #	Description	Result	Requirement fulfilled
F2.2	The EF API should acknowledge the receipt of the digital object and metadata	The start method returns a boolean value for this purpose.	yes
F2.3	Given an atomic digital object and metadata, the EF should be able to connect to one or more technical registries to generate a list of possible emulation pathways	This is done by several method calls: a) the file needs to characterized; b) a certain file format can produce multiple pathways: a) List<Format> characterise(File digObj) b) List<Pathway> getPathways(Format format)	yes
F3.1	The EF should be able to run at least three types of emulators	Release 1.0.0 contains 6 emulators: Dioscuri, Qemu, VICE, UAE, BeebEm and JavaCPC. Calling start(FILE), where FILE is a local file will launch the proper emulator, assuming FILE is known in the EF database.	yes
F3.2	The EF should be able to run at least one Java-based emulator	Both Dioscuri and JavaCPC are emulators written in Java.	yes
F3.3	The EF should be able to run at least one KVM-based emulator	The EF is already capable of running any emulator that can be invoked via the command line. At the time of writing of this document, it was not yet possible to test integration of the EF with the KVM version of the C64 emulator VICE. This will be carried out before the end of the project.	partly
F4.1	EF should be able to prepare a 'software package' based on available pathway, digital object and any additional metadata	This can be done in 3 steps: a) get the file format of a digital object; b) get the possible pathways for a particular file format; c) get a list of software packages for a particular pathway. a) List<Format> characterise(File digObj) b) List<Pathway> getPathways(Format format) c) List<SoftwarePackage> getSoftwareByPathway(Pathway pathway)	yes
F4.2	EF should be able to handle at least two types of atomic digital objects	Release 1.0.0 of the EF support the rendering of 21 atomic objects: Amiga Disk Image, Amstrad Tape Image, Amstrad Disk Image, BBC Micro Image, Commodore C64 Tape Image, Commodore C64 Disk Image, Portable Document Format, Extensible Markup Language, Plain text, JPEG File Interchange Format,	yes

Requirement #	Description	Result	Requirement fulfilled
		Windows Bitmap, Graphics Interchange Format, Tagged Image File Format, Portable Network Graphics, Hypertext Markup Language, WordPerfect for MS-DOS/Windows Document, Microsoft Word, Motorola Quark Express Document, ARJ archive data, LHarc 1.x/ARX archive data [lh0], DOS/Windows executable and ISO 9660 CD-ROM.	
F4.6	EF should be able to display the atomic digital object in the emulated environment	The method start(FILE), where FILE is a file format of any of the 21 listed in F4.2, will cause the appropriate emulator to be launched.	yes
F4.8	EF should be able to configure the selected emulator	The method start(FILE), where FILE is a file format of any of the 21 listed above, will cause the appropriate emulator to properly configured and launched.	yes
F5.1	The end-user should be able to start and stop emulation processes	This can be done by calling either: - boolean start(File file) or - boolean stop()	yes
F5.3	The end-user should not be required to configure the emulation process, i.e. the EF provides a default configuration	See F4.8	yes
F5.12	The administrator should be able to configure the connection to the technical registries	This can be done in the user.properties file.	yes
F5.13	The administrator should be able to configure the amount of information displayed to the end-user	This can be done in the log4j.properties file.	yes

Using Apache Ant, the task “test.report” can be executed which performs black box tests on all public, protected and package protected methods in the EF core, including all method calls in the table above. It will also produce an HTML report inside build-directory that shows all tests passed without failures or errors.

3.2.2. Conclusion

All mandatory functional requirements are implemented except F3.3 (running an KVM based emulator). To accomplish F3.3 the EF development team is working closely with the KVM development team. All implemented requirements also pass their unit tests without warnings, or errors.

3.3. Usability Test Results

Objective	Verify the EF fulfils the end-user's expectations or collect features it currently lacks.
Tool(s) used	Hold workshops and collect feedback from end-users. Test at the National Archives of the Netherlands with a couple of digital objects from their collection. Test at the National Library of the Netherlands (KB) with a couple of digital objects from their collection.
Success criteria	No feedback (all expectations were met), or features to incorporate in the EF.

3.3.1. Results from workshops and tests

Workshops have been organised to let interested people get acquainted with the EF and what it could do for them in their organisation. At the end of these workshops, and during performed demo's of the EF, feedback was collected. Data from the following workshops were used in this test report:

- Paris workshop – 23 September 2011;
- The Hague workshop – 27 October 2011;
- Rome workshop – 30 November 2011.

Besides these workshops, both the National Archives of the Netherlands (NA henceforth), and The National Library of the Netherlands (KB henceforth) performed tests with digital objects from their collections.

Below is a table of remarks and recommendations gathered from these workshops and tests. In the meantime, a new version of the EF (1.1.0)[18] has been released that implements some of the remarks/wishes. Furthermore, version 2.0.0 is in preparation and might include the other remarks that did not make it in 1.1.0. This is indicated in the table below as well.

Remark / wish	Implemented in 1.1.0	Possibly in 2.0.0
Add user feedback on rendering environment for selecting the best emulated environment.	-	-
Add language preference for emulated environment. This can be used to auto select the native language in the rendered environment so that the user gets the preferred	Yes	-

[18] KEEP EF version 1.1.0, software available at: http://emuframework.sf.net

Remark / wish	Implemented in 1.1.0	Possibly in 2.0.0
language in the software.		
Give extra support to the user about the emulated environment. Example: which command should be entered in a command-line environment.	yes	-
Ensure complex objects can be run (such as CD-ROMs, websites, folders with multiple files in them).	-	yes
Auto select the host platform (native MS Windows/Linux/MacOS) so that the EF only shows the environments that actually can run on the computer of the user.	yes	-
Define separated admin and user roles.	yes	-
Integrate emulator from SIMH emulation project into EF.	-	yes
Start emulators and software without selecting a digital file first.	-	yes
Add original software documentation and external references (web addresses) to support to the user.	yes	-
Error messages are not always clear.	work in progress	yes
Double-click on file to auto run.	yes	-
Identification of files is not always correct, in such cases, letting the end-user provide a file format would be helpful.	-	yes
Easier addition of software packages and emulators in the SWA and EA.	yes	-
Possibility, or a manual that explains how to integrate the EF with an existing repository.	-	-

3.3.2. Conclusion

The first release of the EF (1.0.0) is a decent prototype which includes a solid amount of features. But during the workshops, and tests at NA and KB, one remark/wish came back: there is a real need for the EF to be able to render complex digital objects, such as websites and multimedia software.

3.4. Performance Test Results

Objective	Determine the behaviour under varying workload.
Tool(s) used	Apache JMeter
Success criteria	Met the performance requirement.

3.4.1. Setup

As the biggest performance challenge lies between the interaction of EF with the web services EA and SWA, the test focused on the performance of retrieving software and emulators from the archives to the EF. The following setup was used:

- both archives ran on a Virtualized Windows 2008 server edition, with 512 Megabytes of RAM assigned each, and 4 (virtual) processors available on the Windows 2008 server;
- in the same 1 Gbit LAN, 4 JMeter clients (all on Windows XP PC's) with 100 Mbit Ethernet adapter were setup to perform the web service requests to both the EA and SWA;
- in the same 1 Gbit LAN, 1 PC with 100 Mbit Ethernet adapter instructed all the 4 JMeter clients to tell them what, and when, to request from the EA and SWA.

The JMeter profile is provided in Appendix A, which can be used to reproduce the tests. Note that the profile XML file assumes the EA and SWA are running on http://keep.wpakb.kb.nl through ports 9000 and 9001. These settings will need to be adjusted if the host and/or ports differ, of course.

Each of the four clients were instructed to call the following public methods from the EA and SWA with varying times in between the requests:

method	archive	# MB returned
DownloadEmulator	EA	2
GetEmulatorPackage	EA	< 1
GetEmulatorPackageList	EA	< 1
GetEmusByHardware	EA	< 1
GetSupportedHardware	EA	< 1
DownloadSoftware	SWA	50
GetAllSoftwarePackagesInfo	SWA	< 1
GetPathwaysByFileFormat	SWA	< 1
GetSoftwarePackageInfo	SWA	< 1

3.4.2. Results

In the table below is a list of average return times of each of the public methods of the EA and SWA during a period of 15 minutes of continued requests:

EF function (method)	pause between requests (sec.)	average response (ms.)
DownloadEmulator	10	500
	5	500
	1	800
GetEmulatorPackage	10	< 10
	5	< 10
	1	< 10
GetEmulatorPackageList	10	< 10
	5	< 10
	1	< 10
GetEmusByHardware	10	< 10
	5	< 10
	1	< 10
GetSupportedHardware	10	< 10
	5	< 10
	1	< 10
DownloadSoftware	10	2800
	5	3100
	1	4000
GetAllSoftwarePackagesInfo	10	< 10
	5	< 10
	1	< 10
GetPathwaysByFileFormat	10	< 10
	5	< 10
	1	< 10

Changing the parameters to the called methods into invalid ones (i.e. if a method expects a number, provide a string instead), did not result in any unwanted behaviour. The web services simply returned an error message stating the parameter was incorrect.

3.4.3. Conclusion

Most of the public methods from the EA and SWA returned what they are supposed to return in less than 10 milliseconds. Downloading an emulator every second still did not exceed more than 1 second in response. Only the Linux package took a bit longer (~4 seconds) when requesting it once every second from each of the 4 clients. But this is still an acceptable time.

3.5. Stress- and Volume Test Results

Objective	Determine the behaviour under a large amount of requests to the EA and SWA from different clients. Determine the behaviour under a low assignment of RAM.

Tool(s) used	Apache JMeter
Success criteria	No error messages, or when problems do occur, meaningful error messages.

3.5.1. Large amount of requests

See performance test results.

3.5.2. Low memory

Running the same test outlined in 3.4, but then assigning less than 64 MB of RAM to each of the archives, caused the SWA to terminate with a OutOfMemoryException in its log files. The EA had no such problems: it ran in more or less the same time as indicated in 3.4.

3.5.3. Conclusion

The total amount of memory assigned to each of the EA and SWA archives needs to be at least 64 MB of RAM. Preferably at least as much RAM as the largest image in its archive is. So if there is an image containing Windows 98 with MS Office included, totalling 200 MB, then the RAM is recommended to be at least 256 MB for the SWA. This could become an issue if an image of more than 8 GB is managed by the SWA. But in practice, images are compressed and it is not likely to transfer images of that size. Nevertheless, administrators should be made aware of these kind of limitations.

3.6. Configuration Test Results

Objective	Determine if the EF, or components of it, produce meaningful messages after providing incorrect configuration settings, and observe if the EF terminates in a proper way (no crashes).
Tool(s) used	A locally installed version of the EF. Embedded H2 database.
Success criteria	Meaningful error messages after providing invalid settings.

3.6.1. Results

setting	**resulting in**
Invalid username and/or password to internal database	5 retries and then an expected message: java.io.IOException: Multiple connection attempts to database failed: org.h2.jdbc.JdbcSQLException: Wrong user name or password [8004-133] ...
Invalid internal database name	5 retries and then an expected message: Database "/home/bart/Documents/KEEP/EF110/database/h2/EF_engineE" not found [90013-133]

	...
Invalid EA or SWA web address (URL)	Results in the following error message: EmulatorArchive or SoftwareArchive could not be contacted.
If in admin mode, an invalid username and/or password	5 retries and then an expected message: java.io.IOException: Multiple connection attempts to database failed: org.h2.jdbc.JdbcSQLException: Wrong user name or password [8004-133] ...

3.6.2. Conclusion

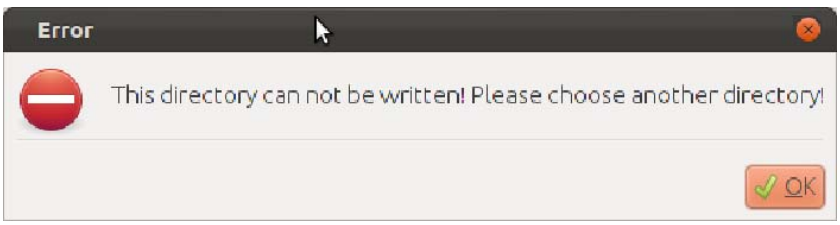
Purposely providing incorrect information for those settings that are mandatory for the EF to operate properly results in the EF to gracefully exit with meaningful error messages.

3.7. Installation Test Results

Objective	Observe the behaviour when installing the EF under varying circumstances.
Tool(s) used	The EF installer.
Success criteria	Installation is successful, or if not, exit or wait with a meaningful error message.

3.7.1. Installation without sufficient rights

Trying to install the EF in a directory without sufficient rights, produces the following error:



The installation process does not terminate, but remains at the present step: it waits for a valid installation directory.

3.7.2. Installation while another installation is running

When trying to run the EF-installer while another instance of the installer is already running, produces the following warning:

I.e., the installation can continue, but the installer warns the user about this.

3.7.3. Installation on a USB stick without sufficient space

When trying to install the EF on a disk without sufficient space, the following error is produced:

The installation process does not terminate, but remains at the present step: it waits for a valid installation directory.

3.7.4. A valid installation

Installing the EF on a disk with enough free space, and with proper user rights to perform the installation, results in a proper installation. The EF can be launched by running either the `runEF.bat` or `runEF.sh` file (assuming there is a EA- and SWA running on `localhost`, or another location).

3.7.5. Conclusion

The installer produces meaningful warnings, or error messages, and does not crash when trying to do something invalid. After a valid installation, the EF is launched without a problem.

4. Conclusions and recommendations

The EF withstood all performed tests successfully. In short, the EF proved to:

- √ be reliable as it does not change the bytes of digital objects or software;
- √ meet all but one of the mandatory requirements defined by the project;
- √ satisfy the user's expectations during the workshops and user tests;
- √ have good performance in interaction with the SWA and EA;
- √ perform well under difficult circumstances such as low memory or many requests;
- √ behave well when wrong configuration entries are done;
- √ come with a solid installer.

The EF did not fully meet the requirement on the ability to run at least one KVM-based emulator. This will be solved before the end of the KEEP project. Apart from that, no major shortcomings were identified.

Several useful recommendations were given during the workshops and tests:

- support for complex objects (e.g. websites, multimedia applications) by the EF;
- add language preference for emulated environment;
- give extra support to the user about the emulated environment;
- auto select the host platform (native MS Windows/Linux/MacOS);
- define separated admin and user roles;
- integrate emulator from SIMH emulation project into EF;
- start emulators and software without selecting a digital file first;
- add original software documentation and external references (web addresses) to support to the user;
- error messages are not always clear and should be improved;
- double-click on file to auto run an emulated environment;
- identification of files is not always correct, in such cases, letting the end-user provide a file format would be helpful;
- easier addition of software packages and emulators in the SWA and EA;
- possibility, or a manual, that explains how to integrate the EF with an existing repository.

The recently released version 1.1.0 of the EF already incorporates many of the recommended improvements to the EF. The final version of the KEEP EF (2.0.0) is expected to cover even more.

Appendix A: Jmeter test profile

```
<jmeterTestPlan version="1.2" properties="2.1">
<hashTree>
<TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Test Plan" enabled="true">
<stringProp name="TestPlan.comments"/>
<boolProp name="TestPlan.functional_mode">false</boolProp>
<boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
<elementProp name="TestPlan.user_defined_variables" elementType="Arguments" guiclass="ArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">
<collectionProp name="Arguments.arguments"/>
</elementProp>
<stringProp name="TestPlan.user_define_classpath"/>
</TestPlan>
<hashTree>
<ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="KEEP webservice users/connections" enabled="true">
<stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
<elementProp name="ThreadGroup.main_controller" elementType="LoopController" guiclass="LoopControlPanel" testclass="LoopController" testname="Loop Controller" enabled="true">
<boolProp name="LoopController.continue_forever">false</boolProp>
<intProp name="LoopController.loops">-1</intProp>
</elementProp>
<stringProp name="ThreadGroup.num_threads">4</stringProp>
<stringProp name="ThreadGroup.ramp_time">5</stringProp>
<longProp name="ThreadGroup.start_time">1317806375000</longProp>
<longProp name="ThreadGroup.end_time">1317806375000</longProp>
<boolProp name="ThreadGroup.scheduler">false</boolProp>
<stringProp name="ThreadGroup.duration"/>
<stringProp name="ThreadGroup.delay"/>
</ThreadGroup>
<hashTree>
<WebServiceSampler guiclass="WebServiceSamplerGui" testclass="WebServiceSampler" testname="ea :: DownloadEmulator" enabled="true">
<elementProp name="HTTPsampler.Arguments" elementType="Arguments">
<collectionProp name="Arguments.arguments"/>
</elementProp>
<stringProp name="HTTPSampler.domain">keep.wpakb.kb.nl</stringProp>
<stringProp name="HTTPSampler.port">9001</stringProp>
<stringProp name="HTTPSampler.protocol">http</stringProp>
<stringProp name="HTTPSampler.path">/emulatorarchive/</stringProp>
<stringProp name="WebserviceSampler.wsdl_url">http://keep.wpakb.kb.nl:9001/emulatorarchive?wsdl</stringProp>
<stringProp name="HTTPSampler.method">POST</stringProp>
<stringProp name="Soap.Action">urn:DownloadEmulator</stringProp>
<stringProp name="HTTPSamper.xml_data">
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:emul="http://emulatorarchive.keep.eu"> <soapenv:Header/> <soapenv:Body> <emul:emulatorID_2>3</emul:emulatorID_2> </soapenv:Body> </soapenv:Envelope>-
</stringProp>
<stringProp name="WebServiceSampler.xml_data_file"/>
<stringProp name="WebServiceSampler.xml_path_loc"/>
<stringProp name="WebserviceSampler.timeout"/>
<stringProp name="WebServiceSampler.memory_cache">true</stringProp>
<stringProp name="WebServiceSampler.read_response">true</stringProp>
<stringProp name="WebServiceSampler.use_proxy">false</stringProp>
<stringProp name="WebServiceSampler.proxy_host"/>
<stringProp name="WebServiceSampler.proxy_port"/>
</WebServiceSampler>
<hashTree/>
```

<WebServiceSampler guiclass="WebServiceSamplerGui" testclass="WebServiceSampler" testname="ea :: GetEmulatorPackage" enabled="true">
<elementProp name="HTTPsampler.Arguments" elementType="Arguments">
<collectionProp name="Arguments.arguments"/>
</elementProp>
<stringProp name="HTTPSampler.domain">keep.wpakb.kb.nl</stringProp>
<stringProp name="HTTPSampler.port">9001</stringProp>
<stringProp name="HTTPSampler.protocol">http</stringProp>
<stringProp name="HTTPSampler.path">/emulatorarchive/</stringProp>
<stringProp name="WebserviceSampler.wsdl_url">http://keep.wpakb.kb.nl:9001/emulatorarchive?wsdl</stringProp>
<stringProp name="HTTPSampler.method">POST</stringProp>
<stringProp name="Soap.Action">urn:GetEmulatorPackage</stringProp>
<stringProp name="HTTPSamper.xml_data">
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:emul="http://emulatorarchive.keep.eu"> <soapenv:Header/> <soapenv:Body> <emul:emulatorID_1>4</emul:emulatorID_1> </soapenv:Body> </soapenv:Envelope>
</stringProp>
<stringProp name="WebServiceSampler.xml_data_file"/>
<stringProp name="WebServiceSampler.xml_path_loc"/>
<stringProp name="WebserviceSampler.timeout"/>
<stringProp name="WebServiceSampler.memory_cache">true</stringProp>
<stringProp name="WebServiceSampler.read_response">true</stringProp>
<stringProp name="WebServiceSampler.use_proxy">false</stringProp>
<stringProp name="WebServiceSampler.proxy_host"/>
<stringProp name="WebServiceSampler.proxy_port"/>
</WebServiceSampler>
<hashTree/>
<WebServiceSampler guiclass="WebServiceSamplerGui" testclass="WebServiceSampler" testname="ea :: GetEmulatorPackageList" enabled="true">
<elementProp name="HTTPsampler.Arguments" elementType="Arguments">
<collectionProp name="Arguments.arguments"/>
</elementProp>
<stringProp name="HTTPSampler.domain">keep.wpakb.kb.nl</stringProp>
<stringProp name="HTTPSampler.port">9001</stringProp>
<stringProp name="HTTPSampler.protocol">http</stringProp>
<stringProp name="HTTPSampler.path">/emulatorarchive/</stringProp>
<stringProp name="WebserviceSampler.wsdl_url">http://keep.wpakb.kb.nl:9001/emulatorarchive?wsdl</stringProp>
<stringProp name="HTTPSampler.method">POST</stringProp>
<stringProp name="Soap.Action">urn:GetEmulatorPackageList</stringProp>
<stringProp name="HTTPSamper.xml_data">
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:emul="http://emulatorarchive.keep.eu"> <soapenv:Header/> <soapenv:Body> <emul:dummyElement>0</emul:dummyElement> </soapenv:Body> </soapenv:Envelope>
</stringProp>
<stringProp name="WebServiceSampler.xml_data_file"/>
<stringProp name="WebServiceSampler.xml_path_loc"/>
<stringProp name="WebserviceSampler.timeout"/>
<stringProp name="WebServiceSampler.memory_cache">true</stringProp>
<stringProp name="WebServiceSampler.read_response">true</stringProp>
<stringProp name="WebServiceSampler.use_proxy">false</stringProp>
<stringProp name="WebServiceSampler.proxy_host"/>
<stringProp name="WebServiceSampler.proxy_port"/>
</WebServiceSampler>
<hashTree/>
<WebServiceSampler guiclass="WebServiceSamplerGui" testclass="WebServiceSampler" testname="ea :: GetEmusByHardware" enabled="true">
<elementProp name="HTTPsampler.Arguments" elementType="Arguments">
<collectionProp name="Arguments.arguments"/>
</elementProp>
<stringProp name="HTTPSampler.domain">keep.wpakb.kb.nl</stringProp>
<stringProp name="HTTPSampler.port">9001</stringProp>
<stringProp name="HTTPSampler.protocol">http</stringProp>

<stringProp name="HTTPSampler.path">/emulatorarchive/</stringProp>

<stringProp name="WebserviceSampler.wsdl_url">http://keep.wpakb.kb.nl:9001/emulatorarchive?wsdl</stringProp>

<stringProp name="HTTPSampler.method">POST</stringProp>

<stringProp name="Soap.Action">urn:GetEmusByHardware</stringProp>

<stringProp name="HTTPSamper.xml_data">

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:emul="http://emulatorarchive.keep.eu"> <soapenv:Header/> <soapenv:Body> <emul:hardwareID>x86</emul:hardwareID> </soapenv:Body> </soapenv:Envelope>

</stringProp>

<stringProp name="WebServiceSampler.xml_data_file"/>

<stringProp name="WebServiceSampler.xml_path_loc"/>

<stringProp name="WebserviceSampler.timeout"/>

<stringProp name="WebServiceSampler.memory_cache">true</stringProp>

<stringProp name="WebServiceSampler.read_response">true</stringProp>

<stringProp name="WebServiceSampler.use_proxy">false</stringProp>

<stringProp name="WebServiceSampler.proxy_host"/>

<stringProp name="WebServiceSampler.proxy_port"/>

</WebServiceSampler>

<hashTree/>

<WebServiceSampler guiclass="WebServiceSamplerGui" testclass="WebServiceSampler" testname="ea :: GetSupportedHardware" enabled="true">

<elementProp name="HTTPsampler.Arguments" elementType="Arguments">

<collectionProp name="Arguments.arguments"/>

</elementProp>

<stringProp name="HTTPSampler.domain">keep.wpakb.kb.nl</stringProp>

<stringProp name="HTTPSampler.port">9001</stringProp>

<stringProp name="HTTPSampler.protocol">http</stringProp>

<stringProp name="HTTPSampler.path">/emulatorarchive/</stringProp>

<stringProp name="WebserviceSampler.wsdl_url">http://keep.wpakb.kb.nl:9001/emulatorarchive?wsdl</stringProp>

<stringProp name="HTTPSampler.method">POST</stringProp>

<stringProp name="Soap.Action">urn:GetSupportedHardware</stringProp>

<stringProp name="HTTPSamper.xml_data">

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:emul="http://emulatorarchive.keep.eu"> <soapenv:Header/> <soapenv:Body> <emul:dummyElement_2/> </soapenv:Body> </soapenv:Envelope>

</stringProp>

<stringProp name="WebServiceSampler.xml_data_file"/>

<stringProp name="WebServiceSampler.xml_path_loc"/>

<stringProp name="WebserviceSampler.timeout"/>

<stringProp name="WebServiceSampler.memory_cache">true</stringProp>

<stringProp name="WebServiceSampler.read_response">true</stringProp>

<stringProp name="WebServiceSampler.use_proxy">false</stringProp>

<stringProp name="WebServiceSampler.proxy_host"/>

<stringProp name="WebServiceSampler.proxy_port"/>

</WebServiceSampler>

<hashTree/>

<WebServiceSampler guiclass="WebServiceSamplerGui" testclass="WebServiceSampler" testname="swa :: DownloadSoftware" enabled="true">

<elementProp name="HTTPsampler.Arguments" elementType="Arguments">

<collectionProp name="Arguments.arguments"/>

</elementProp>

<stringProp name="HTTPSampler.domain">keep.wpakb.kb.nl</stringProp>

<stringProp name="HTTPSampler.port">9000</stringProp>

<stringProp name="HTTPSampler.protocol">http</stringProp>

<stringProp name="HTTPSampler.path">/softwarearchive/</stringProp>

<stringProp name="WebserviceSampler.wsdl_url">http://keep.wpakb.kb.nl:9000/softwarearchive?wsdl</stringProp>

<stringProp name="HTTPSampler.method">POST</stringProp>

<stringProp name="Soap.Action">urn:DownloadSoftware</stringProp>

<stringProp name="HTTPSamper.xml_data">

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sof="http://softwarearchive.keep.eu"> <soapenv:Header/> <soapenv:Body> <sof:softwareID_2>IMG-1001</sof:softwareID_2> </soapenv:Body> </soapenv:Envelope>

```
</stringProp>
<stringProp name="WebServiceSampler.xml_data_file"/>
<stringProp name="WebServiceSampler.xml_path_loc"/>
<stringProp name="WebserviceSampler.timeout"/>
<stringProp name="WebServiceSampler.memory_cache">true</stringProp>
<stringProp name="WebServiceSampler.read_response">true</stringProp>
<stringProp name="WebServiceSampler.use_proxy">false</stringProp>
<stringProp name="WebServiceSampler.proxy_host"/>
<stringProp name="WebServiceSampler.proxy_port"/>
</WebServiceSampler>
<hashTree/>
<WebServiceSampler guiclass="WebServiceSamplerGui" testclass="WebServiceSampler" testname="swa :: GetAllSoftwarePackagesInfo" enabled="true">
<elementProp name="HTTPsampler.Arguments" elementType="Arguments">
<collectionProp name="Arguments.arguments"/>
</elementProp>
<stringProp name="HTTPSampler.domain">keep.wpakb.kb.nl</stringProp>
<stringProp name="HTTPSampler.port">9000</stringProp>
<stringProp name="HTTPSampler.protocol">http</stringProp>
<stringProp name="HTTPSampler.path">/softwarearchive/</stringProp>
<stringProp name="WebserviceSampler.wsdl_url">http://keep.wpakb.kb.nl:9000/softwarearchive?wsdl</stringProp>
<stringProp name="HTTPSampler.method">POST</stringProp>
<stringProp name="Soap.Action">urn:GetAllSoftwarePackagesInfo</stringProp>
<stringProp name="HTTPSamper.xml_data">
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sof="http://softwarearchive.keep.eu"> <soapenv:Header/> <soapenv:Body> <sof:dummyElement_1>1</sof:dummyElement_1> </soapenv:Body> </soapenv:Envelope>
</stringProp>
<stringProp name="WebServiceSampler.xml_data_file"/>
<stringProp name="WebServiceSampler.xml_path_loc"/>
<stringProp name="WebserviceSampler.timeout"/>
<stringProp name="WebServiceSampler.memory_cache">true</stringProp>
<stringProp name="WebServiceSampler.read_response">true</stringProp>
<stringProp name="WebServiceSampler.use_proxy">false</stringProp>
<stringProp name="WebServiceSampler.proxy_host"/>
<stringProp name="WebServiceSampler.proxy_port"/>
</WebServiceSampler>
<hashTree/>
<WebServiceSampler guiclass="WebServiceSamplerGui" testclass="WebServiceSampler" testname="swa :: GetPathwaysByFileFormat" enabled="true">
<elementProp name="HTTPsampler.Arguments" elementType="Arguments">
<collectionProp name="Arguments.arguments"/>
</elementProp>
<stringProp name="HTTPSampler.domain">keep.wpakb.kb.nl</stringProp>
<stringProp name="HTTPSampler.port">9000</stringProp>
<stringProp name="HTTPSampler.protocol">http</stringProp>
<stringProp name="HTTPSampler.path">/softwarearchive/</stringProp>
<stringProp name="WebserviceSampler.wsdl_url">http://keep.wpakb.kb.nl:9000/softwarearchive?wsdl</stringProp>
<stringProp name="HTTPSampler.method">POST</stringProp>
<stringProp name="Soap.Action">urn:GetPathwaysByFileFormat</stringProp>
<stringProp name="HTTPSamper.xml_data">
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sof="http://softwarearchive.keep.eu"> <soapenv:Header/> <soapenv:Body> <sof:fileFormat>ISO 9660 CD-ROM</sof:fileFormat> </soapenv:Body> </soapenv:Envelope>
</stringProp>
<stringProp name="WebServiceSampler.xml_data_file"/>
<stringProp name="WebServiceSampler.xml_path_loc"/>
<stringProp name="WebserviceSampler.timeout"/>
<stringProp name="WebServiceSampler.memory_cache">true</stringProp>
<stringProp name="WebServiceSampler.read_response">true</stringProp>
<stringProp name="WebServiceSampler.use_proxy">false</stringProp>
```


```
<stringProp name="WebServiceSampler.proxy_host"/>
<stringProp name="WebServiceSampler.proxy_port"/>
</WebServiceSampler>
<hashTree/>
<WebServiceSampler guiclass="WebServiceSamplerGui" testclass="WebServiceSampler" testname="swa :: GetSoftwarePackageInfo" enabled="true">
<elementProp name="HTTPsampler.Arguments" elementType="Arguments">
<collectionProp name="Arguments.arguments"/>
</elementProp>
<stringProp name="HTTPSampler.domain">keep.wpakb.kb.nl</stringProp>
<stringProp name="HTTPSampler.port">9000</stringProp>
<stringProp name="HTTPSampler.protocol">http</stringProp>
<stringProp name="HTTPSampler.path">/softwarearchive/</stringProp>
<stringProp name="WebserviceSampler.wsdl_url">http://keep.wpakb.kb.nl:9000/softwarearchive?wsdl</stringProp>
<stringProp name="HTTPSampler.method">POST</stringProp>
<stringProp name="Soap.Action">urn:GetSoftwarePackageInfo</stringProp>
<stringProp name="HTTPSamper.xml_data">
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sof="http://softwarearchive.keep.eu"> <soapenv:Header/> <soapenv:Body> <sof:softwareID_1>IMG-1001</sof:softwareID_1> </soapenv:Body> </soapenv:Envelope>
</stringProp>
<stringProp name="WebServiceSampler.xml_data_file"/>
<stringProp name="WebServiceSampler.xml_path_loc"/>
<stringProp name="WebserviceSampler.timeout"/>
<stringProp name="WebServiceSampler.memory_cache">true</stringProp>
<stringProp name="WebServiceSampler.read_response">true</stringProp>
<stringProp name="WebServiceSampler.use_proxy">false</stringProp>
<stringProp name="WebServiceSampler.proxy_host"/>
<stringProp name="WebServiceSampler.proxy_port"/>
</WebServiceSampler>
<hashTree/>
<ResultCollector guiclass="GraphVisualizer" testclass="ResultCollector" testname="Graph Results" enabled="true">
<boolProp name="ResultCollector.error_logging">false</boolProp>
<objProp>
<name>saveConfig</name>
<value class="SampleSaveConfiguration">
<time>true</time>
<latency>true</latency>
<timestamp>true</timestamp>
<success>true</success>
<label>true</label>
<code>true</code>
<message>true</message>
<threadName>true</threadName>
<dataType>true</dataType>
<encoding>false</encoding>
<assertions>true</assertions>
<subresults>true</subresults>
<responseData>false</responseData>
<samplerData>false</samplerData>
<xml>true</xml>
<fieldNames>false</fieldNames>
<responseHeaders>false</responseHeaders>
<requestHeaders>false</requestHeaders>
<responseDataOnError>false</responseDataOnError>
<saveAssertionResultsFailureMessage>false</saveAssertionResultsFailureMessage>
<assertionsResultsToSave>0</assertionsResultsToSave>
<bytes>true</bytes>
</value>
</objProp>
```

```
<stringProp name="filename">C:\Apps\jmeter\bin\data.csv</stringProp>
</ResultCollector>
<hashTree/>
</hashTree>
</hashTree>
</hashTree>
</jmeterTestPlan>
```

Appendix B: list of EF 1.0.0 system functions (methods)

autoSelectEmulator(List<EmulatorPackage> emuPacks) Select an emulator automatically from a list of emulators The selection process picks the first encountered emulator that can run on the current host system
autoSelectFormat(List<Format> formats) Select a format from a list of formats.
autoSelectPathway(List<Pathway> pathways) Select a valid pathway automatically from a list of potential pathways The selection process simply picks the first encountered satisfiable pathway
autoSelectSoftwareImage(List<SoftwarePackage> swPacks) Select a software image automatically from a list of software images.
characterise(File digObj) Characterise a digital object and returns information on format names, mime types and the reporting tools.
cleanUp() Clean up any temporary files and directories that were created by the Core Engine to unpack files, run emulators, etc.
extractPathwayFromFile(File metadataFile) Retrieve the technical environment, i.e.
getCoreSettings() Get the Core Engine settings
getEmuConfig(Integer conf) Get the configuration map of all available emulator parameters Useful for manual configuration of the emulator, to be used with setEmuConfig()
getEmulatorsByPathway(Pathway pathway) Returns a list of supported emulators that satisfy a given pathway.
getEmuListFromArchive() Get the list of all emulator packages available in the Emulator Archive
getEmusByHWFromArchive(String hardwareName) Get the list of emulator packages that support a hardware type in the emulator archive
getFileInfo(File digObj) Characterise a digital object and returns file information
getPathways(Format format) Get pathways for a given file formatName.
getRegistries() Retrieve the list of technical registries
getSoftwareByPathway(Pathway pathway) Returns a list of supported software packages that satisfy a given pathway.

getSoftwareListFromArchive() Get all software packages available in the software archive
getSupportedHardwareFromArchive() Get the list of hardware supported by the Emulator Archive
getTechMetadata(File digObj) Characterise a digital object and returns technical metadata information
getTitle() Get the Emulation Framework title from the jar manifest
getVendor() Get the Emulation Framework vendor from the jar manifest
getVersion() Get the Emulation Framework version from the jar manifest
getWhitelistedEmus() Select the whitelisted emulator IDs from the local database
isPathwaySatisfiable(Pathway pathway) Checks if a given pathway is satisfiable given the available emulators and software images
matchEmulatorWithSoftware(Pathway pathway) Match emulators with a list of associated software images from a given pathway
prepareConfiguration(File digObj, EmulatorPackage emuPack, SoftwarePackage swPack, Pathway pathway) Prepares the configuration settings for the selected emulation process The resulting configuration (emulator options) can be edited using setEmuOptions() and getEmuOptions()
registerObserver(CoreObserver coreObs) Register an observer
removeObserver(CoreObserver coreObs) Remove an observer
runEmulationProcess(Integer conf) Run the chosen emulation process An emulator must have already been selected and its configuration settings properly prepared.
setEmuConfig(Map<String,List<Map<String,String>>> options, Integer conf) Set the emulator parameters Useful for manual configuration of the emulator to be used with getEmuConfig()
setRegistries(List<DBRegistry> listReg) Insert registry information from list into the local database This replaces all existing registry information with the contents of the list
start(File file) Launches the emulation process automatically (i.e.
start(File file, File metadata)

Launches the emulation process automatically (i.e.
start(File file, List<Pathway> pathways) Launches the emulation process given a digital object and a list of pathways to select from.
start(File file, Pathway pathway) Launches the emulation process given a digital object and a specific pathway.
stop() Stop the Core Emulator Framework engine
unListEmulator(Integer i) Removes an emulator ID from the whitelist in the local database (list of emulators that will be used for rendering a digital object)
whiteListEmulator(Integer i) Adds an emulator ID to the whitelist in the local database (list of emulators that will be used for rendering a digital object)