Keeping Emulation Environments Portable

FP7-ICT-231954

System Maintenance Guide for the Emulation Framework version 1.1.0 (December 2011)

Release date	21 December 2011
Release version	1.1.0
Author(s)	Bram Lohman (Tessella) David Michel (Tessella) Edo Noordermeer (Tessella)
Organisation	KEEP project

Executive Summary

This is the System Maintenance Guide (SMG) for the Emulation Framework. The EF is software developed by the international KEEP project, co-funded by the European Unions 7th Framework Programme.

The System Maintenance Guide outlines how to build and maintain the system, and how to set up the development environment.

Developed in Java, the system is by definition cross-platform and can therefore be developed on any platform that supports Java. An Ant build script is provided to perform all necessary build tasks related to the project, such as compiling the source code, launching the unit test suite, setting up the database, running static analysis tools or building a release package.

The Emulation Framework has been developed as a library, and as such is intended for use by an external system; it doesn't constitute a stand-alone product on its own. However, for development and demonstration purposes, two access methods have been developed: a built-in shell that allows direct access to the public Application Programming Interface (API); and a Graphical User Interface (GUI). A list and description of the available commands for the shell is included in this document.

The Emulation Framework has three main external dependencies: an Emulator Archive, a Software Archive and a technical metadata registry. The Emulator Archive is used to access (certified) Emulator Packages. The Software Archive provides software images of operating systems and applications that the emulators require to render the environment. Finally, the technical metadata registry is required for retrieving information about which computer platform dependencies exist for digital objects (e.g. WordPerfect documents require the application WordPerfect, operating system MS DOS and an x86 PC or compatible architecture. For each of these dependencies, a simple prototype has been created to be able to fully demonstrate the Emulation Framework. The registry prototype is incorporated in the Software Archive prototype.

Several objects used within the framework, such as Emulator Packages, Software Packages, and Pathways make use of XML schemas describing their properties. For each of these objects, the relevant XML schema is described and a sample file is included.

Examples are provided showing how to employ the basic functionality of the framework when running from the Command Line Interface.

Table of Contents

1 Introduction

1.1 Purpose and scope

This document provides information about how the Emulation Framework (EF) is constructed, maintained and deployed. It is intended for developers and, to a limited extent, system administrators who need to maintain the software. It does not describe how to use or install the system; this is covered by the EF System User Guide [SUG].

This document covers the core software, internal database and supporting objects that make up the Emulation Framework application. The aim of this document is to aid in developing, installing and maintaining the application.

Although this document gives an outline of the inner workings of the application, as well as the interfaces to external systems, it does not cover details of their setup or maintenance. Neither is this a detailed guide to the overall structure of the EF – this can be found in the EF Architectural Design Document [ADD].

1.2 Context of this Issue

This version of the SMG describes **version 1.1.0 of the Emulation Framework**.

This document describes the Emulation Framework environment that has been released in December 2011. This includes two prototype archives (Emulator and Software Archive), some sample test data, the Emulation Framework, and a GUI.

1.3 About the KEEP project

KEEP (Keeping Emulation Environments Portable) is a 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 nine organisations representing a wide range of stakeholders in Europe: cultural heritage institutes , research institutes, commercial partners and the gaming industry. The project has a duration of three years and ends February 2012.

More information can be found on the KEEP website: http://www.keep-project.eu

1.4 About the software in this release

The EF software is divided into **Core, Software Archive** and **Emulator Archive** components. The Core is the technical heart of the system, performing the automatic identification of file formats, selecting the required software and automatically configuring the emulation environment. It has a simple GUI to interact directly with the user. For selecting the software and emulator, the Core interacts with external services such as technical registries containing file format classifications, the Software Archive that contains disk images and the Emulator Archive that contains the emulators available for the EF.

The Core, Software Archive and Emulator Archive are developed by Tessella with support from the National Library of the Netherlands. The Core GUI is developed by the National Library of the Netherlands.

2 System Context and Interfaces

2.1 Overview and Context

The following diagram is taken from the URD. It shows the context and boundaries of the Emulation Framework. The task of the EF is to provide users access to digital objects of any kind via emulation.

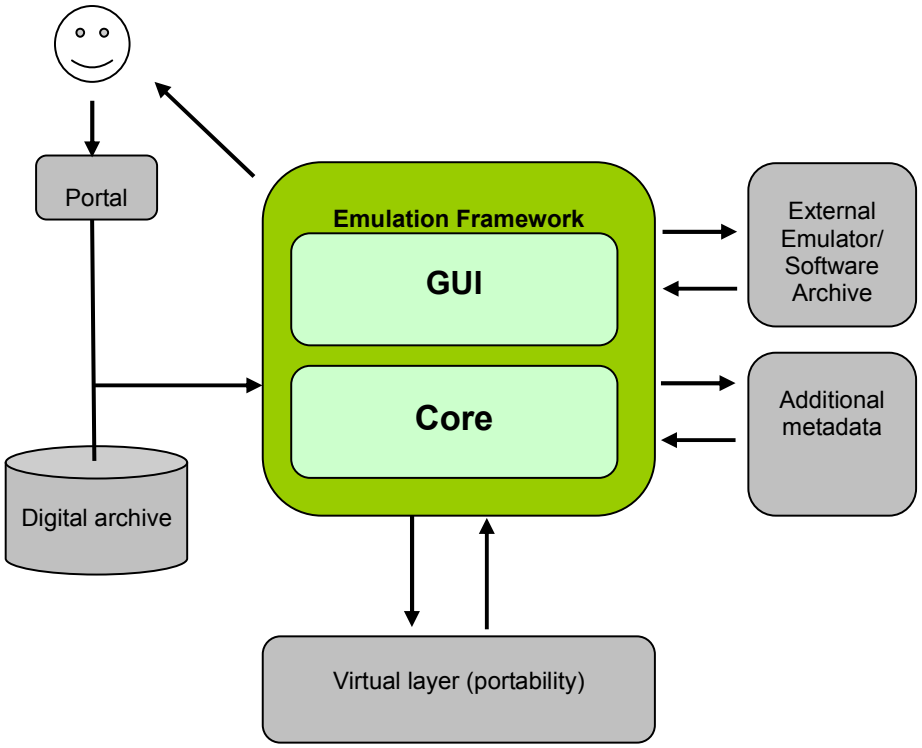


Figure 1 : EF system overview and system boundaries

Figure 1 shows a high-level overview of the system (green, in scope) and its boundaries (grey, outside system scope). Within the EF a distinction is made in *Core* and *GUI*. The Core is responsible for managing emulation processes while the GUI provides a rendering environment plus additional services to the user.

3 Configuration

The **Core**, **Emulator Archive** and **Software Archive** components can be configured using text-based property-files.

3.1 Core configuration

The configuration of the Core is stored in a file called **user.properties**, located in the ./eu/keep subdirectory, and contains the internal database connection properties, the location of the Emulator and Software Archive, as well as the various directories used by the system.

Property key	Default property value	Comment
h2.db.driver	org.h2.Driver	JDBC driver class
h2.jdbc.prefix	jdbc:h2:	JDBC url prefix
h2.db.url	./database/h2/EF_engine	Database location on disk
h2.db.exists	;IFEXISTS=TRUE	Flag to indicate whether the database should be created (FALSE), or fail (TRUE) if it doesn't exist.
h2.db.server	;AUTO_SERVER=TRUE	The database connection type
h2.db.schema	;SCHEMA=engine	Name of the schema used for the internal database
h2.db.admin	sa	Database admin login name
h2.db.adminpassw	CEF_Engine	Database admin password
h2.db.user	cef	Database user login name
h2.db.userpassw	cef	Database user password
software.archive.url	http://localhost:9000/softwarearchive/	Location of the Software Archive
emulator.archive.url	http://localhost:9001/emulatorarchive/	Location of the Emulator Archive
exec.dir	./exec	Temporary directory where the emulators are installed for use (will be deleted after use)
system.tmpdir	./tmp	Directory used by the host system to store temporary files
accepted.languages[1]	all	Emulator and Software languages which are accepted. Emulators or Software which use other languages will not be selected or displayed. Languages are entered as a comma-separated list of 2-letter 'locale'-codes, e.g. en (English), nl (Dutch), fr (French), de (German). See appendix B for a complete list of language codes. To accept all available languages, enter 'all'.

The GUI has a separate configuration file called **gui.properties,** also located in the ./eu/keep subdirectory.

This file contains the database connection parameters for the different components; depending on whether the Emulation Framework was installed in 'client' or 'server' mode, this may be just the EF, or the EF, Emulator Archive and Software Archive.

[1] The list of accepted languages can also be configured via the EF GUI (see section 2.6 in the System User Guide).

Property key	Default property value	Comment
ef.db.driver	org.h2.Driver	JDBC driver class
ef.jdbc.prefix	jdbc:h2:	JDBC url prefix
ef.db.url	./database/h2/EF_engine	Database location on disk
ef.db.exists	;IFEXISTS=TRUE	Flag to indicate whether the database should be created if it doesn't exists (FALSE), or fail if it doesn't exist (TRUE)
ef.db.server	;AUTO_SERVER=TRUE	Database connection type
ef.db.schema.name	engine	Name of the schema used for the internal database
ef.db.schema	;SCHEMA=engine	Name of the schema used for the internal database
ef.db.admin	sa	Database admin login name
ef.db.adminpassw	CEF_Engine	Database admin password
ef.db.user	cef	Database user login name
ef.db.userpassw	cef	Database user password
ea.db.driver	org.h2.Driver	JDBC driver class
ea.jdbc.prefix	jdbc:h2:	JDBC url prefix
ea.db.url	./ea/database//EF_ea	Database location on disk
ea.db.exists	;IFEXISTS=TRUE	Flag to indicate whether the database should be created if it doesn't exists (FALSE), or fail if it doesn't exist (TRUE)
ea.db.server	;AUTO_SERVER=TRUE	Database connection type
ea.db.schema.name	emulatorarchive	Name of the schema used for the internal database
ea.db.schema	;SCHEMA=emulatorarchive	Name of the schema used for the internal database
ea.db.admin	sa	Database admin login name
ea.db.adminpassw	EA_Engine	Database admin password
ea.db.user	ea	Database user login name
ea.db.userpassw	ea	Database user password
swa.db.driver	org.h2.Driver	JDBC driver class
swa.jdbc.prefix	jdbc:h2:	JDBC url prefix
swa.db.url	./database/h2/EF_swa	Database location on disk
swa.db.exists	;IFEXISTS=TRUE	Flag to indicate whether the database should be created if it doesn't exists (FALSE), or fail if it doesn't exist (TRUE)
swa.db.server	;AUTO_SERVER=TRUE	Database connection type
swa.db.schema.name	softwarearchive	Name of the schema used for the internal database
swa.db.schema	;SCHEMA=softwarearchive	Name of the schema used for the internal database
swa.db.admin	sa	Database admin login name
swa.db.adminpassw	SWA_Engine	Database admin password
swa.db.user	swa	Database user login name
swa.db.userpassw	swa	Database user password

3.2 Emulator Archive configuration

Similar to the Core, the Emulator Archive also has a **user.properties** file for configuration. It is located in the ./ea subdirectory and it contains the emulator database connection properties, and the location of the Emulator Archive.

The table below outlines the configuration properties and their defaults.

Property key	Default property value	Comment
h2.db.driver	org.h2.Driver	JDBC driver class
h2.jdbc.prefix	jdbc:h2:	JDBC url prefix
h2.db.url	./ea/database//EF_ea	Database location on disk
h2.db.exists	;IFEXISTS=TRUE	Flag to indicate whether the database should be created (FALSE), or fail (TRUE) if it doesn't exist.
h2.db.schema	;SCHEMA=emulatorarchive	Name of the schema used for the internal database
h2.db.admin	sa	Database admin login name
h2.db.adminpassw	EA_Engine	Database admin password
h2.db.user	ea	Database user login name
h2.db.userpassw	ea	Database user password
server.soap.address	http://localhost:9001/emulatorarchive/	webservice Server address

Note that the server.soap.address property must match the EF emulator.archive.url property for the connection between the two components to be successful.

3.3 Software Archive configuration

Similar to the Core, the Software Archive also has a **user.properties** file for configuration. It is located in the ./swa subdirectory and it contains the software database connection properties, and the location of the Software Archive.

The table below outlines the configuration properties and their defaults.

Property key	Default property value	Comment
swa.db.driver	org.h2.Driver	JDBC driver class
swa.jdbc.prefix	jdbc:h2:	JDBC url prefix
swa.db.url	./database/h2/EF_swa	Database location on disk
swa.db.exists	;IFEXISTS=TRUE	Flag to indicate whether the database should be created (FALSE), or fail (TRUE) if it doesn't exist.
swa.db.schema	;SCHEMA=softwarearchive	Name of the schema used for the internal database
swa.db.admin	sa	Database admin login name
swa.db.adminpassw	SWA_Engine	Database admin password
swa.db.user	swa	Database user login name
swa.db.userpassw	swa	Database user password
server.soap.address	http://localhost:9000/softwarearchive/	Server address

Note that the server.soap.address property must match the EF software.archive.url property for the connection between the two components to be successful.

4 The Development Environment

The Emulation Framework makes use of standard tools, and therefore no specific development environment is required. The following tools are used to access, build and develop the project:

- Subversion (SVN) Source code revision control system
- Sun Java 1.6 Java Development Kit (JDK)
- Sun Java 1.6 Java Runtime Environment (JRE)
- Apache Ant 1.7.x build system[2]
- Apache Ivy 2.x.x dependency manager[3]
- H2 DBMS engine

All these tools are open-source and freely obtainable. It is recommended to use the version described above.

4.1 Source files

The EF code is hosted in three Subversion (SVN) repositories, available at:

http://emuframework.svn.sourceforge.net/viewvc/emuframework/Core/

http://emuframework.svn.sourceforge.net/viewvc/emuframework/EmulatorArchive/

http://emuframework.svn.sourceforge.net/viewvc/emuframework/SoftwareArchive/

Note: this SVN repository can be accessed (read-only) by anyone, but requires authentication for committing (uploading) files.

The 'trunk' contains the main (current) development branch. There is a 'branches' directory that contains the different versions used during development of experimental changes/modifications. The 'tag' directory contains, as its name indicates, the various tagged copies of the trunk corresponding to a particular 'frozen' version.

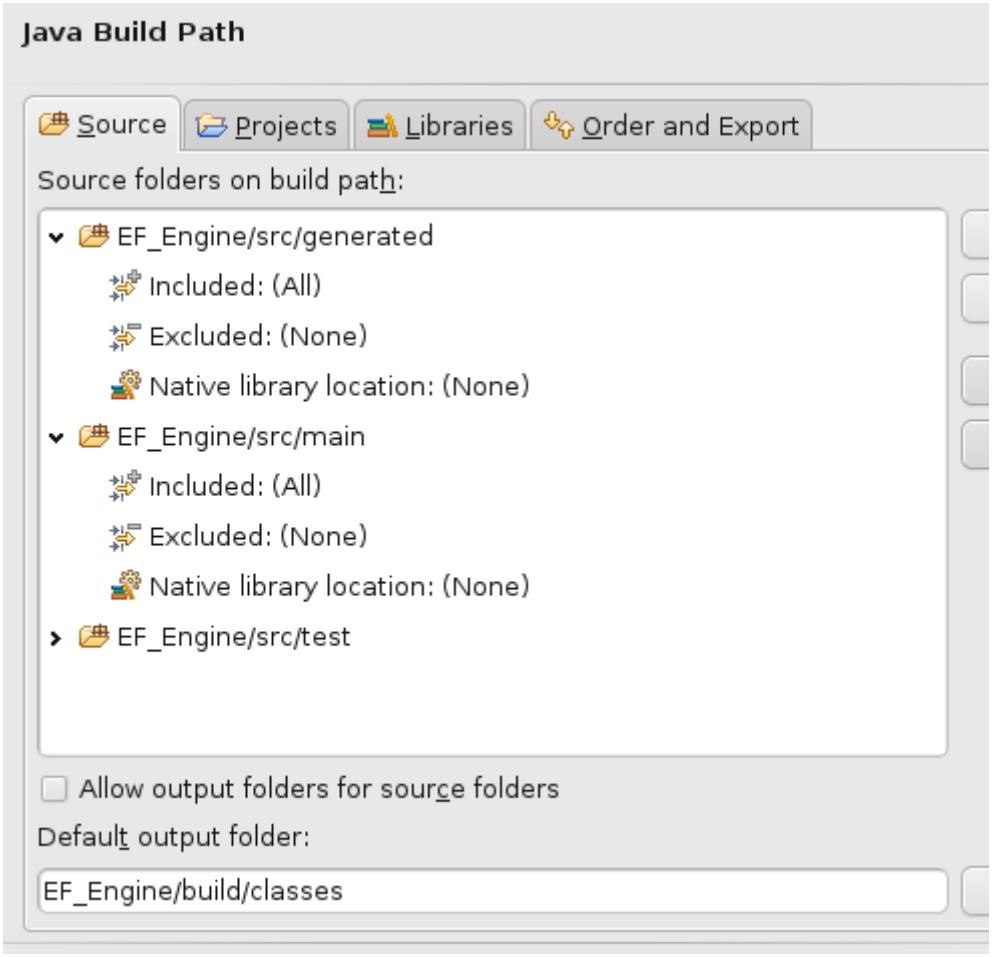
4.2 Example development setup

A common Java software development platform is Eclipse[4]. Below are the steps to set up the project in Eclipse.

[2] http://ant.apache.org/

[3] http://ant.apache.org/ivy/

[4] http://www.eclipse.org/



Each component should be checked out as a separate project.

The image shows the directories to add as source directories, along with the default output folder as specified in the Ant script.

The relevant libraries – the JAR files from the 'lib/' directory – will need to be added to the build path[5].

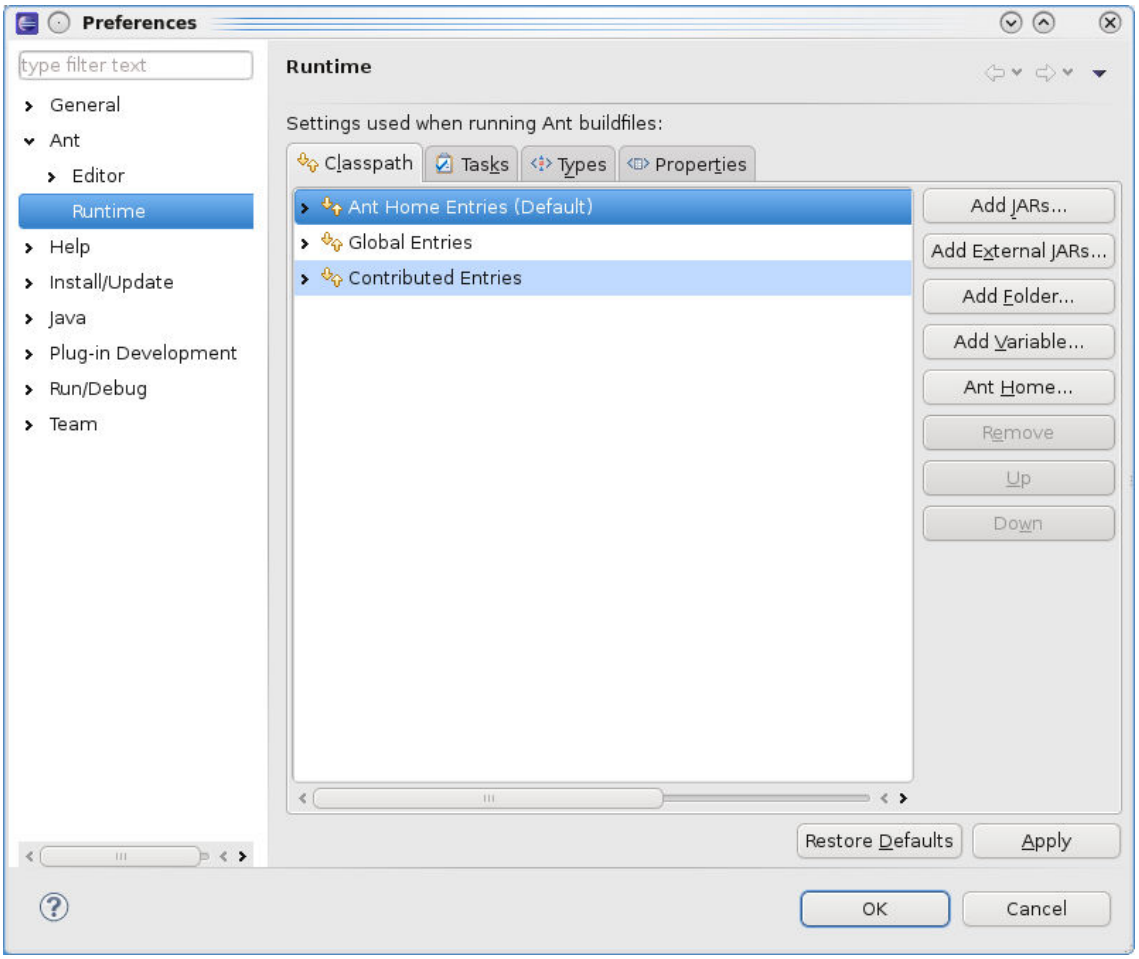
4.3 Managing dependencies

The EF uses Ivy to manage the jar-dependencies.

To use Ivy, Apache Ant has to be set up to support it. The Ivy jar file (ivy-n.n.n.jar e.g. ivy-2.2.0.jar), is located in the ‘./lib-local’ directory.

In a development environment, e.g. Eclipse, the Ivy jar has to be added to the Ant classpath. This can be done via Window -> Preferences -> Ant -> Runtime (Classpath -> Add External JARs).

[5] Note: these libraries are downloaded using the Ivy dependency manager, and may not exist until the relevant action is performed. See section □



Make sure that the development environment uses the same version of Ant as required by the Ivy jar.

Note: Ant needs several jars to run the build script properly; these jars are kept locally in the './lib-local/ant' directory. Also, not all jars are published on Maven (e.g. DROID, FITS, etc.). For these, a local copy is kept in './lib-local/external'. The ivy-external.xml file is defined which retrieves these jars. The main ivy.xml[6] references these jars as a normal dependency.

To download the required jars for the EF, run the following ant-targets in Core:

ivy-publish-external
ivy-retrieve

The downloaded jars are stored in Core/lib/dev/, Core/lib/dist/ and Core/lib/fits/.

4.4 Build system

It is recommended to use the provided Ant build script (**build.xml**) to compile, build and test the code. Ant is cross-platform and independent of the development environment.

The **build.xml** file relies on a **build.xml.common** file for certain macros used in the targets. The Ivy targets also rely on the **ivy.xml**, **ivy-external.xml** and **ivysettings.xml** files. A selection of common Ant targets is listed in Table 1. A comprehensive set can be found in Appendix A.

[6] See http://mvnrepository.com/ for Apache Ivy dependency links

Table 1: Selection of common Ant script targets

Ant target	Comment
compile	Compiles the code base
clean	Deletes output files and directories created during a build, i.e. *./build*, *./src/generated/*
db.create	Creates and populates the internal database
db.drop	Deletes the database
generated.src	Generates source code from WSDL/XSD files
ivy-publish	Publish the Core jar to the repository
jar	Creates a JAR
javadoc	Runs the javadoc, document generator for Java source code
release	Creates a release package for the Core project
release.installer	Creates a release package for the Core, Emulator Archive and Software Archive using IzPack. Requires the Emulator and Software Archive to be available and build
test.run	Prepares and runs the unit tests

4.5 Auto-generating required source files

The development environment requires several auto-generated files for it to run correctly. These are generated by Apache CXF and placed by default in the src/generated directory. Apache CXF uses the following input files to generate Java code:

- ./resources/external/softwarearchive/softwarearchive.wsdl
- ./resources/external/softwarearchive/PathwaySchema.xsd
- ./resources/external/softwarearchive/SoftwarePackageSchema.xsd
- ./resources/external/emulatorarchive/emulatorarchive.wsdl
- ./resources/external/emulatorarchive/EmulatorPackageSchema.xsd

The Ant target generated.src will run the necessary code to auto-generate the required files.

4.6 Setting up the internal database

The Core EF uses an internal database to store metadata information. The Subversion repository includes a database that is configured for use, but the Ant script provides targets to re-generate a database. The targets starting with 'db.*' can be used to generate this database.

The database used is H2, a Java based database with a small footprint and an integrated browser-based database viewer.

The viewer can be started by running the H2 library; it should automatically open a browser with the log-in screen. See section 6 for detailed information on the database.

4.7 Building the Emulation Framework JAR

Given the Ant build script, it is very easy to build core by simply running the *jar* target which will generate the compiled class files and the JAR in the *build* folder.

4.8 Creating a release package

A release package can be easily created by calling the *release.installer* target in the Ant script. This will call IzPack[7] which in turns uses *./resources/release/install.xml* to configure the installation package.

[7] IzPack website, available at: http://izpack.org/

5 Emulation Framework dependencies

The Emulation Framework relies on several components to successfully render an environment. In this section, these dependencies and their configuration are described.

5.1 Technical registry

The EF may use technical registries to retrieve technical information about file format and platform dependencies such as required operating system, applications, drivers, etc.

Currently, no such technical registry is operational and openly available, although a proof of concept has been shown to work with PRONOM. As such, the Software Archive contains simple metadata to generate Pathways.

For external registries within the EF, each technical registry has its own class file. For example:

- eu.keep.registry.UDFRRegistry
- eu.keep.registry.PronomRegistry

The information about these registries and their metadata is stored in the Software Archive database.

5.2 Emulator Archive

The EF uses emulators to render the environment. To organise the available emulators that are found compatible with the EF, an Emulator Archive has been created. This archive runs as a separate web-service, which the EF can access as a client. The server-client interaction is achieved via web services (Apache CXF library) and uses a WSDL as an interface definition. The Emulator Archive also defines an EmulatorPackage object using XSD. Both files, located in **Core/trunk/resources/external/emulatorarchive/,** are linked to the Emulator Archive repository:

EmulatorArchive/trunk/resources/emulatorarchive.wsdl
EmulatorArchive/trunk/resources/EmulatorPackageSchema.xsd

For details of these schemas, please see section 7.

Since the framework doesn't hold any emulators locally, it depends on the Emulator Archive to supply these. As the emulation process is being configured, the Emulator Archive server will be contacted for the appropriate emulator that can satisfy the selected emulation Pathway.

The Emulator Archive is contained as a separate project in the Emulation Framework.

5.3 Software Archive

A Software Archive has been created to manage the software required by the emulators. Similar to the Emulator Archive, the Software Archive runs as a separate web service, with the EF as a client. The server-client interaction is achieved via web-services (Apache CXF library). The Software Archive also defines Pathway and SoftwarePackage objects using XSD. All three files, located in **Core/trunk/resources/external/softwarearchive/,** are linked to the Software Archive repository:

SoftwareArchive/trunk/resources/softwarearchive.wsdl
SoftwareArchive/trunk/resources/SoftwarePackageSchema.xsd
SoftwareArchive/trunk/resources/PathwaySchema.xsd

For details of these schemas, please see section 7.

Since the framework doesn't hold any software images locally, it depends on the Software Archive to supply these. As the emulation process is being configured, the Software Archive server will be contacted for the appropriate software image that can satisfy the selected emulation Pathway.

The Software Archive is contained as a separate project in the Emulation Framework.

6 Database management

The Core EF and the Software Archive and Emulator Archive prototypes use internal databases to store (meta)data. The database system used is H2, a Java based database with a small footprint and an integrated browser-based database viewer (see http://www.h2database.com). An administrator can directly access this database (without the Emulation Framework) by using this viewer.

Warning: manually editing the various database tables, when done incorrectly, can lead to unexpected results, including EF malfunction. Do not edit a live production database unless you are confident that you know what you are doing.

To start the viewer, simply run the h2-1.2.133.jar file, available in

EmulatorArchive/trunk/lib/
SoftwareArchive/trunk/lib/

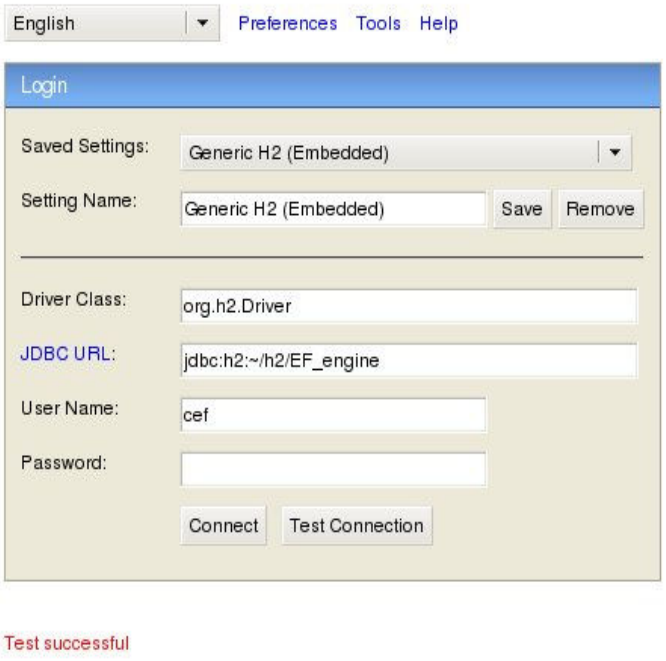
or, after running Ivy, in

Core/lib/dist

or, in the installed release directory, in

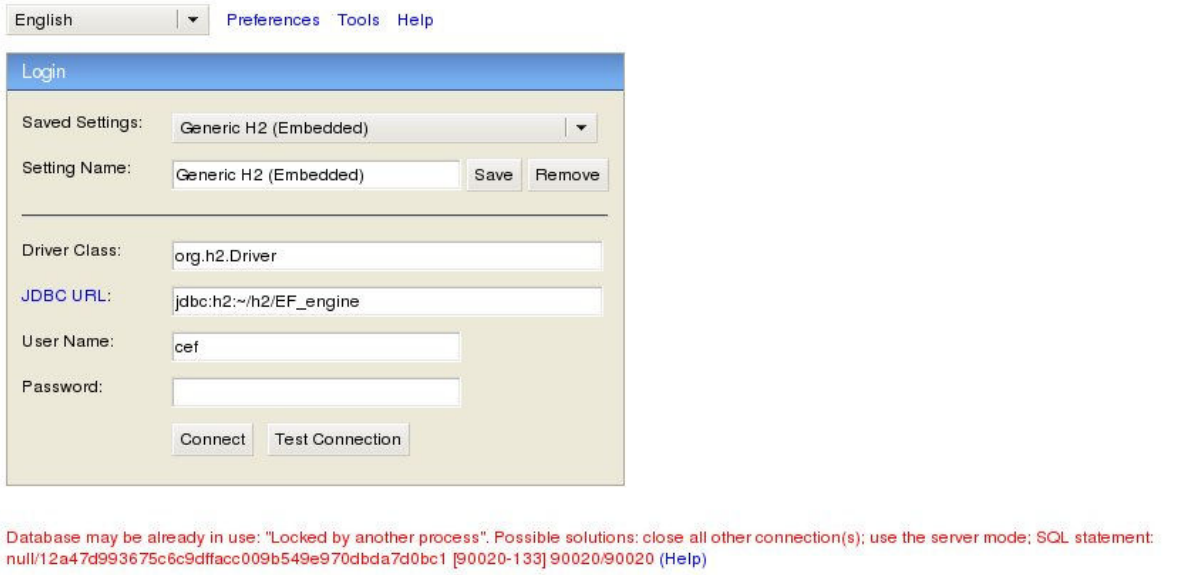
./lib/dist

It should automatically open a browser with the log-in screen:

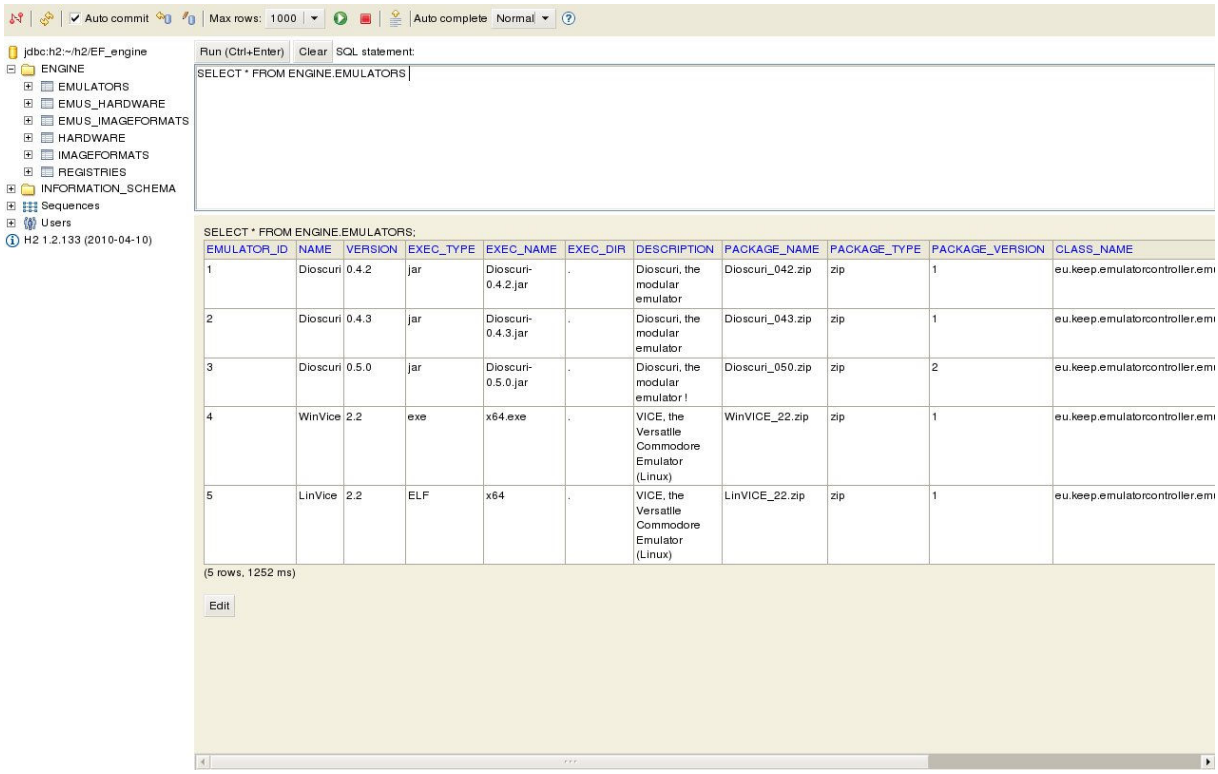


To connect to a database, enter the JDBC URL, user name and password (see following sections). The 'Test Connection' button can be used to check if the url, user name and password are correct.

If a connection to the database is already open, an error message will appear (as expected for all embedded databases where only one connection can be established at a time):



Once the connection is established, you can navigate between the various schemas/tables of the database and execute SQL statements directly.



6.1 Core embedded database management

The embedded (internal) database in the Core Emulation Framework is currently used only for maintaining the 'Emulator Whitelist': a table containing the IDs of emulators that are allowed to be run. For more details, please see the [ADD].

To connect to the EF Core database, enter the full path to file

Core/trunk/database/h2/EF_engine.h2.db (development environment) or
./database/h2/EF_engine.h2.db (installed release directory)

for the JDBC URL. Omit the .h2.db suffix.

For user name and password, enter the values given in the **Core/trunk/user.properties**.

6.2 Emulator Archive database population

The Emulator Archive database holds the binaries and metadata for the available emulators. For more details, please see the [ADD].

To connect to the Emulator Archive database, enter the full path to file
EmulatorArchive/trunk/database/db/EF_ea.h2.db (development environment) or
./ea/database/db/EF_ea.h2.db (installed release directory)

for the JDBC URL. Omit the .h2.db suffix.

For user name and password, enter the values given in the
EmulatorArchive/trunk/user.properties.

The Emulator Archive database can also be accessed via the corresponding tab in the test GUI (only available when running the EF in 'server' mode). Simple database record insertion and deletion is supported. Note that due to the size of some of the emulators, BLOBs are not shown in the GUI, and cannot be inserted. To successfully insert these records, it is suggested to use 'Add Emulator' wizard in the test GUI (see section 3.3.1 in the System [SUG]).

Although the Emulator Archive contains a number of default emulators, it can be desirable to insert other emulators. The preferred method to do this is by using the 'Add Emulator' wizard in the test GUI (see section 3.3.1 in the [SUG]). However, it is also possible to do it directly using the H2 browser interface. To successfully do this, a number of parameters relating to the emulator must be set correctly. The following example demonstrates this process

1. Add hardware
   Add a new hardware id and name to the HARDWARE table, if necessary. Note: This hardware must also be available in the Software Archive
2. Add image format
   Add a new image format id and name to the IMAGEFORMATS table, if necessary
3. Add emulator
   Add a new emulator id, name, version, etc. to the EMULATORS table. See the Architectural Design Document for a full description of the columns
4. Link the emulator to the hardware
   Add the emulator id and hardware id link in the EMUS_HARDWARE table, indicating the emulator runs on the specified hardware
5. Link the emulator to the image format
   Add the emulator id and image format id link in the EMUS_IMAGEFORMATS table, indicating the emulator runs on the specified hardware. Note: the emulator may support more than one image format.

6.3 Software Archive database population

The Software Archive database holds the binaries and metadata for the available software images. For more details, please see the [ADD].

To connect to the Emulator Archive database, enter the full path to file **SoftwareArchive/trunk/database/db/EF_swa.h2.db** (development environment) or **./swa/database/db/EF_swa.h2.db** (installed release directory)

for the JDBC URL. Omit the .h2.db suffix.

For user name and password, enter the values given in the **SoftwareArchive/trunk/user.properties**.

The Emulator Archive database can also be accessed via the corresponding tab in the test GUI (only available when running the EF in 'server' mode). Simple database record insertion and deletion is supported. Note that due to the size of some of the software images, BLOBs are not shown in the GUI, and cannot be inserted. To successfully insert these records, it is suggested to use 'Add Software wizard in the test GUI (see section 3.3.2 in the [SUG]).

Although the Software Archive contains a number of default software images, it can be desirable to insert others. To successfully do this, a number of parameters relating to the software image must be set correctly, plus a supporting pathway must be created. The following example demonstrates this process

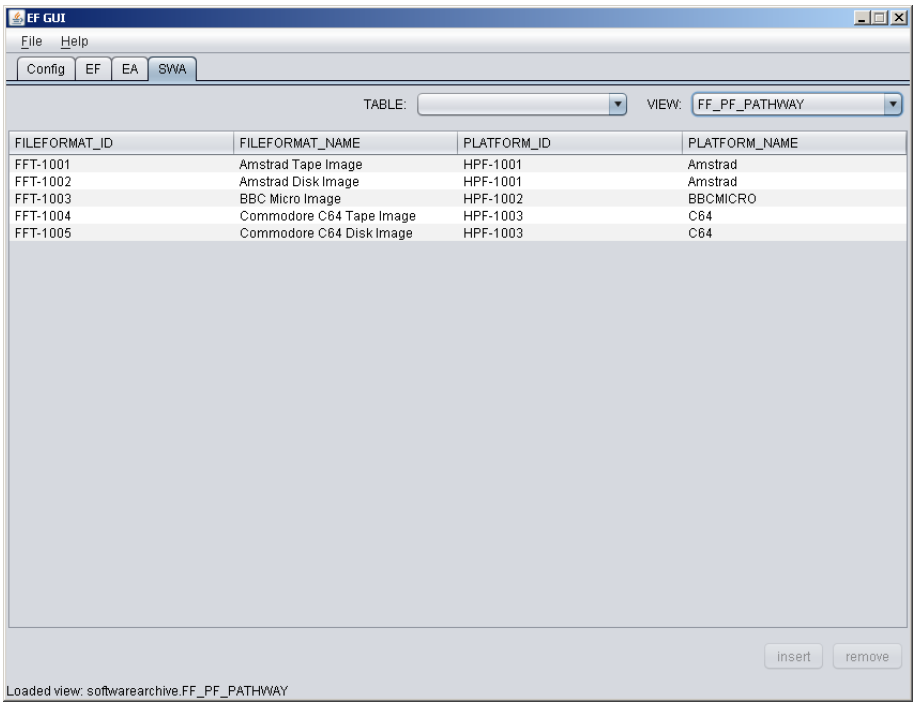
1. Add file format
   Add a new file format id and name to the FILEFORMATS table, if necessary. Note: This file format must also be available in the Core database.
2. Add application
   Add a new application id and name to the APPS table, if necessary.
3. Add operating system
   Add a new operating system id and name to the OP_SYS table, if necessary.
4. Add platform
   Add a new platform id and name to the PLATFORMS table, if necessary.
5. Add an image format
   Add a new image format id and name to the IMAGEFORMATS table, if necessary. This image format must also be available in the EA database.
6. Add a software image
   Add a software image id, name, image format id and platform id to the IMAGES table. Add the image file to the IMAGEBLOBS table.
7. Create the pathway
   a. Link the file format to an application, operating system and/or platform
      Depending on the pathway, a file format must either be run in an application (e.g. a WordPerfect file runs in the WordPerfect application), operating system (e.g. an ISO file is run directly by Windows) or a hardware platform (e.g. a C64 file runs directly on the Commodore 64). In the appropriate table (FILEFORMATS_APPS, FILEFORMATS_OPSYS, FILEFORMATS_PLATFORM), link the file format to the appropriate level in the pathway.
   b. Link the application to an operating system and software image
      An application requires an operating system to run, and the software image needs to know what applications are contained in it. The appropriate links

between the application and operating system/software image need to be made in the APPS_OPSYS/APPS_IMAGES tables, respectively.

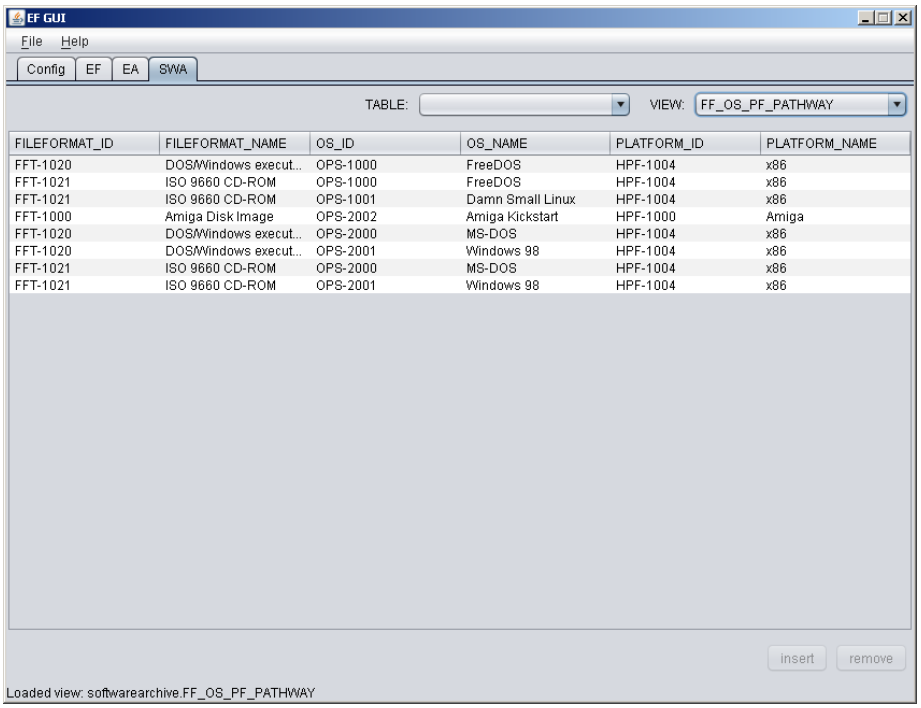
c. Link the operating system to a platform and software image
An operating system runs on a platform, and the software image needs to know what operating systems are contained in it. The appropriate links between the operating system and platform/software image need to be made in the OPSYS_PLATFORMS/OPSYS_IMAGES tables, respectively.

8. Check the pathway is correctly generated
Using the PATHWAYS view, the pathway generated in step 7 should be visible. This view contains all pathways, be they file format to platform; file format to operating system to platform; or file format to application to operating system to platform. For those pathways that do not have all the parts of the software stack, the appropriate columns will be null.

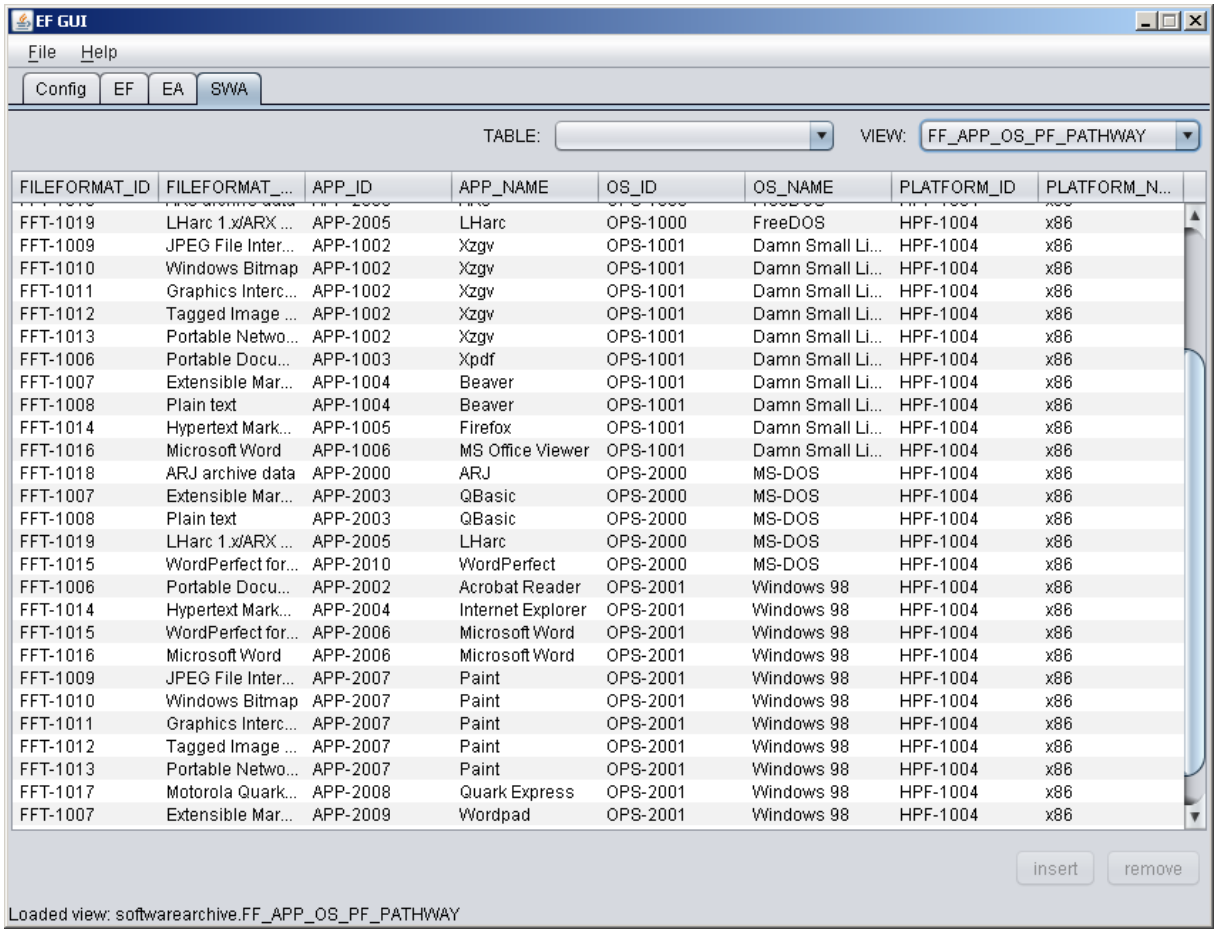
Example of file formats to platform pathways:



Example of file formats to operating system to platform pathways:



Example of file formats to applications to operating system to platform pathways:



7 Data model and schemas

7.1 Emulator Package

Schema: EmulatorPackageSchema.xsd

Each EF-compliant emulator is transferred from the Emulator Archive to a receiver in an Emulator Package, schematically shown in Figure 2.

An Emulator Package contains a *package* element describing the package itself with an *id*, *version* and *type* field, as well as a package name.

The *emulator* element describes the emulator software and includes some descriptive fields (such as *name, version, and description*) and technical elements such as a list of *hardware* that the emulator can emulate, a list of software *imageFormat* (such as FAT12, FAT32, D64, etc.) that the emulator can read.

The *emulator.executable* element contains information about the executable itself. The *type* field defines the type of executable (such as jar for java-based emulators, exe for Windows native executables and ELF for Linux executables); the *name* field contains the executable file name. The location field contains the local path within the container file from where the binary will run.

Finally, the *emuLanguage_list* element consists of a list of language codes referenced to by the emulators.

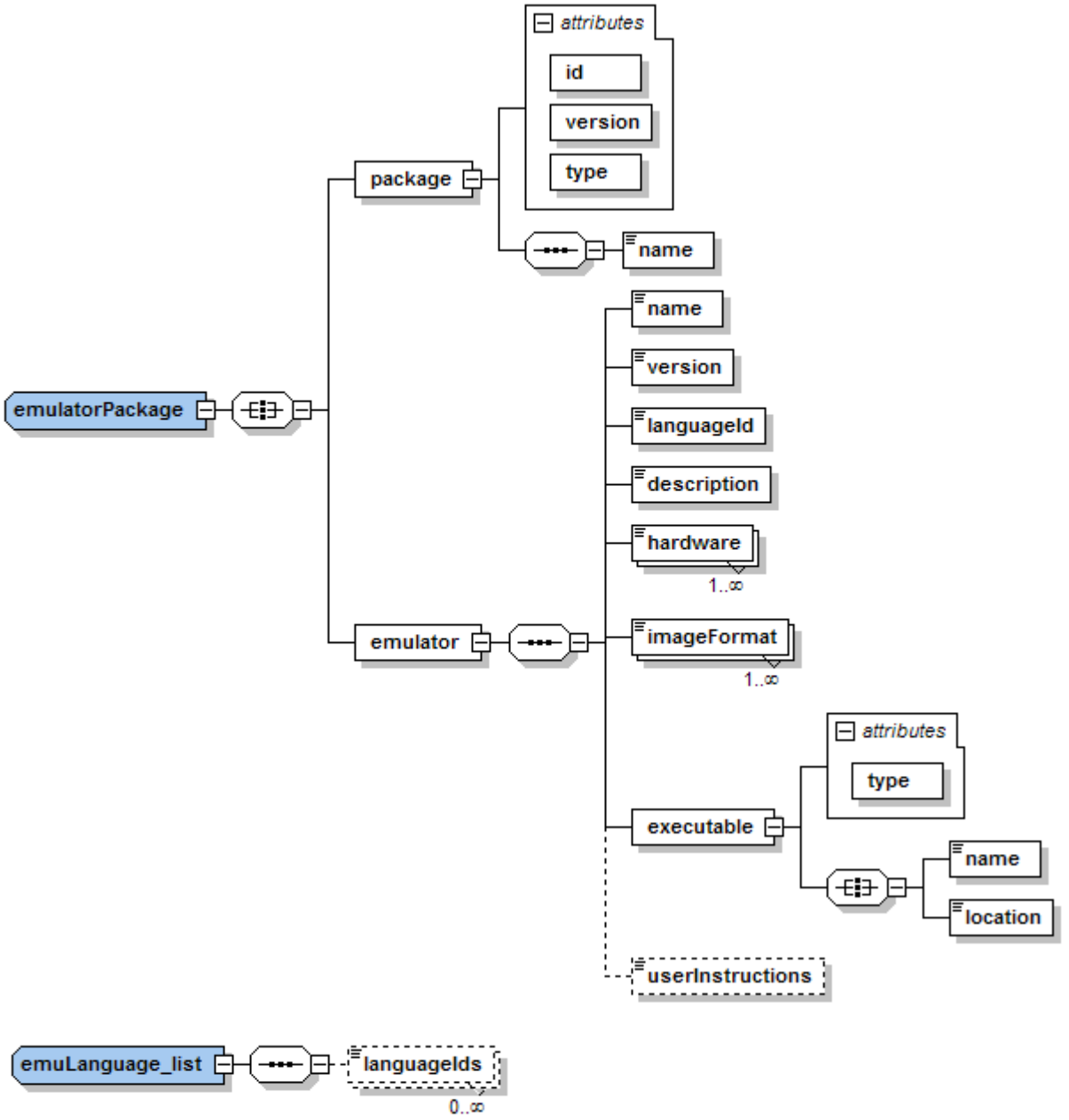


Figure 2: Emulator Package schema

Note: the version number of the package element is different from the version number of the emulator. The latter corresponds to the actual emulator software version number whereas the package version only concerns the package itself which can then be used to update existing packages with newer package version if necessary.

An example of metadata that corresponds to the schema above is as follows:

```
<ea:emulatorPackage xmlns:ea="http://emulatorarchive.keep-project.eu/EmulatorPackage">
    <package id="1" version="1" type="zip">
        <name>Dioscuri_042.zip</name>
    </package>
    <emulator>
        <name>Dioscuri</name>
```

```
        <version>0.4.2</version>
        <languageId>en</languageId>
        <description>Dioscuri, the modular emulator</description>
        <hardware>x86</hardware>
        <imageFormat>FAT12</imageFormat>
        <imageFormat>FAT16</imageFormat>
        <executable type="jar">
            <name>Dioscuri-0.4.2.jar</name>
            <location>.</location>
        </executable>
    </emulator>
</ea:emulatorPackage>
```

7.2 Emulator Archive web services

Schema: emulatorarchive.wsdl

The Emulator Archive offers several web services to obtain data from the database. The externally available services are shown in Figure 3, along with their parameters.

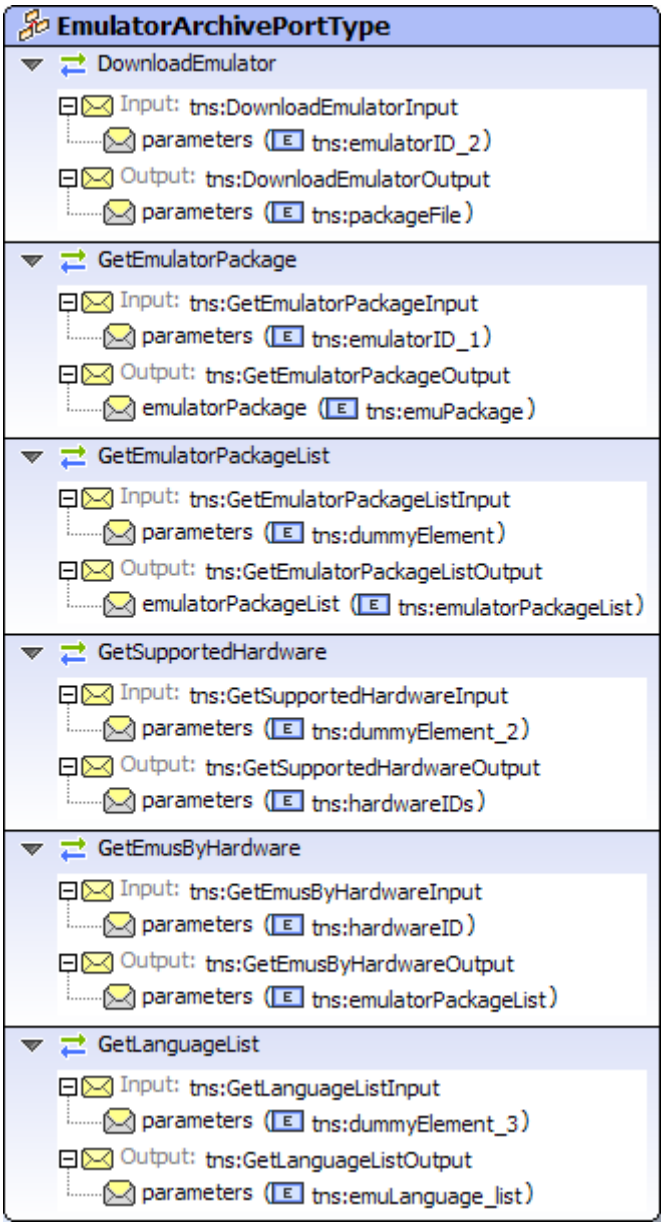


Figure 3: Emulator Archive web services

The following table explains these functions in more detail:

Function	Input	Output	Explanation
DownloadEmulator	Emulator Package ID	Binary Stream (File)	Downloads the emulator binary from the database, given a valid emulator ID
GetEmulatorPackage	Emulator Package ID	Emulator Package	Retrieves the emulator metadata (not including the binary) given a valid emulator ID
GetEmulatorPackageList	Dummy number	All Emulator Packages	Retrieves the emulator metadata (not including the binary) for all packages in the database. Note that a dummy input must be supplied

GetSupportedHardware	Dummy number	All hardware IDs	Retrieves all hardware IDs in the database Note that a dummy input must be supplied
GetEmusByHardware	Hardware ID	Emulator Packages	Retrieves the emulator metadata (not including the binary) for all packages that support the given hardware ID.
GetLanguageList	Dummy number	All referenced languages	Returns a list of languages that are used by one or more emulators

7.3 Dependency (pathway) schema

Schema: PathwaySchema.xsd

The environment that can render digital objects, consisting of the digital object file format, and hardware platform and possibly an application and/or operating system, is called a Pathway. The Software Archive defines a Pathway in an XSD schema, which is schematically shown in Figure 4 below:

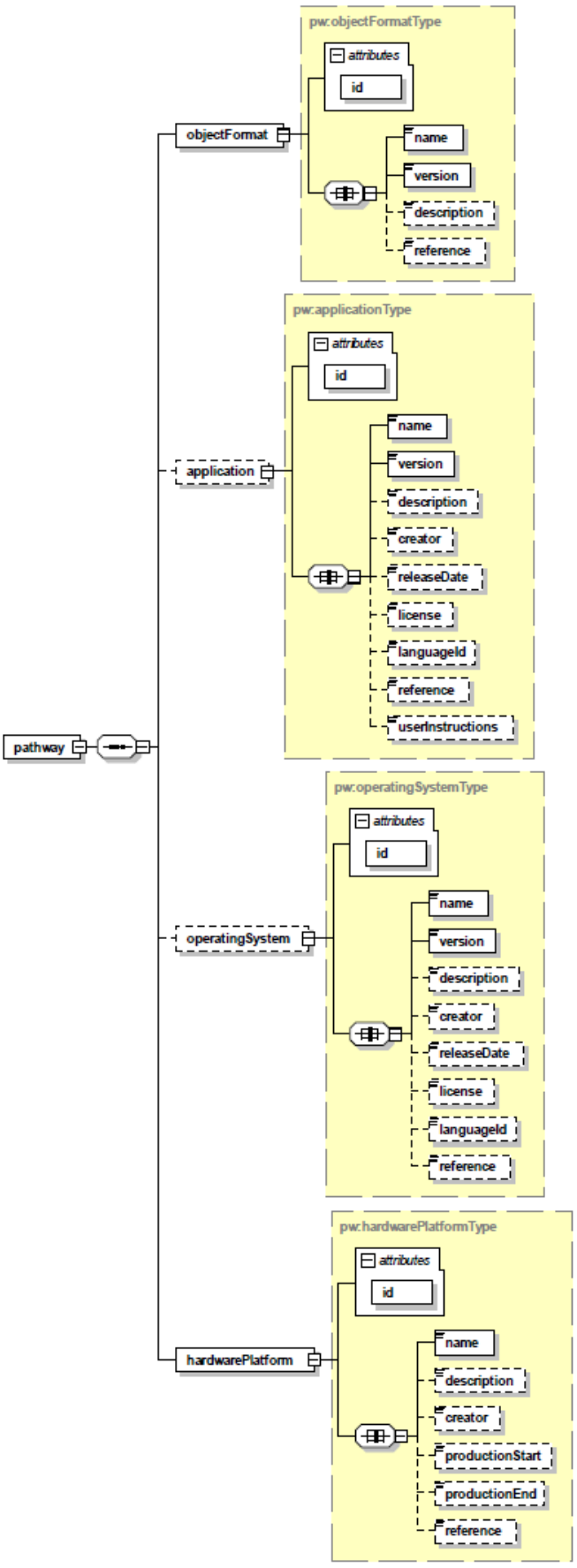


Figure 4: Pathway schema

Each of the four elements has a mandatory *ID, name* and (except for the hardware platform) *version*. Other optional elements include a *description, creator, reference*, etc.

Although a Pathway is made up of four elements (digital object format, application, operating system and hardware platform), only the object format and platform are mandatory. The application and operating system may not need to be defined for a valid Pathway.

The Pathway schema also defines a *registryType* element, which describes metadata for Technical Registries, and a *efFormat* element, which describes an EF fileformat:

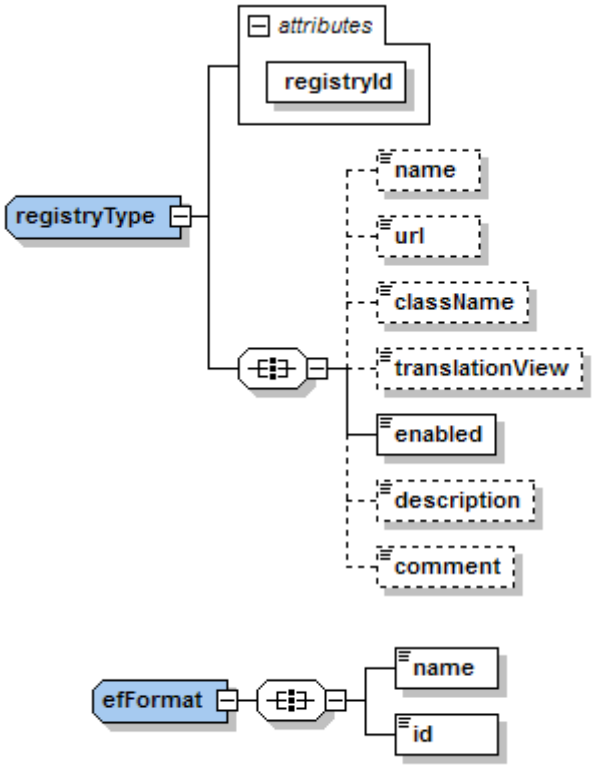


Figure 5: Pathway schema: registryType and efFormat elements

7.4 Software Package schema

Schema: SoftwarePackageSchema.xsd

The software required to render digital objects is transferred from the Software Archive to a receiver in a Software Package, schematically shown in Figure 5.

The Software Package contains a software image which consists of one or more operating systems containing one or more applications. The descriptive metadata has a Software Package *id* (attribute), and *format* and *description* elements. The *format* element corresponds to the *imageFormat* element in the Emulator Package schema.

Each operating system (*os* element) and application (*app* element) is defined using the Pathway operating system and application type, respectively.

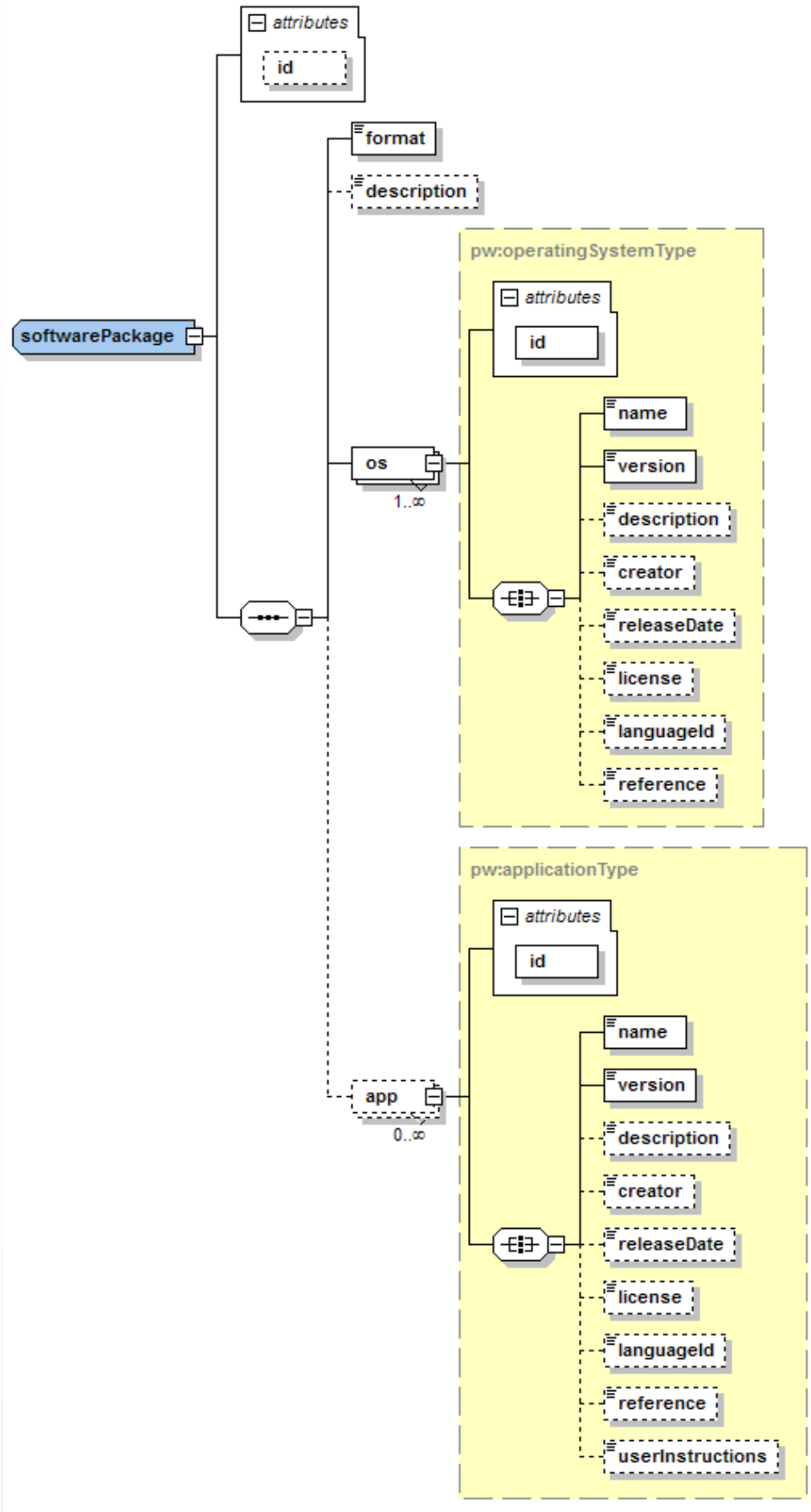


Figure 5: Software Package Schema

An example of metadata that corresponds to the schema above is as follows:

```
<softwarePackage id="IMG-1000">
        <description>MS-DOS 5.00 Operating System + basic text editor EDIT 1.0</description>
        <format>FAT32</format>
        <pw:os id="OPS-2000">
            <name>MSDOS</name>
            <version>5.00</version>
        </pw:os>
        <pw:app id="APP-3000">
            <name>EDIT</name>
            <version>1.0</version>
        </pw:app>
        <pw:app id="APP-3001">
            <name>Q-BASIC</name>
            <version>2.1</version>
        </pw:app>
</softwarePackage>
```

7.5 Software Archive web services

Schema: softwarearchive.wsdl

The Software Archive offers several web services to obtain data from the database. The externally available services are shown in Figure 6, along with their parameters.

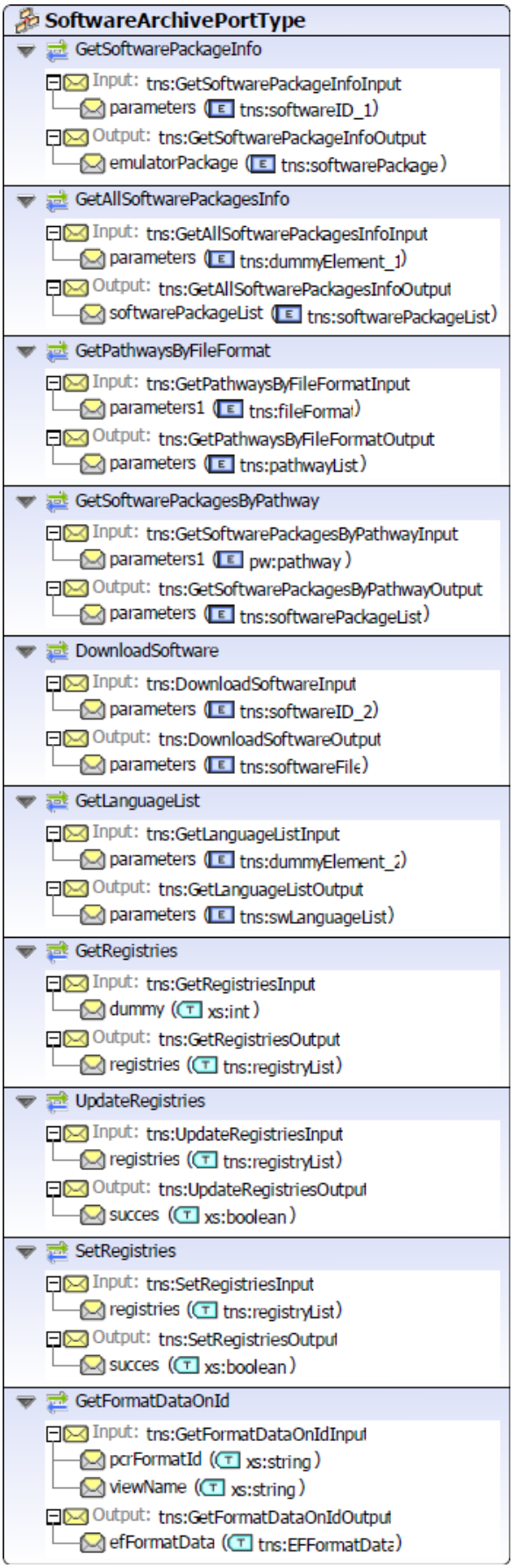


Figure 6: Software Archive web services

The following table explains these functions in more detail:

Function	Input	Output	Explanation
GetSoftwarePackageInfo	Software Package ID	Software Package	Retrieves the software metadata (not including the binary) given a valid software ID
GetAllSoftwarePackagesInfo	Dummy String	All Software Packages	Retrieves the software metadata (not including the binary) for all packages in the database. Note that a dummy input must be supplied
GetPathwaysByFileFormat	File Format	Pathways	Retrieves the viable Pathways for rendering a file given the File Format
GetSoftwarePackagesByPathway	Pathway	Software Packages	Retrieves the software metadata (not including the binary) for all packages that can satisfy the given Pathway.
DownloadSoftware	Software Package ID	Binary Stream (File)	Downloads the software image from the database, given a valid Software Package ID
GetLanguageList	Dummy String	All referenced languages	Returns a list of languages that are used by one or more operating systems or applications.
GetRegistries	Dummy number	All registered registries	Returns metadata for the registered Technical Registries
UpdateRegistries	List of Registries	Updates registries in the database.	Updates metadata for Technical Registries. The registries must already exist in the database.
SetRegistries	List of Registries	Inserts registries in the database	Insert metadata for Technical Registries. The old metadata will be overwritten.
GetFormatDataOnId	PCR Fromat ID, database view name	Retrieves the EF fileformat ID and fileformat name from the database given a PCR ID	Retrieves the fileformat ID and name used internally by the EF, given a PCR ID.

8 Quick Guide to running the Emulation Framework

Currently the default interface to run the Emulation Framework is the test GUI, part of the Emulation Framework. The test GUI is described in detail in the [SUG].

An alternative, command-line interface is also available, and is generated as part of the *release.installer* target. To select the command-line interface, rather than the test GUI, the main class in the EFCore.jar manifest must be changed from eu.keep.gui.GUI to eu.keep.core.EFCliAutoComp. The Ant target *jar* can be changed to do this automatically.

Before running the Emulation Framework, make sure that the required Software Archive and Emulator Archive releases are available (run the respective *release* targets in the Software Archive and Emulator Archive build files).

Command Line Interface

The Command Line Interface, based on BeanShell, is provided as part of the Emulation Framework Core so that it can be debugged without requiring an external interface such as a front-end GUI.

This built-in shell offers direct access to the public API (see section 9) by creating an instance *m* of the *Kernel* object that would normally be used by the host program. The auto-completion (using the tab key) of methods name and file/directory paths makes for quicker and easier usability.

Basic workflow

Here is a list of the basic commands:

- Digital object characterisation:

```
m.characterise(new File("/my/path/to/file/myFile.xyz"))
```

- Start an emulation process from a digital object with no metadata

```
m.start (new File("./testData/digitalObjects/text.txt"))
```

- Start an emulation process from a digital object with metadata included

```
m.start (new File("./testData/digitalObjects/text.txt"), new File("./testData/digitalObjects/text.txt.xml"))
```

9 Public API

The API is defined by the interface definition in **CoreEngineModel.java,** which is implemented by the **Kernel.java** class.

The following sections show how to use the Command Line interface to achieve different results. For instructions on how to use the GUI, please refer to [SUG].

9.1 Running an emulation process manually

An emulation process can run manually, where all the steps normally done automatically when using the start() methods can run sequentially. This allows the user to manually set the options.

- **Characterise a file**

  The characterisation process will return a list of *Format* object which can, in turn, be queried for more detailed information about the identified format.

```
m.characterise(new File("/my/path/to/file/myFile.xyz"))
m.getTechMetadata(new File("/my/path/to/file/myFile.xyz"))
m. getFileInfo(new File("/my/path/to/file/myFile.xyz"))
```

- **Retrieve Pathways for a given file format**

  Once a format has been selected, it is necessary to retrieve a list of Pathways that can render the format.

```
List<Format> formats = m.characterise(new File("/my/path/to/file/myFile.xyz"))
List<Pathway> Pathways = m. getPathways(formats.get(0))
m.isPathwaySatisfiable(Pathways.get(0))
```

  The list of Pathways thus retrieved only represents a list of potential/theoretical Pathways but doesn't have any knowledge of the available software and emulators that are available locally or via an archive. It is therefore important to filter this list with only satisfiable Pathways, i.e. Pathways can actually be rendered by the Emulation Framework. For this, the method *isPathwaySatisfiable* can be used to verify the satisfiability of a given Pathway by checking the availability of the required emulator and software images.

  Similarly, an automatic selection of a valid Pathway can be achieved with a call to the method *autoSelectPathway* which returns the first satisfiable Pathway.

- **Set the list of allowed emulators**

  Before selecting an emulator, the list containing the emulators which are allowed to be used must be populated. This can be done as follows:

```
m.getWhitelistedEmus()
m.whiteListEmulator(1)
```

To remove an emulator from the allowed list, the method *unListEmulator* can be used

- **Retrieve emulator/software images for a given Pathway**

  Once a Pathway has been selected, a list of compatible emulators and software images must be produced.

  To do so, a couple of method calls are provided.

  First a list of emulator and a list of software image must be retrieved from the Pathway:

```
m.getEmulatorsByPathway(Pathways.get(0))
m.getSoftwareByPathway(Pathways.get(0))
```

A matching between the list of emulators and software images has to be performed as certain emulators may not be compatible with certain image formats. A map of emulators with their respective list of compatible software images can be retrieved as follows.

```
m.matchEmulatorWithSoftware(Pathways.get(0))
```

If an automatic selection of a specific emulator and software is required, the methods *autoSelectEmulator* and *autoSelectSoftwareImage* can be used.

- **Configure and run the emulation process**

  Once an emulator and software image have been selected, the emulation process needs to be configured and started. This is achieved as follows:

```
List<EmulatorPackage> emuList = m.getEmuListFromArchive()
List<SotfwarePackage> swList = m.getSoftwareListFromArchive()
Integer i = m.prepareConfiguration(new File("/my/path/to/file/myFile.xyz"), emuList.get(0), swList.get(0), Pathways.get(0))
m.runEmulationProcess(i)
```

Note: the above example picks a random emulator and software; this is not recommended but shown for illustrative purposes. It is suggested to use the previously shown emulator/software Pathway selection.

It is then possible to modify the default option set for the chosen emulator by retrieving its list of options, modify them and finally reset them back, before actually building the configuration, by using the methods *getEmuOptions* and *setEmuOptions*

10 Appendix A: Ant targets

Table 2: Complete list of external Ant targets

Ant target	Comment
compile	Compiles the code base
checkstyle	Runs checkstyle, the static code analysis tool for java source code
clean	Deletes output files and directories created during a build, i.e. *./build*, *./src/generated/*
copy.resources	Copies the required resources (property files, schemas, etc.) to the classpath
db.create	Creates and populates the internal database
db.drop	Deletes the database
db.reset	Resets the database (calls db.drop and db.create consecutively)
generated.src	Generates source code from WSDL/XSD files
ivy-publish	Publish the EFKernel jar to the repository
ivy-publish-external	Publish the external jars to the repository
ivy-report	Generates a report detailing all the dependencies of the module
ivy-resolve	Resolves transitive dependencies
ivy-retrieve	Retrieve dependencies into cache
jar	Creates a JAR
javadoc	Runs the javadoc, document generator for Java source code
release	Creates a release package for the Core project
release.installer	Creates a release package for the Core, Emulator Archive and Software Archive using IzPack. Requires the Emulator and Software Archive to be available and build
svnstat	Runs svnstat, a tool that generates statistic on subversion usage
test.compile	Compiles the unit tests source code
test.copy.resources	Copies the required test resources (property files, schemas, etc.) to the classpath
test.db.create	Creates the test database
test.db.drop	Deletes the test database
test.db.reset	Resets the test database (calls test.db.drop and test.db.create consecutively)
test.report	Creates the junit html report
test.run	Prepares and runs the unit tests

11 Appendix B: language codes

Table 3: Complete list of language codes

Language code	Language name	Language code	Language name	Language code	Language name
aa	Afar	hy	Armenian	or	Oriya
ab	Abkhazian	hz	Herero	os	Ossetian
ae	Avestan	ia	Interlingua (International Auxiliary Language Association)	pa	Punjabi
af	Afrikaans	id	Indonesian	pi	Pali
ak	Akan	ie	Occidental	pl	Polish
am	Amharic	ig	Igbo	ps	Pashto
an	Aragonese	ii	Sichuan Yi	pt	Portuguese
ar	Arabic	ik	Inupiaq	qu	Quechua
as	Assamese	io	Ido	rm	Romansh
av	Avaric	is	Icelandic	rn	Rundi
ay	Aymara	it	Italian	ro	Romanian
az	Azerbaijani	iu	Inuktitut	ru	Russian
ba	Bashkir	ja	Japanese	rw	Kinyarwanda
be	Belarusian	jv	Javanese	sa	Sanskrit
bg	Bulgarian	ka	Georgian	sc	Sardinian
bh	Bihari languages	kg	Kongo	sd	Sindhi
bi	Bislama	ki	Kikuyu	se	Northern Sami
bm	Bambara	kj	Kwanyama	sg	Sango
bn	Bengali	kk	Kazakh	si	Sinhala
bo	Tibetan	kl	Kalaallisut	sk	Slovak
br	Breton	km	Central Khmer	sl	Slovenian
bs	Bosnian	kn	Kannada	sm	Samoan
ca	Catalan	ko	Korean	sn	Shona
ce	Chechen	kr	Kanuri	so	Somali
ch	Chamorro	ks	Kashmiri	sq	Albanian
co	Corsican	ku	Kurdish	sr	Serbian
cr	Cree	kv	Komi	ss	Swati
cs	Czech	kw	Cornish	st	Sotho, Southern
cu	Church Slavic	ky	Kyrgyz	su	Sundanese
cv	Chuvash	la	Latin	sv	Swedish
cy	Welsh	lb	Letzeburgesch	sw	Swahili
da	Danish	lg	Ganda	ta	Tamil
de	Deutsch	li	Limburger	te	Telugu
dv	Dhivehi	ln	Lingala	tg	Tajik

dz	Dzongkha	lo	Lao	th	Thai
ee	Ewe	lt	Lithuanian	ti	Tigrinya
el	Greek, Modern (1453-)	lu	Luba-Katanga	tk	Turkmen
en	English	lv	Latvian	tl	Tagalog
eo	Esperanto	mg	Malagasy	tn	Tswana
es	Spanish	mh	Marshallese	to	Tonga (Tonga Islands)
et	Estonian	mi	Maori	tr	Turkish
eu	Basque	mk	Macedonian	ts	Tsonga
fa	Persian	ml	Malayalam	tt	Tatar
ff	Fulah	mn	Mongolian	tw	Twi
fi	Finnish	mr	Marathi	ty	Tahitian
fj	Fijian	ms	Malay	ug	Uyghur
fo	Faroese	mt	Maltese	uk	Ukrainian
fr	French	my	Burmese	ur	Urdu
fy	Western Frisian	na	Nauru	uz	Uzbek
ga	Irish	nb	Norwegian Bokmål	ve	Venda
gd	Gaelic	nd	North Ndebele	vi	Vietnamese
gl	Galician	ne	Nepali	vo	Volapük
gn	Guarani	ng	Ndonga	wa	Walloon
gu	Gujarati	nl	Nederlands	wo	Wolof
gv	Manx	nn	Norwegian Nynorsk	xh	Xhosa
ha	Hausa	no	Norwegian	yi	Yiddish
he	Hebrew	nr	South Ndebele	yo	Yoruba
hi	Hindi	nv	Navaho	za	Zhuang
ho	Hiri Motu	ny	Chichewa	zh	Chinese
hr	Croatian	oc	Occitan (post 1500)	zu	Zulu
ht	Haitian	oj	Ojibwa		
hu	Hungarian	om	Oromo		