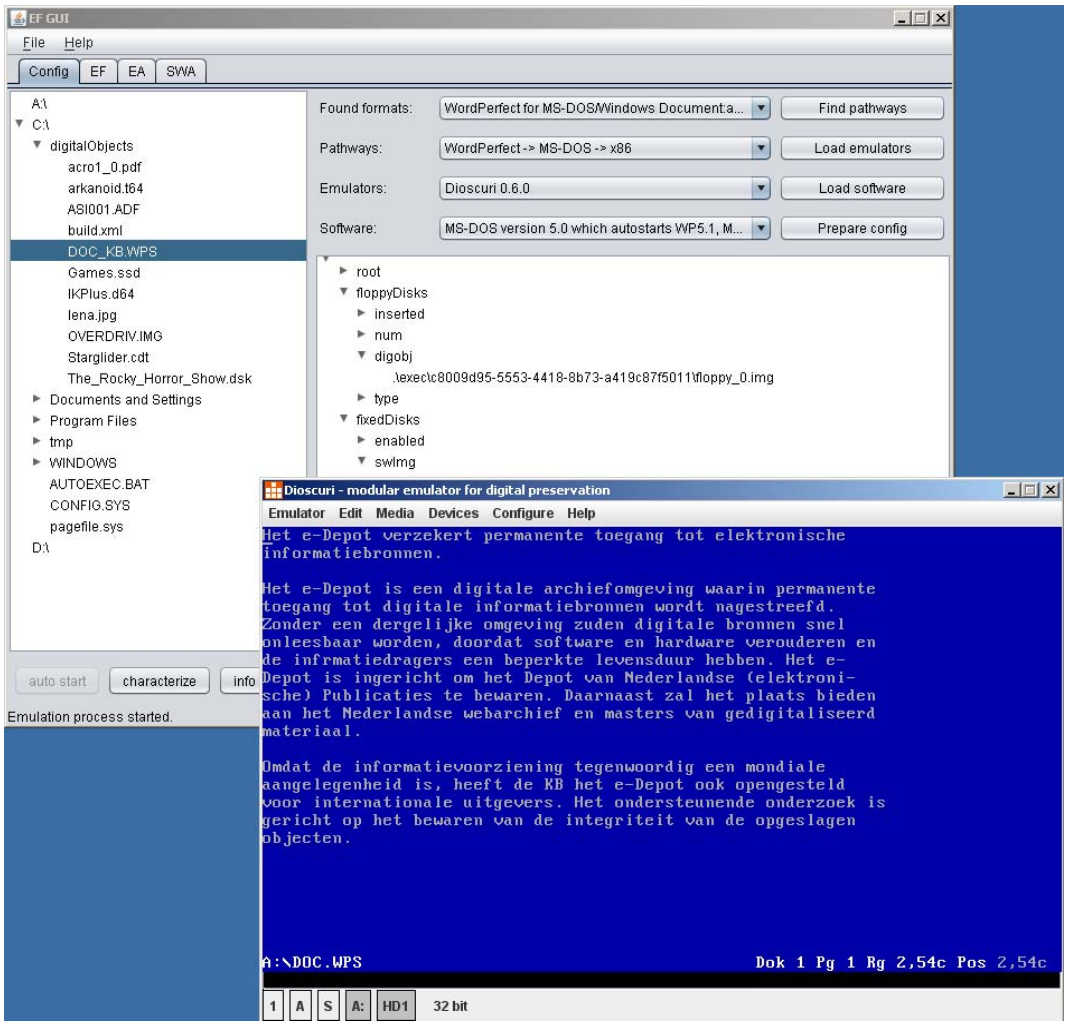
Keeping Emulation Environments Portable

FP7-ICT-231954

# System Maintenance Guide for the Emulation Framework version 1.0 (May 2011)



| | |
|---|---|
| Deliverable number | *Part of deliverable D2.3 (based on I2.2)* |
| Nature | *Report* |
| Dissemination level | *CO* |
| Status | *Draft / Finalised / Reviewed / Final* |
| Workpackage number | *WP2* |
| Lead beneficiary | *TES* |
| Author(s) | *Bram Lohman (Tessella)* |

## Document history

### Revisions

| Version | Date | Author | Changes |
|---|---|---|---|
| 0.1 | 23-05-2011 | Bram Lohman | Initial version |
| 1.0 | 30-05-2011 | Bram Lohman | Prepared for release |
| 1.0 | 10-11-2011 | Bart Kiers | Adjusted incorrect links to SVN repository. |

# Executive Summary

This document is a System Maintenance Guide 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 former 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 schema's 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.

## List of Related Documents

| | |
|---|---|
| Description of Work [DoW] | Overall project description |
| User Requirements Document [URD] | Requirements for the Emulation Framework |
| Architectural Design Document [ADD] | Design for the Emulation Framework |
| System User Guide [SUG] | User and administrator guide for Emulation Framework |

## Abbreviations

| | |
|---|---|
| Emulation Framework | EF |
| Graphical User Interface | GUI |
| Application Programming Interface | API |
| Web Service Description Language | WSDL |
| File Information Tool Set | FITS |

# 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 is the first version of the SMG for version 1.0 of the Emulation Framework.

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

## 1.3 About KEEP

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

The EF software can be divided into **Core, Software Archive** and **Emulator Archive** components. The Core is the technical heart of the system, performing the automatic characterisation 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.

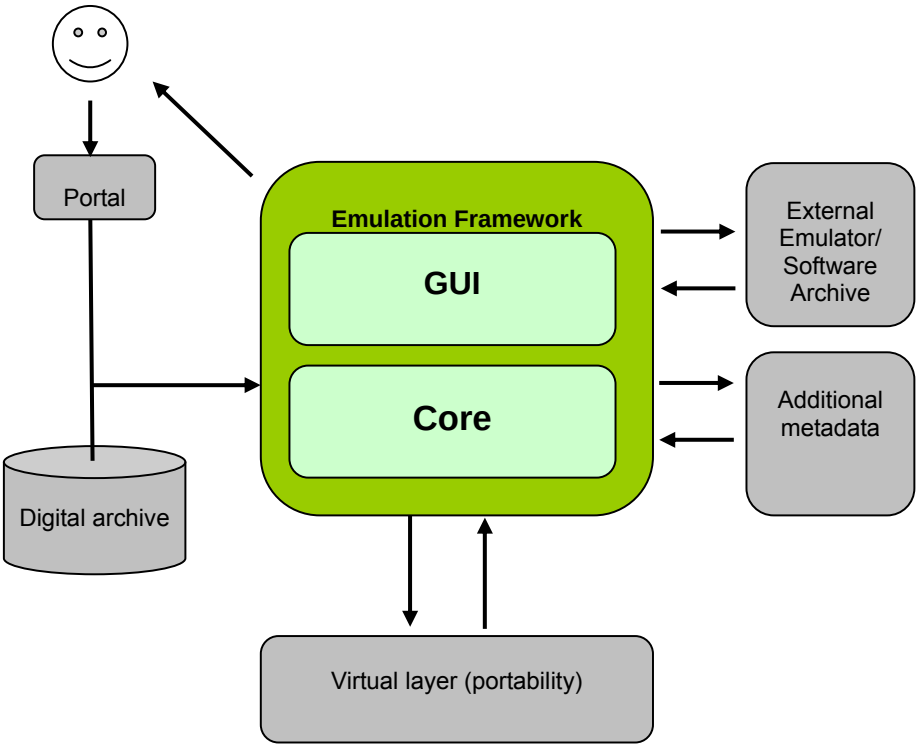



**Figure 1 : EF system overview and system boundaries**

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 configuration of the Core is stored in a file called **user.properties**, 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 if it doesn't exists (FALSE), or fail if it doesn't exist (TRUE) |
| 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) |
| tmp.dir | ./tmp | Directory used by the host system to store temporary files |

The GUI has a separate configuration file called **gui.properties**.

This file contains the database connection parameters for the different components; depending on access level, this may be just the EF, or the EF, Emulator Archive and Software Archive.

| 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.userassw | 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.userassw | 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.userassw | swa | Database user password |

# 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[1]
- Apache Ivy 2.x.x dependency manager[2]
- 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 a Subversion (SVN) repository, 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.

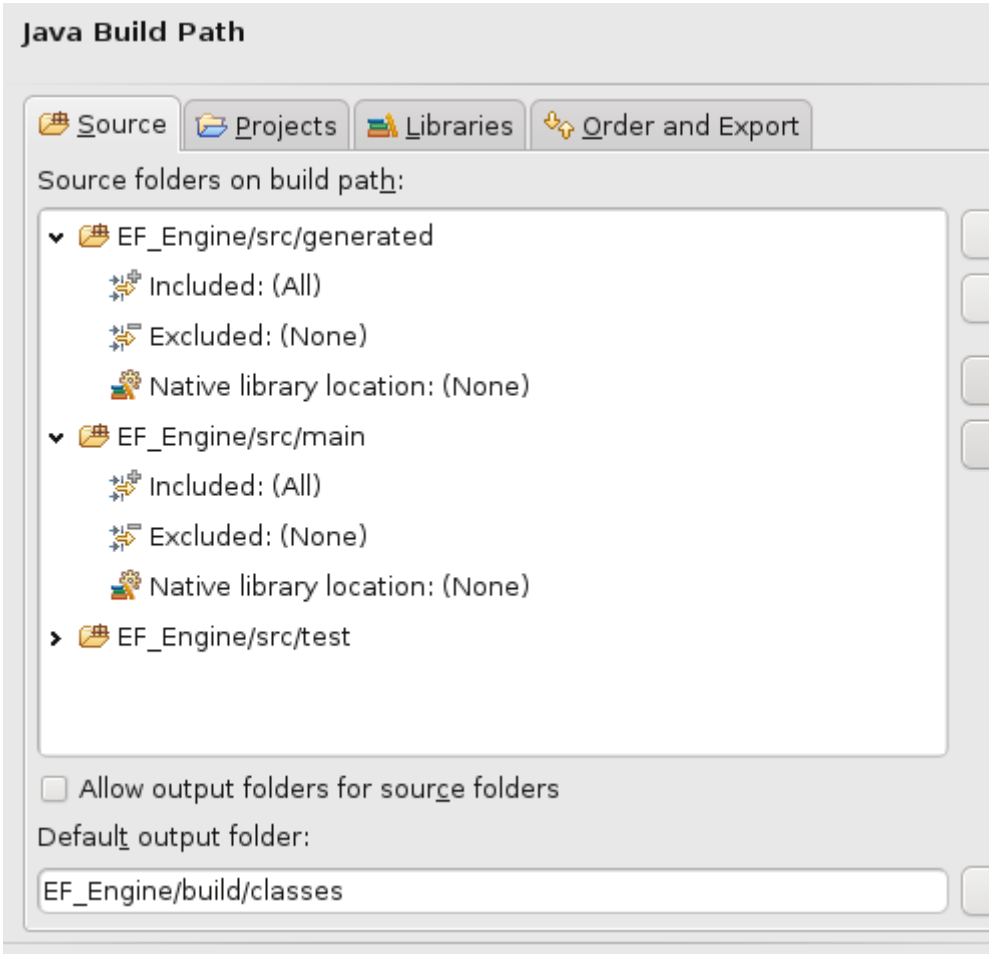
## 4.2 Example development setup

A common Java software development platform is Eclipse[3]. Below are the steps to set up the project in Eclipse.

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

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

[3] 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[4].

## 4.3 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.

**Table 1: Selection of common Ant script targets**

| Ant target | Comment |
|---|---|
| compile | Compiles the code base |

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

| | |
|---|---|
| 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 |

## 4.4 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. The Ant target generated.src will run the necessary code to generate these files.

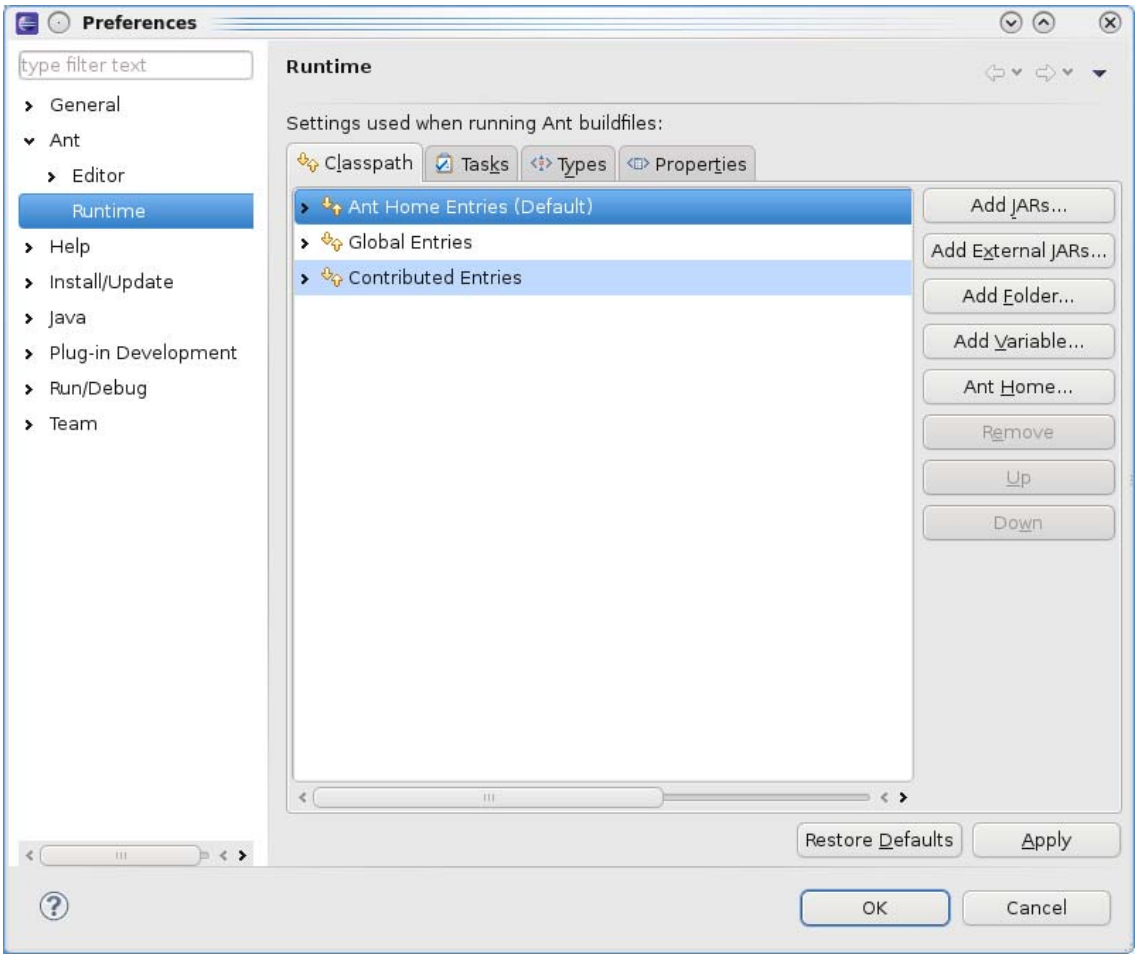
Apache CXF uses the following input files to generate Java code:

- ./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

## 4.5 Managing dependencies

To use Ivy, Apache Ant has to be set up to support it. The latest version of Ivy should be downloaded from http://ant.apache.org/ivy/download.cgi. This is typically a zip file, it is not necessary to install the whole system. The Ivy jar file (ivy-n.n.n.jar e.g. ivy-2.1.0.jar) can be extracted from the archive and added to the ant lib directory (e.g. C:\Program Files\apache-ant-1.8.0\lib).

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 JARs)




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[5] references these jars as a normal dependency.

## 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 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 web-based database viewer.

The viewer can be started by running the H2 library; it should automatically open a browser with the log-in screen.

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

## 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.

### 4.7.1 Quick Guide to running the Emulation Framework

A simple command line script that can run the Emulation Framework is generated as part of the *release.installer* target. 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).

Currently the default interface is the test GUI, part of the Emulation Framework. This can be changed to a command line by changing the main class in the manifest jar. The Ant target *jar* can be changed to do this automatically.

**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 7) 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.

**Graphical User Interface**

The Graphical User Interface provides a simple clickable interface for testing purposes. It offers access to most of the methods of the public API, but not all. However, it provides a user-friendly alternative to the Command Line Interface.

**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"))
```

## 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[6] which in turns uses *./resources/release/install.xml* to configure the installation package.

[6] 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 chapter, 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 internal EF 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 chapter 6.

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 chapter 6.

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 Models and schemas

## 6.1 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. It 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 *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.

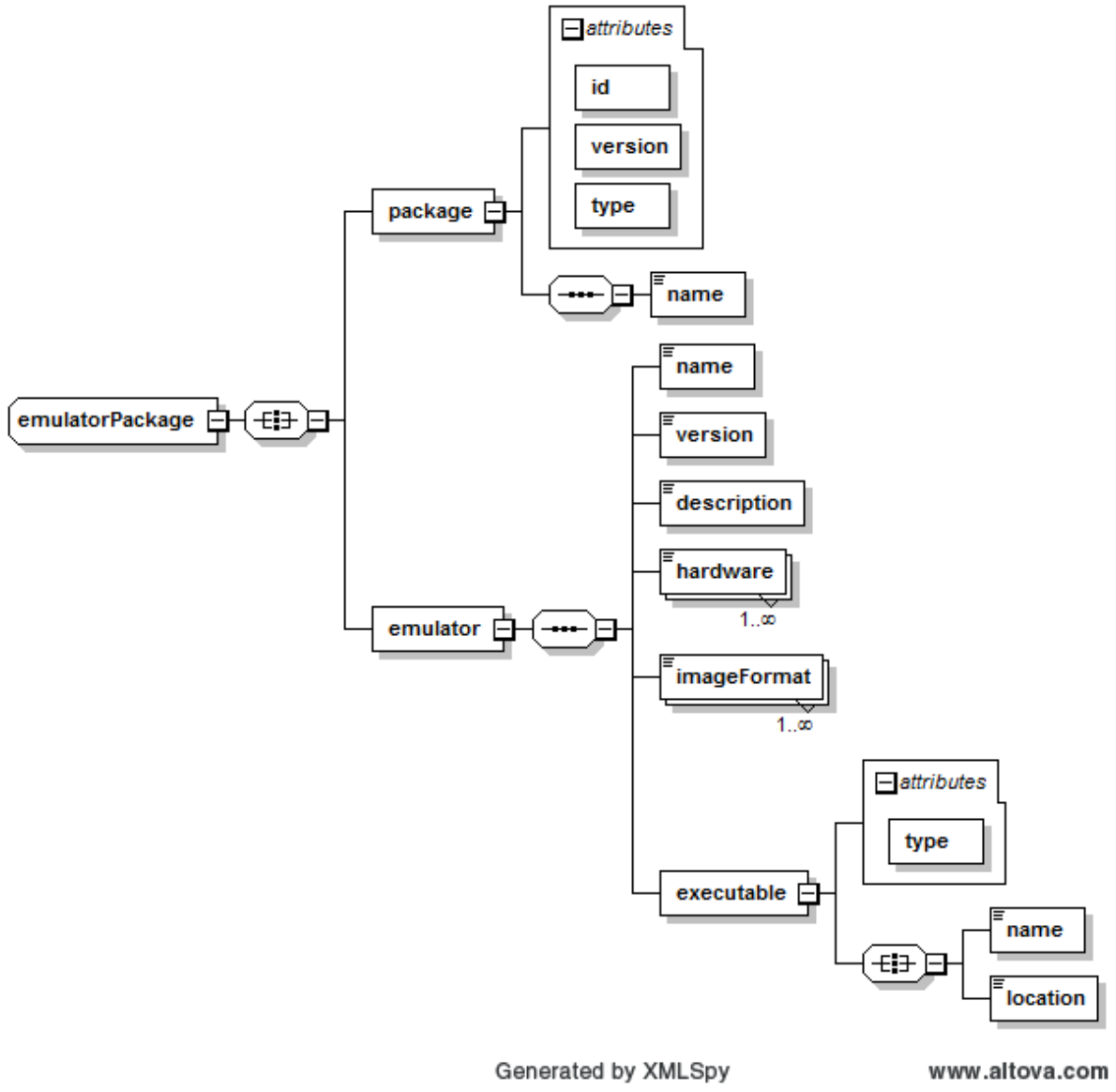



**Figure 2: Emulator Package schema**

**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>
        <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>
```

## 6.2 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.

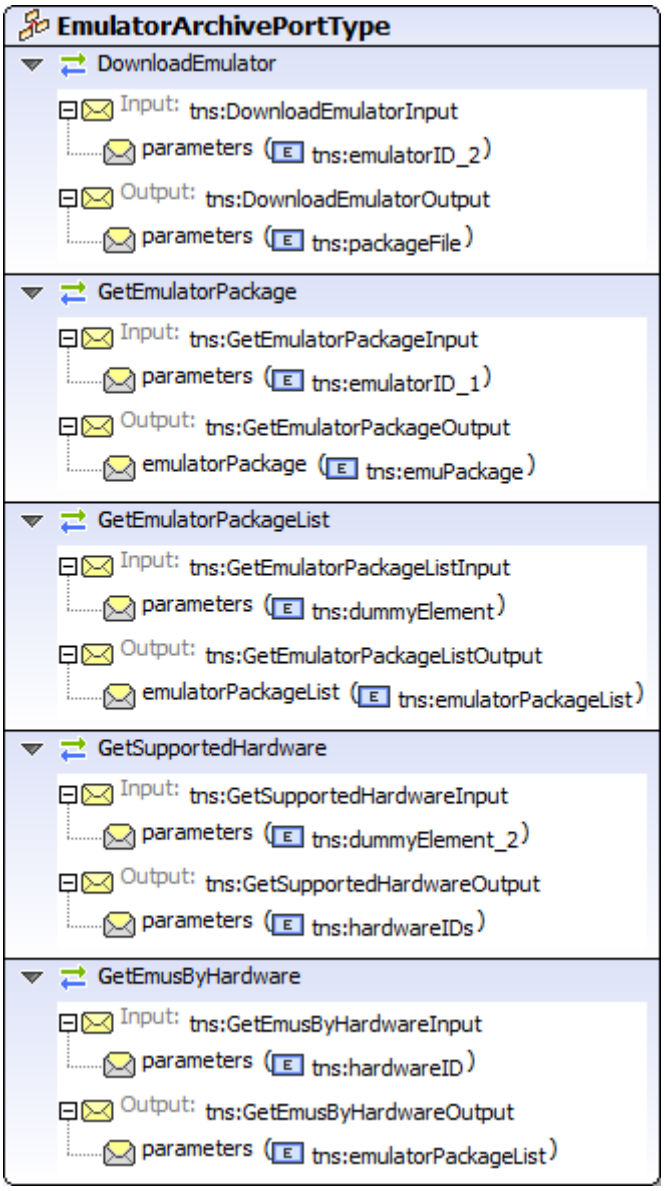



**Figure 3: Emulator Archive web services**

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 | None | 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 | None | 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. |

## 6.3 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:

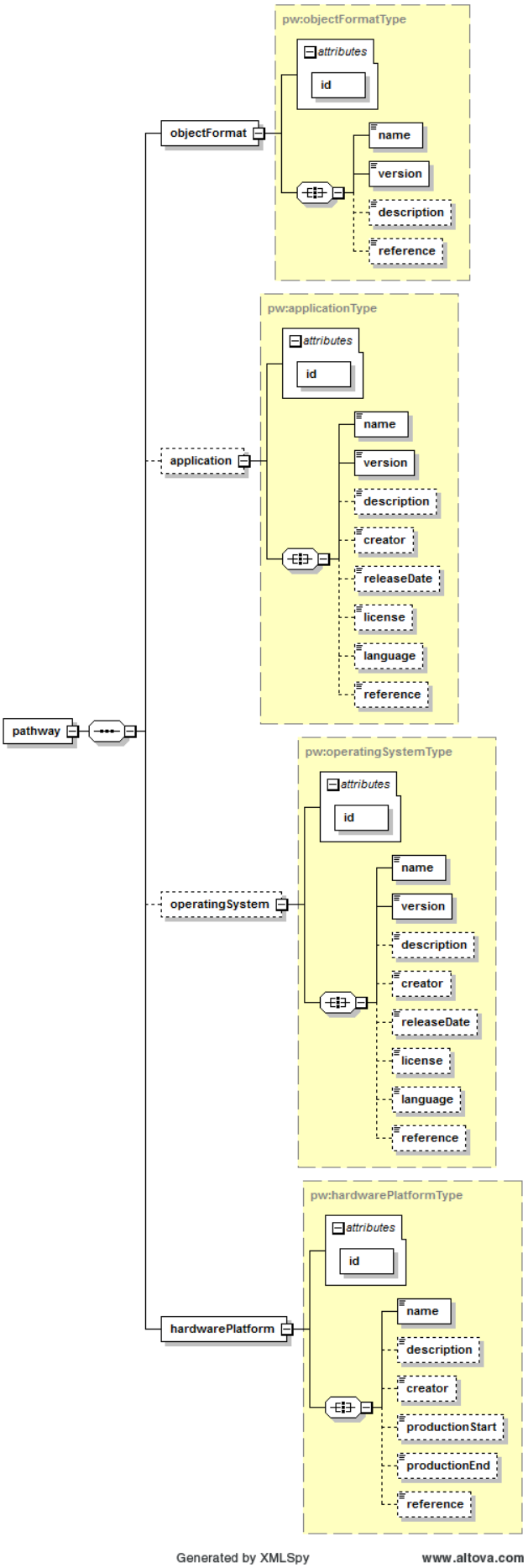

pw:objectFormatType
attributes
id
objectFormat
name
version
description
reference
pw:applicationType
attributes
id
application
name
version
description
creator
releaseDate
license
language
reference
pathway
pw:operatingSystemType
attributes
id
operatingSystem
name
version
description
creator
releaseDate
license
language
reference
pw:hardwarePlatformType
attributes
id
hardwarePlatform
name
description
creator
productionStart
productionEnd
reference
Generated by XMLSpy
www.altova.com

**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.

## 6.4 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.

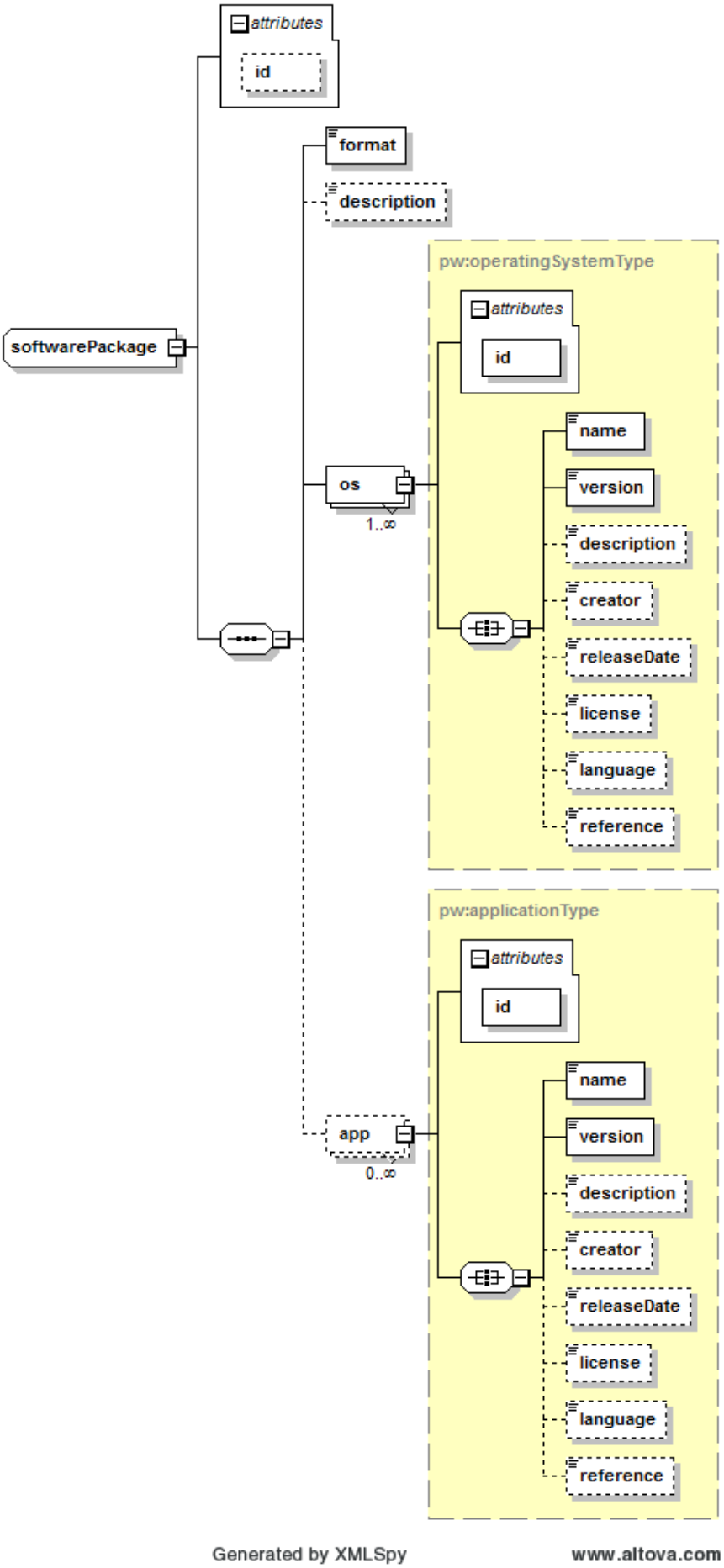

attributes
id
format
description
softwarePackage
pw:operatingSystemType
attributes
id
os
1..∞
name
version
description
creator
releaseDate
license
language
reference
pw:applicationType
attributes
id
app
0..∞
name
version
description
creator
releaseDate
license
language
reference
Generated by XMLSpy
www.altova.com

**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>
```

## 6.5 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.

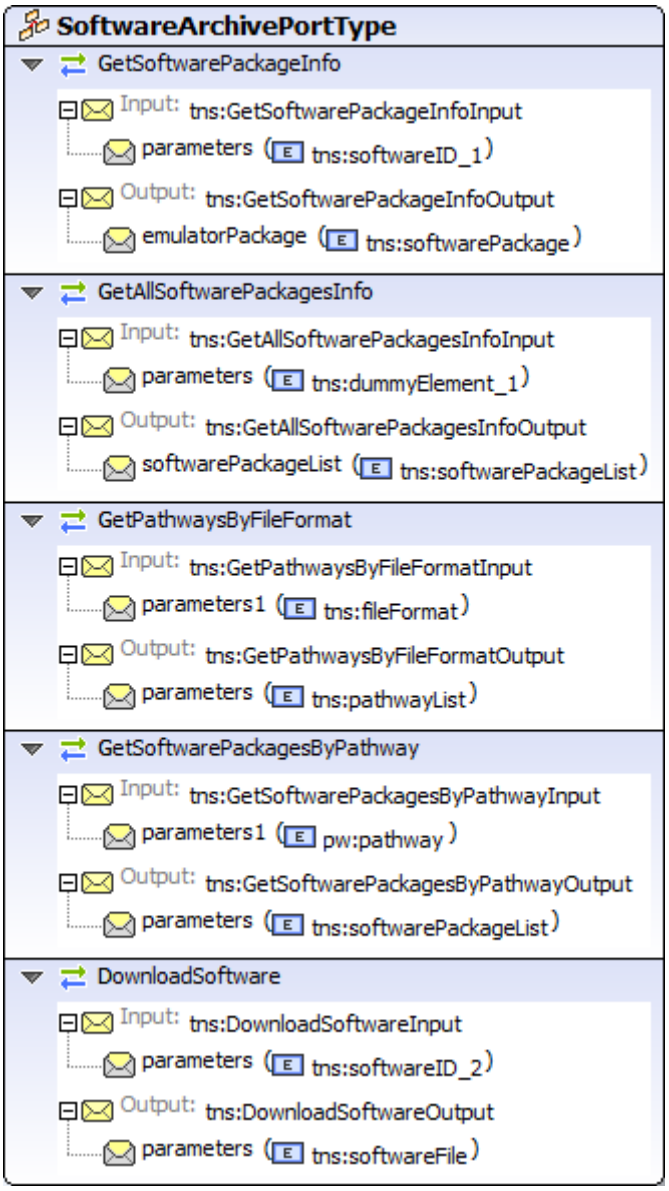



**Figure 6: Software Archive web services**

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 | None | All Software Packages | Retrieves the software metadata (not including the binary) for all packages in the database.<br>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 |

# 7 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].

## 7.1 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*

# 8 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 |
| 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.report | Creates the junit html report |
| test.run | Prepares and runs the unit tests |