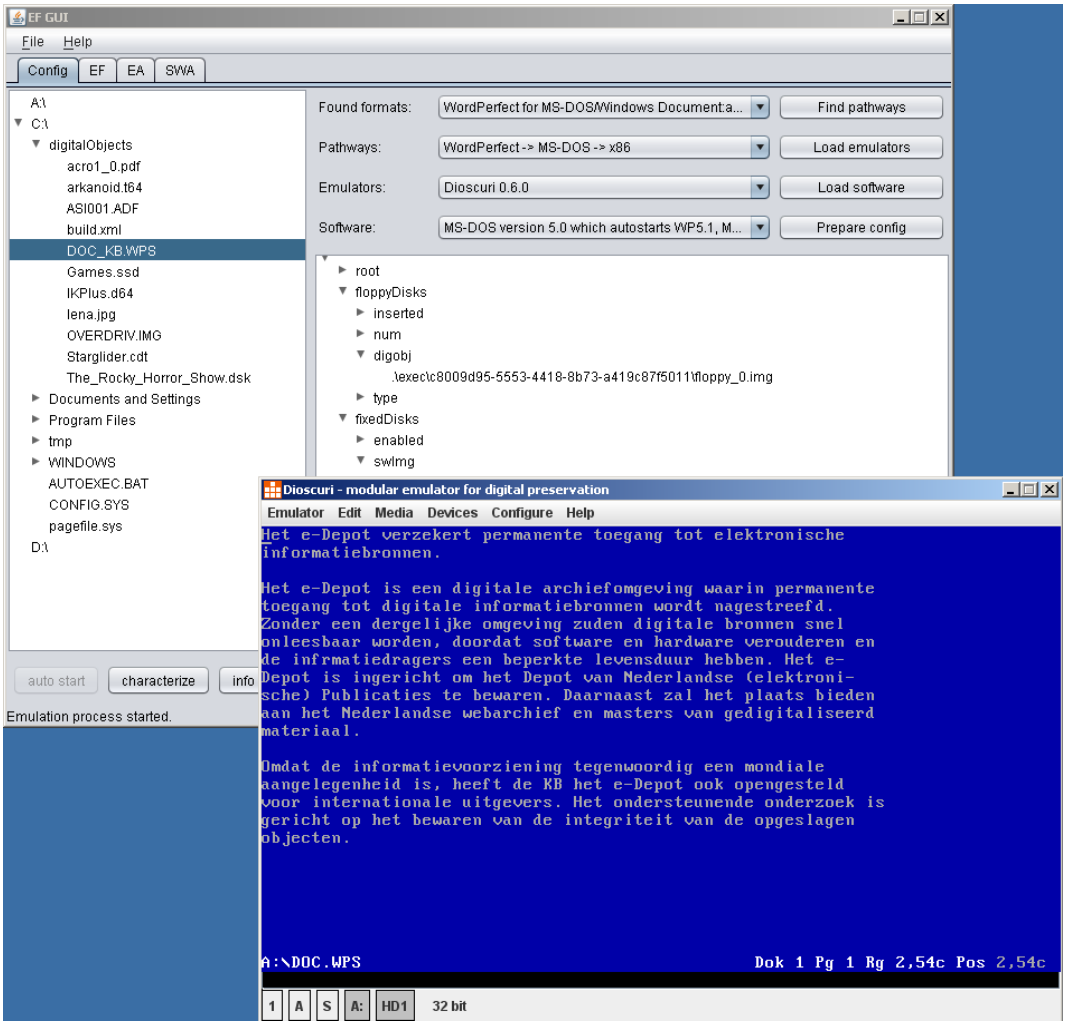
Keeping Emulation Environments Portable

FP7-ICT-231954

Architectural Design Document 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	09-06-2011	Bram Lohman	Initial version
1.0	16-06-2011	Bram Lohman	Prepared for release

1 Executive Summary

This document describes the architectural design for the Emulation Framework, part of the KEEP project. The architectural design outlines the development of the Emulation Framework.

The design follows the Scrum methodology, an iterative software development framework. Major design decisions and motives are documented when made; once detailed component design is completed the design is updated. The document covers the Core component.

The system context includes connections to external interfaces (both inputs and outputs) such as technical registries used for Pathway generation, a Software Archive for Software Package retrieval, an Emulator Archive for downloading Emulator Packages, and the Graphical User Interface to control the system.

The overall system functionality is to provide the user with the best available environment to render a digital object, and this can be broken down into several steps which are described and shown in several workflow diagrams.

The system consists of several components: the Kernel component is the top level component and communicates with and controls each of the other components. The Characteriser takes care of digital objects identification by characterising the input file and optionally connecting to one or more external registries to retrieve information about the required technical environment. The Downloader connects to external systems such as an Emulator Archive and Software Archive retrieve emulators and software required to render the digital object. The Controller contains the logic to configure, run, and output the results of each of the emulators running in the EF.

The EF stores certain information in an internal database, and uses a data access layer to retrieve this information.

Interaction between the user or automated machines is handled via a documented API.

Java was chosen as the development language because of familiarity, widely supported libraries and overall portability; for the internal database H2 was chosen because of the small footprint and integrated web-interface. Software unit tests were used as a testing / debugging strategy to ensure a guarantee a fully working code base.
To fulfil the requirement of supporting at least two emulators, initial support was focused on Dioscuri (x86) and Vice (Commodore 64); the former because of the developer's familiarity with it and the popularity of the platform, the latter because of the maturity of the code and popularity in the gaming community. Support for several other emulators has since been added: Qemu (x86), UAE (Amiga), BeebEm (BBC Micro), JavaCPC (Amstrad CPC).

A glossary containing definitions of terms used in this document is included at the end of the document.

List of Related Documents

Description of Work [DoW]	Overall project description
User Requirements Document [URD]	Requirements for the Emulation Framework
Scrum Product Backlog [SPB]	Development tasks and prototypes

Abbreviations

Koninklijke Bibliotheek	KB
Tessella plc	TES
Description of Work	DoW
User Requirements Document	URD
Architectural Design Document	ADD
Emulation Framework	EF
Unified Modelling Language	UML
Simple Object Access Protocol	SOAP
Web Services Description Language	WSDL
File Information Tool Set	FITS

2 Table of Contents

3 Introduction

This document describes the design of the Emulation Framework described in [DoW].

The design is based on the user requirements and the user scenarios specified in [URD].

This document forms the basis of the planning of the development phase. Since the development follows an iterative approach, the results from the various development iterations will be used to update the design to come to a complete architectural design. At the end of the development the document can be used as a developer's guide and as a basis for writing a System Maintenance Guide.

3.1 Objectives and scope

This design will follow the agile software development methodology, which is based on iterative development. The chosen methodology is Scrum, an iterative incremental framework. The final system will be the result of several iterations of prototypes, expanding and adding complexity during each iteration. Prototypes may implement a solution to a particular issue that seems viable at the time, but at a later stage may hinder further development. It is not unlikely that development will backtrack to resolve these issues and result in different prototypes and / or a different design strategy.

Any such major design decisions will be documented here for future reference.

Motives for design decisions made during the project's development phase will also be noted in this design to ensure the complete design process is documented.

The focus of this design is on the Core; it does not cover the Graphical User Interface, which has been classed as a separate component of the Emulation Framework. However, the requirements of the GUI are also specified in [URD].

3.2 Outline of this document

This release of the design document focuses on a high-level design of the components. Several assumptions have been made, and will be labelled as such.

The overall structure of the document is as follows:

Section 4 contains an overview of the architecture.

Section 5 describes the major system components.

Section 6 lists the prototypes that will be produced in the development phase.

Section 7 has information on external components.

4 System overview

4.1 Overview and Context

The Emulation Framework allows rendering of digital files and computer games in their native environment. This offers 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.

Emulation is done by using existing emulators which are carefully selected on their capability to mimic the functionality of these platforms. The following illustration shows how this works in three steps.

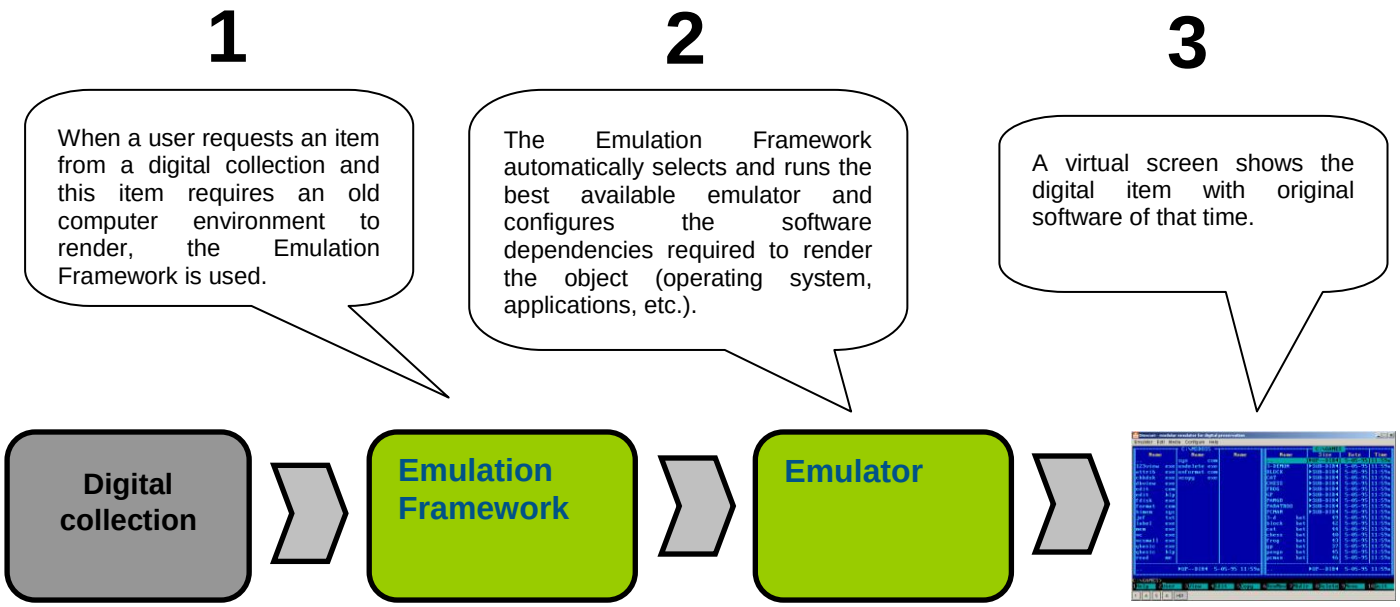


Figure 1: Emulation Framework workflow

A simplified context model diagram of the Emulation Framework, showing the flow of data between the system and its environment across the system boundaries is as follows:

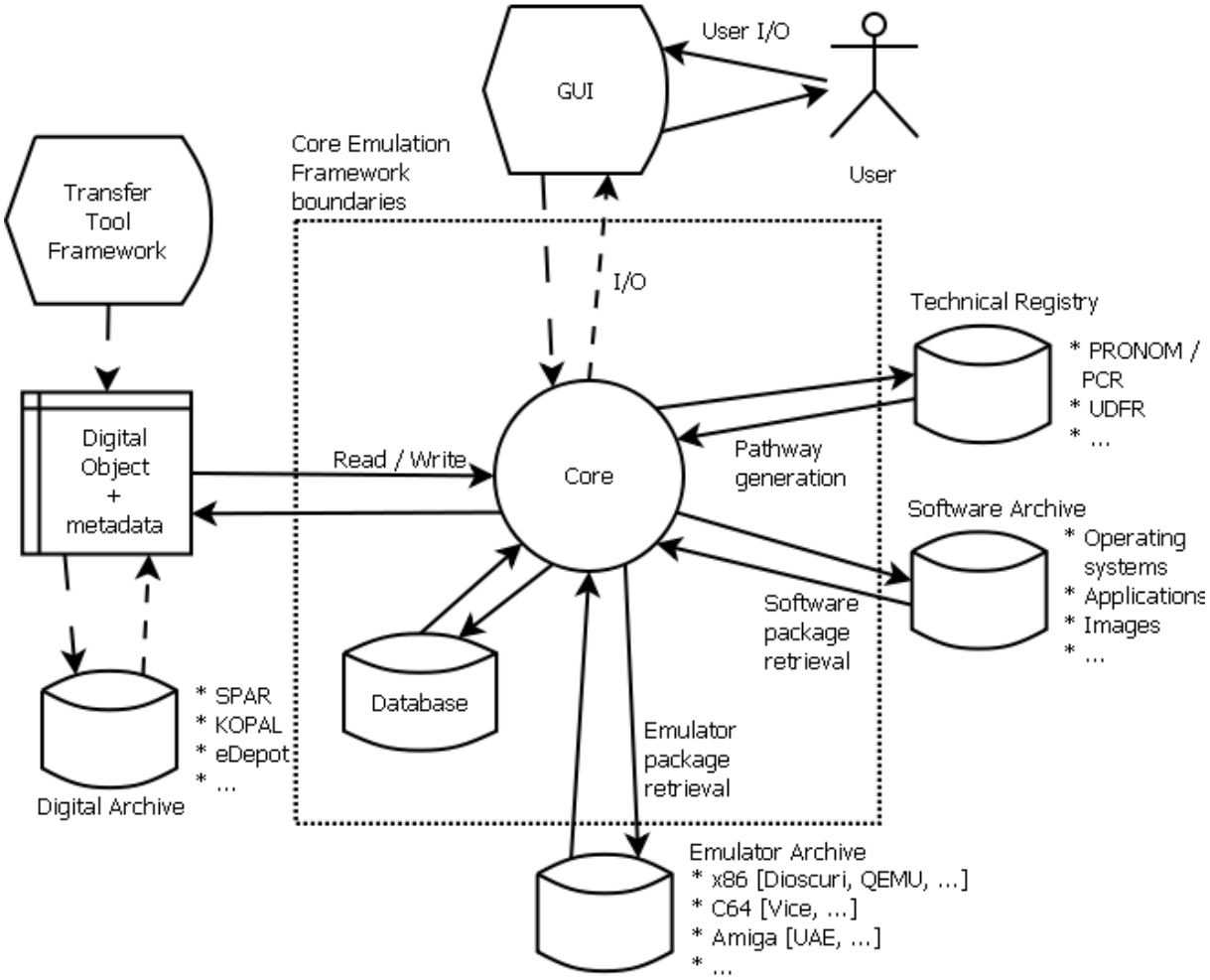


Figure 2: Emulation Framework system context diagram

4.1.1 Inputs

Figure 2 shows the external inputs of the Emulation Framework. Dashed arrows indicate optional inputs; these may not be necessary to complete the rendering of the digital object.

The human user will interact with the Emulation Framework via an externally developed GUI. This GUI will act as the intermediate for all input/output the user has for the system. The Core also allows for communication with automated environments; these can use the same API that the GUI uses.

The starting point for the EF is the digital object. This digital object is in the form of a computer file, and can be classed as two types: as an 'atomic' file (e.g. PDF file or Word document) or a 'compound' file (e.g. a compressed collection such as a ZIP/RAR file or a standardized image format such as an ISO 9660 image or cartridge image). The digital object may or may not include appropriate metadata describing it. This object is likely to come from a digital archive located at the memory institutions, such as DNB's Kopal, BnF's SPAR, or KB's e-Depot.

External technical registries provide the EF with a pathway describing the required hardware, software and application to render the digital objects. Examples of such registries are PRONOM[1] / Planets Core Registry or the UDFR[2].

[1] PRONOM at The National Archives, *http://www.nationalarchives.gov.uk/pronom/*

The EF renders the selected digital object using emulators. These emulators are downloaded from the Emulator Archive, an external database containing (certified) emulator packages. An emulator package contains executable files to run the emulator, and associated metadata describing the emulator's hardware, configuration, etc.

The pathway for rendering a digital object consists not only of a description of a hardware emulator, but also a description of software dependencies including operating system and application. This software stack is retrieved from an external Software Archive, either as a complete stack or as separate components. The Software Archive is likely to be located at the memory institution; although shown as a separate component in the above figure for clarity, it may be part of a single system comprising the digital archive, Emulator Archive and Software Archive.

4.1.2 Outputs

All inputs mentioned in section 4.1.1 also act as outputs. Again, dashed arrows indicate optional outputs.

Any user interaction with the EF will be shown as updates in the GUI. Also, any emulator outputs, such as audio and video, are also returned to the GUI. Other functionality such as capturing screenshots will also be presented to the user via the GUI.

The digital object may not only be read from but also written to. This allows the user to save any progress made.

The technical registries will need to be supplied with either the full digital object or a signature uniquely identifying it, for them to look up pathway information.

Prior to downloading emulator packages, one or more queries will be sent to the Emulator Archive for a list of available emulators.

Similarly, the software archive will need to be queried for available content before software can be retrieved from it.

4.1.3 Major System Functionality

The functionality of the EF is to provide the user with the best available environment to render the digital object. This can be broken down into several steps, from set-up to usage. During setup, an administrator installs and configures the EF. This is described in more detail in [SMG] and [SUG].

An end-user can then start the EF, providing it with a digital object which may include identifying and/or characterising metadata. The EF may need to contact an external registry to obtain more information about the digital object; it will provide the registry with the required information identifying the digital object (which may vary between registries), and request possible rendering pathways. These pathways will be presented to the user for selection and/or configuration; not all pathways may be viable depending on the available emulators and software. Once the configuration and selection is complete, the pathway will be used to set up the environment for the digital object. The environment usually consists of the digital object, the rendering application and operating system plus any dependencies such as plug-ins or fonts, and the emulator.

From this point on the EF will act as an intermediary between the user and the underlying emulator, providing several functions independent of the selected emulator such as audio, screen and video capture.

Figure 3 shows the workflow for the above steps.

[2] United Digital Formats Registry, *http://www.gdfr.info/udfr.html*

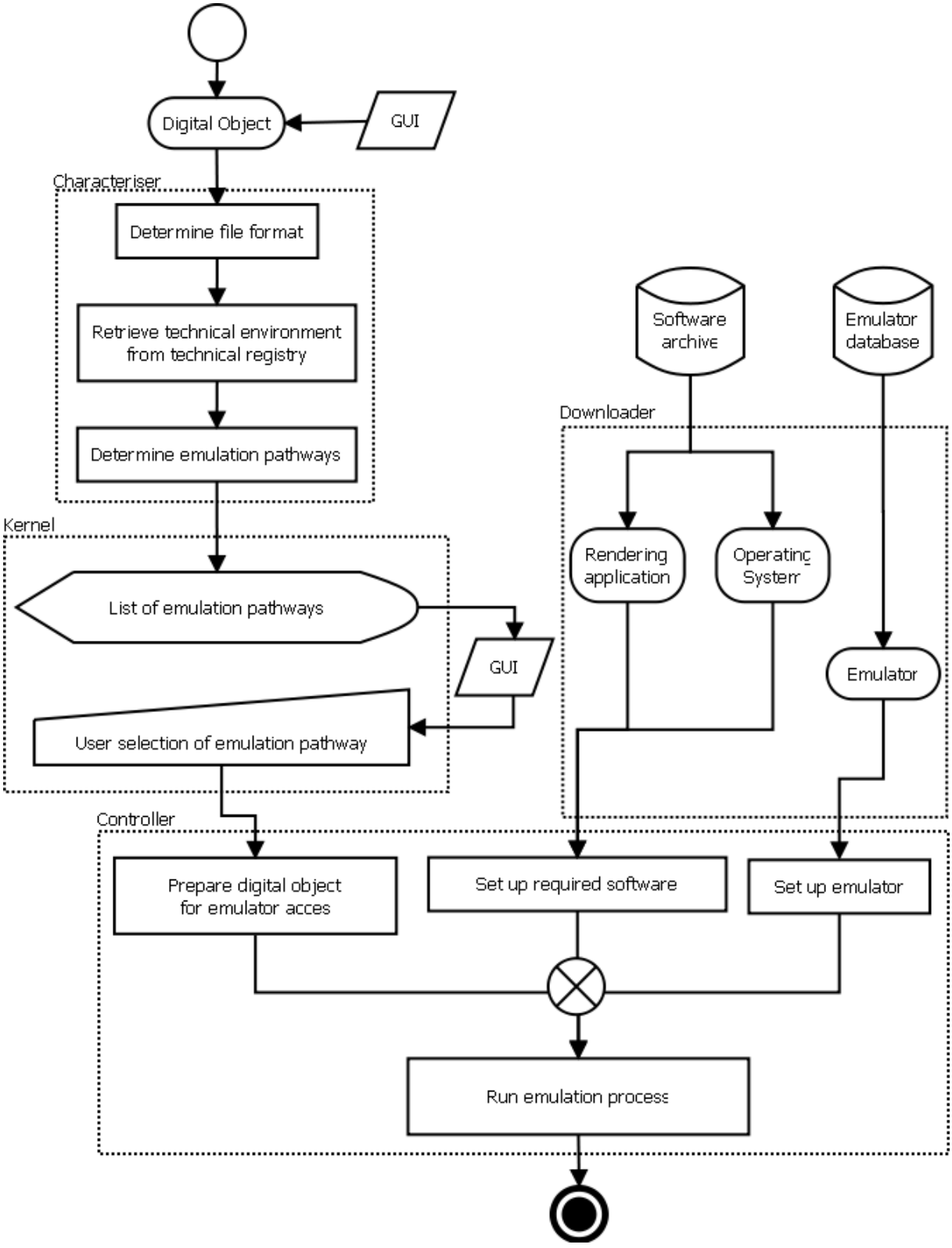


Figure 3: EF 'normal use' workflow diagram

The dotted boxes around the workflows indicate the components responsible for the logic of the workflow step.

4.2 Architectural Summary

The architecture of the main components in the system is shown in the UML Component diagram below.

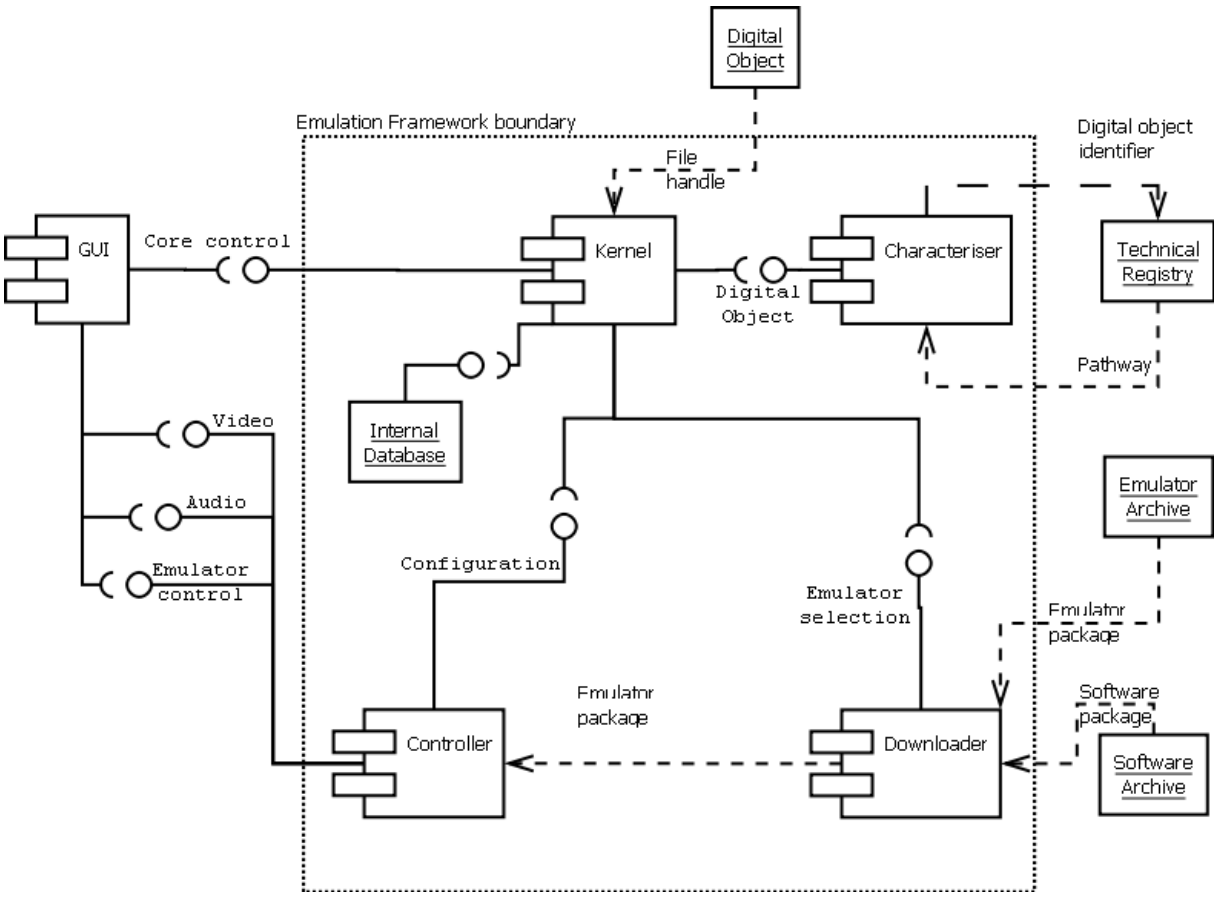


Figure 4: Core UML Component diagram

The socket connections show interfaces between components, while dashed arrows indicate dependencies of components on files or databases. Components outside the EF boundaries are not part of the system to be developed, but the EF relies on some of these to provide full functionality.

Communication between the EF and external components will occur according to the external component's interface. These interfaces are included in this document as a reference.

This diagram is a high-level overview of the system components, and as such may not include all connections between the different components. Detailed sub-level components and their functionality will be outlined in the following sections.

5 System Components

The sections outline the main packages in the Core code base. A high-level description of the package functionality, followed by a list of classes fulfilling the separate functions and any connection to external interfaces, is given.

5.1 Kernel

5.1.1 Business Logic

The Kernel is the central component of the Core. It connects the internal components and provides the external interface. This interface provides a path for the communication between the external input and the component's outputs.

The Kernel component will implement the following functionality:

- Provide the input interface
- Provide an observer interface
- Handle GUI/automated environment input
- Delegate work to other components

The implementation of a central component with well-defined communication between internal components allows a modular structure to be built, and allows for easy replacement of internal components. This modular structure also simplifies adding new functionality over time.

As such, the Kernel is intended to translate high-level functionality into the required collection of calls required for underlying components to complete the request. For example, a request to generate pathways will result in the Kernel calling functionality to characterise the file, generate a digital object identifier, ask the Characteriser to contact the technical registries and return the resulting list of pathways.

5.1.2 Class Descriptions

The Kernel package consists of the following classes (all part of the eu.keep.kernel package):

Class name	Class type	Class description
CoreEngineModel	Interface	Model interface for the Model-View-Controller design pattern. Provides the external interface for GUI/automated environments to control the EF
CoreObserver	Interface	Observer interface for the Model-View-Controller design pattern. The Core Emulator Framework's model (in the kernel module) will send out updates when the model is changed.
Kernel	Class	Main class of the Core Emulation Framework. Implements the model (in CoreEngineModel) for the MVC pattern. Delegates work to other components. As such, contains class variables for the other components such as Controller, Characteriser, etc.

5.1.3 External Interfaces

The Kernel component provides an interface for the communication between the GUI/automated environment and the Core. As business logic is separated from input and presentation, the interface follows the Model-View-Controller (MVC) architectural pattern. Examples of frameworks in Java that implement MVC are Java Swing[3] for GUI frameworks and Spring[4] or Struts[5] as (web-based) frameworks.

The Core model provides the following functionality:

Initialisation

- Retrieve Core property settings
- Register and remove observers
- Initialise the Core
- Stop the Core

Characterisation

- Characterise a digital object
- Retrieve a list of pathways for a file format
- Determine if the technical environment in a pathway can be build

Emulation

- Get and set emulator configuration options
- Prepare and generate the emulator configuration
- Match a (list of) emulators with a (list of) software images
- Find emulators that match a given pathway
- Find software that matches a given pathway
- Automatically select an appropriate pathway, format, software image or emulator
- Analyse the metadata accompanying a digital object
- Start an emulator based on a digital object (optionally including metadata or pathways)

Technical Registries

- Configure the registry list based on user input

The full specification of the Model is included in the section External components

The Core provides the above specification in public methods; this means that GUIs/automated environments also written in Java can use the Core as library, i.e. an external jar file, to call the required methods.

[3] http://java.sun.com/javase/6/docs/technotes/guides/swing

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

[5] http://struts.apache.org/

5.2 Characteriser

5.2.1 Business Logic

The Characteriser provides functionality to connect to external technical registries for collecting metadata and pathways of user-selected digital objects.

The Characteriser provides the following functionality:

- Characterise a digital object
- Add/remove repositories
- Connect to external repositories and retrieve digital object metadata / pathways
- Convert external repository information from custom format into internal, common format

The Characteriser implements the APIs provided by external registries to retrieve digital object metadata. This will include functionality to convert the retrieved metadata into a generic internal format.

Currently the default technical registry is the prototype developed as part of the Software Archive; the only available external registry is PRONOM, and a proof of concept has shown that the Core can interface with this registry. The Characteriser uses the freely available FITS tool for characterisation of digital objects. PRONOM and FITS are described in more detail in section 7.

A flowchart of the functionality is depicted in the following figure:

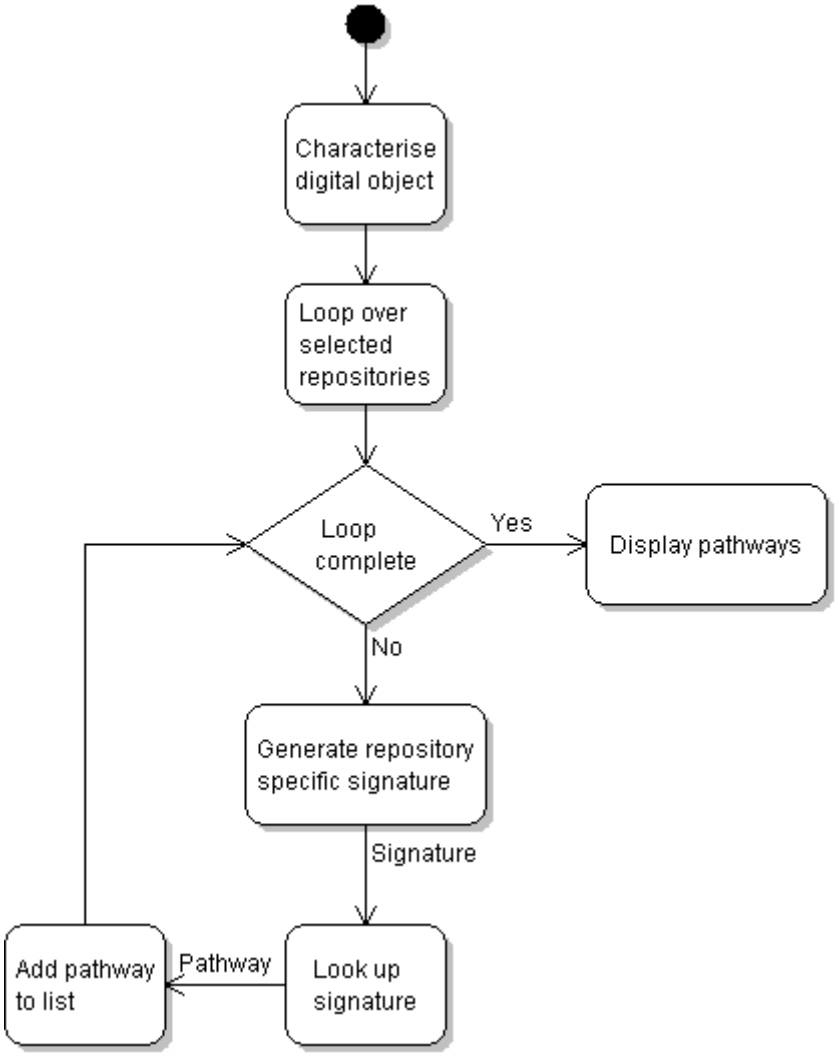


Figure 5: Characteriser flowchart

The characteriser has an internal catalogue of repositories containing the name, version, a short summary and URL of each repository. The user selects which repositories are queried for pathways for the digital object.

For each external repository, a class will implement the generic Characteriser repository interface, which contains common methods expected to be supported by the repositories (generate identifier, lookup identifier, etc.) This allows the system to loop over the implemented repositories and query them using generic methods.

When the system looks up a digital object using the specific implementations of the generic interface, the results are amalgamated. The repository-specific pathways are converted to an internal Core format, which is used by other Core components. During the conversion, the system will also try to remove any discrepancies between repository results.

The final selection is passed back to the Kernel.

5.2.2 Class Descriptions

The Characteriser package consists of the following classes (all part of the eu.keep.characteriser package):

Class name	Class type	Class description

Characteriser	Class	Holds registry functionality, such as importing registries into the database, enabling registries for lookup, and characterisation functionality, such as characterising files, looking up pathways
FitsTool	Class	A wrapper class for the File Information Tool Set (FITS)[6] tool. It permits the identification and characterisation of file formats using different tools, reports conflicts.
Format	Class	Represents a file format that contains information about its name, MIME type and a list of the reporting tools. The Format object is represented by the file format name, the MIME type and the list of reporting tools that were used to identify the file format
sub package registry		
PronomRegistry	Class	Represents PRONOM's implementation of the Registry interface. Currently a stub since the registry has not been updated yet for allowing this interaction between the two systems.
Registry	Interface	Represents the Registry interface for technical registries that are supposed to hold information about file formats and associated technical environments with digital preservation, i.e. migration and emulation in mind.
UDFRRegistry	Class	Represents UDFR's implementation of the Registry interface. Currently a stub since the registry has not been updated yet for allowing this interaction between the two systems.

The characteriser uses a Pathway class, generated from the XSD schema in the Software Archive, to represent the emulation pathway or viewpath. Such a pathway consists of a structured description of the complete hardware and software stack needed to render a digital object. This usually consists of four layers (file format, rendering application, host Operating System and finally an hardware platform), but may have more or less, e.g. some console game do not need the rendering application since it is usually bundled with the OS.

5.2.3 External Interfaces

See section 7 for APIs to external registries.

5.2.4 Libraries

The Characteriser uses the FITS tool, created by the Harvard University Library Office for Information Systems. It “identifies, validates, and extracts technical metadata for various file formats. It wraps several third-party open source tools, normalizes and consolidates their output, and reports any errors.”[7]

See section 7.5.1 for more details on FITS.

[6] http://code.google.com/p/fits/

[7] Ibid

5.3 Downloader

5.3.1 Business Logic

The Downloader component handles all functionality related to retrieving external data. This includes emulator packages, software packages, and pathway information.

Emulator packages (which contain specific metadata) can be retrieved from an external database. The emulators are then available for use within the Core; the metadata is used to determine which emulator can fulfil the pathway requirements.

Software packages are retrieved based on pathway information from an external database. These are provided to the Controller for emulator configuration.

Both emulator and software packages are retrieved when required and not stored locally. This means that a connection to an Emulator Archive and Software Archive is required for full functionality.

Registry configuration is stored in the internal database. This data can then be accessed and edited to configure the registry lookup in the Characteriser component.

The Downloader provides the following functionality:

- Store and update configuration of Technical Registries
- Retrieve information and emulator packages from, and download emulator binaries from an Emulator Archive
- Retrieve information and software packages from, and download software images from a Software Archive

To contact, retrieve, and download packages/images from the Emulator Archive and Software Archive, the Downloader implements the API provided by these services.

5.3.2 Class Descriptions

The Downloader package consists of the following classes (all part of the eu.keep.downloader package, with sub-packages indicated):

Class name	Class type	Class description
DataAccessObject	Interface	Database interface for storing information in the local database
Downloader	Class	The Downloader component handles all functionality related to downloading and storing external data. This includes emulator packages, software packages, and registry configurations.
EmulatorArchive	Interface	Interface to the Emulator Archive, the webservice providing emulator package metadata and binary files
SoftwareArchive	Interface	Interface to the Software Archive, the webservice providing software package metadata and binary files
sub package db		
DBRegistry	Class	Object representation of Registries stored in the local database

DBUtil	Class	Common database utilities (setting up / tearing down a connection, etc.)
EmulatorArchivePrototype	Class	A prototype implementation of the EmulatorArchive, acting as client for Emulator Archive webservices.
H2DataAccessObject	Class	H2 database implementation of the DataAccessObject interface.
SoftwareArchivePrototype	Class	A prototype implementation of the SoftwareArchive, acting as client for Software Archive webservices.

5.3.3 External Interfaces

The prototype classes for the Emulator Archive and Software Archive make use of WSDLs, Web Services Description Language files, an XML based protocol for information exchange in decentralized and distributed environments, to generate the relevant APIs. These WSDLs are included in section 7.

The local database tables are described in section 5.6.1.

5.4 Controller

5.4.1 Business Logic

The Controller configures and runs the emulator selected by the Kernel based on the pathway. It selects and sets up the correct emulator configuration, based on technical environment metadata. It presents the emulator's video and audio output and accepts input for controlling the emulator. Although currently not implemented, it is intended that audio and video streams from the emulator are made accessible so they can be recorded.

The Controller component will implement the following functionality:

- Configure, start and run selected emulators
- Handle user input to emulator
- Present video and audio results of emulator to user
- Load user-selected digital object into emulator

5.4.2 Class Descriptions

The Controller package consists of the following classes (all part of the eu.keep.controller package, with sub-packages indicated):

Class name	Class type	Class description
Controller	Class	Entry point for the Controller package. This class contains methods to retrieve files (BLOBs) such as emulator packages from the local database and accept software images
ConfigEnv	Class	Holds information about the configuration

		environment
sub package emulatorConfig		
FMTemplateHelper	Class	Template helper class for Freemarker implementation. Contains general template functionality
SimpleTemplateBuilder	Class	Class to build an emulator's configuration using a simple template. Implements TemplateBuilder
TemplateBuilder	Interface	This interface defines the methods that the different classes of template builders (simple for basic configuration such as 'autorun', 'floppy disks'; complex for 'memory size', 'cpu bits', etc.) should implement to be able to generate a configuration for an emulator.
sub package emulatorRunner		
EmulatorProcessManager	Class	Manager of emulation processes that keep tracks of currently running processes
NativeEmulatorRunner	Class	Class used for starting/stopping an emulation process. Uses the NativeRunner class to launch a native executable and uses the owner (parent class) to start and kill the newly created process.
NativeRunner	Class	Class for starting executables as external processes in new threads

5.4.3 Configuring emulators

Using a Java classes via a builder pattern (deprecated)

Initially, configuring emulators was done using custom Java classes written specifically for each emulator (and version). These were applied using a builder pattern to create a specific emulator configuration. Each of these configuration classes had to implement an interface class describing the different methods used by the core, e.g. build the command line string (to be called subsequently by the NativeRunner class), or wrap bare digital objects into an appropriate disk image.

These Java classes were part of the emulator packages, and not directly part of the framework. It was thought that this approach was slightly cumbersome as it required emulator developers to write custom Java classes, as well as the framework importing class files from the emulator packages. A method where emulator configurations could be added dynamically (i.e. at runtime) without having to change (i.e. add new classes) and recompile the code was needed.

Using template processing

As an alternative to the builder pattern, templates, a processing element that can be combined with a data model and processed by a template engine to produce a result document, were used. This allowed emulator developers to write their custom templates in pseudo-code (rather than Java code), and offered greater flexibility by reading these templates at run-time when unpacking and configuring an emulator. Figure 6 illustrates the processing flow of a template engine. It consists of a data model, which is a source of preformatted data to be used in the result; a template, which contains the output format of the data; a template engine, which combines the data and the template to produce the result,

a document specifically formatted according to the template containing the data from the data model.

In the EF, each emulator package will provide a template, which combined with the data model in the EF, and a specific Java template engine, will result in an emulator specific configuration file

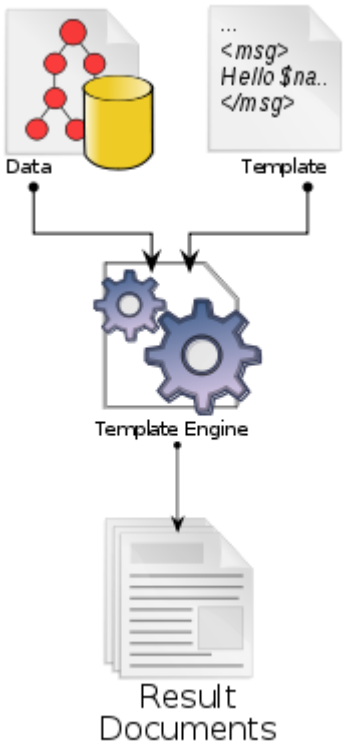


Figure 6: A diagram illustrating all of the basic elements and processing flow of a template engine

Template data model

A visualisation of the data model used in the EF is as follows:

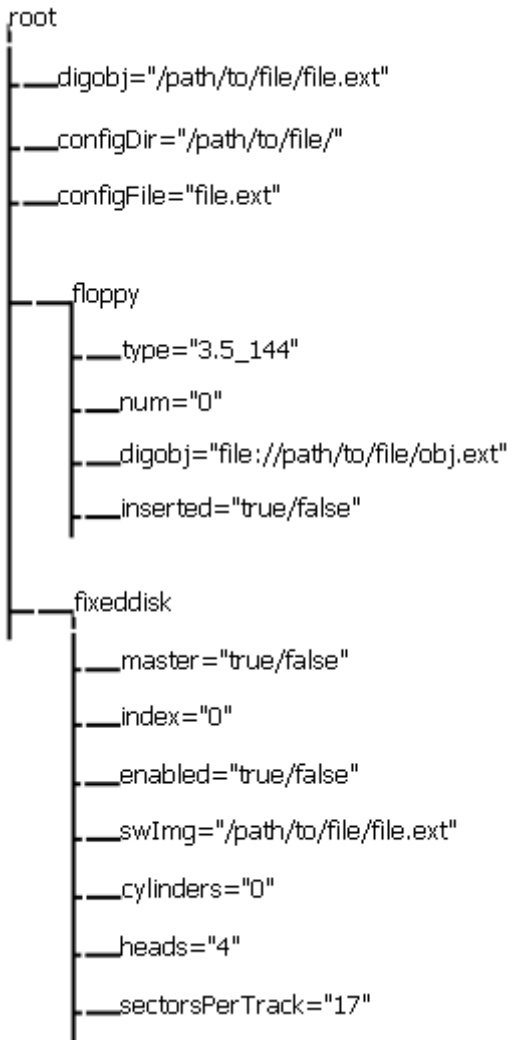


Figure 7: Visualisation of Emulation Framework template data model

Each variable has a string value; some of the branches shown are complex structures (of which there may be more than one of each type within the model) which consist of multiple string values. The emulator template can use any of these variables to generate the necessary configuration file, be it in command-line form, XML format, or as properties file. Not all variables may need to be set (for example, many console emulators have no notion of the drive parameters or contain a fixed disk), but may use a subset of the above.

It should be noted that only the above variables can be used as configurable values for the emulator configuration; however, this should not stop the configuration from containing emulator-specific options based on the values of the variables.

An example of use is given below.

StringTemplate (deprecated)

The first template engine used was the StringTemplate engine[8]. This template was chosen for its simplicity, but also for its use of template interfaces and groups. The former forced template files to adhere to a pre-defined template interface, ensuring robustness by making all emulator configuration templates contain specific 'methods' and signatures. Any template errors could be caught early in the process by matching the template to the interface.

[8] http://www.stringtemplate.org/

The latter allowed specific parts of the template to be called on demand; one of the uses for this was separating the template into 'preamble', 'body', and 'postscript' sections. This allowed specific order in the template, which could be passed to the EmulatorRunner class. An example here is that Java emulators require 'java –jar' to be given prior to the executable, and other options such as '-Xmx1024m' after the executable (e.g. 'java –jar Dioscuri.jar –Xmx1024m'). Passing the template in sections to the runner class allowed it to group these in the right order, since the executable was retrieved from the emulator package itself, not the template.

However, it was soon discovered that the StringTemplate engine uses hardcoded separator values when it is passed a list of template directories, which caused the loading to fail on certain environments (notably absolute paths in Windows containing the ':' and '/' characters). It was decided to abandon the library rather than try to work around these issues.

Freemarker

As a substitute to StringTemplate, the Freemarker template engine was chosen[9]. Freemarker is another relatively simple but popular template engine that does not seem to suffer from the above path loading issue. However, it does not support template interfaces nor template groups, so the templates had to be re-written to be able to support the sectioning; it was decided that the interface support was a nice feature but the lack of it not a showstopper. In any case, the Freemarker engine has an 'attribute' directive allows some level of self-description to be defined within the template.

To support sectioning, each template uses a specific string ('##Section: ${section}##') to indicate the start of a section. The output result is split based on these strings.

An example template is as follows:

```
<#ftl attributes={"configDir":"configDir", "configFile":"configFile",
"digobj":"digobj", "fixedDisks":{"enabled":"enabled",
"index":"index", "master":"master", "cylinders":"cylinders",
"heads":"heads", "sectorsPerTrack":"sectorsPerTrack",
"swImg":"swImg"}, "floppyDisks":{"type":"type", "num":"num",
"digobj":"digobj", "inserted":"inserted"}}>
<#-- Vice 2.2 configuration template (CLI) -->

<#-- Floppy drive letter definition -->
<#assign floppyDriveLetter = {"0":"8", "1":"9", "2":"10", "3":"11"}>

<#-- Drive type definition -->
<#assign driveTypes = {"525_720":"unsupported drive",
"3.5_144":"unsupported drive", "C64_1541":"1541", "C64_1741":"1741"}>

<#-- Seperator macro -->
<#macro separator section="undefined">
##Section: ${section}##
</#macro>

<#-- Floppy drive macro -->
<#macro floppyDisk item>
<#if item.type?has_content >
-drive${floppyDriveLetter[item.num]}type
${driveTypes[item.type]}
-${floppyDriveLetter[item.num]}
"${item.digobj}"
</#if>
<#if item.inserted == "true">
+truedrive
```

[9] http://freemarker.sourceforge.net/

```
<#else>
-truedrive
</#if>
</#macro>

<#-- Start of preamble -->
<@separator section="preamble"/>
<#-- Start of body -->
<@separator section="body"/>
-autostart
"${digobj}"
<#list floppyDisks as floppy>
 <@floppyDisk item=floppy/>
</#list>
<@separator section="postscript"/>
<#-- Start of postscript -->
```

This template defines its structure in the #ftl tag, which corresponds to the data model. It then contains two hashes for looking up emulator-specific values for the floppy drive letter and drive types ('floppyDriveLetter' and 'driveTypes'); the EF uses a 0-based index to define floppy drives based on the number of digital object requiring floppy drives and specific strings for the drive type. After that it defines two macro's that can be called as functions (with variables) in the template body ('separator' and 'floppyDisk'). Using a macro will produce a similar section containing different values. This can either be called multiple times in the template itself (e.g. 'separator'), or the data model may define multiple sequences for a variable (e.g. 'floppyDisk').

Although this template defines the full data model, it only uses the variables 'digobj', and 'floppy.type', 'floppy.num', 'floppy.digobj', 'floppy.inserted'. The rest may be set but are not used in the template.

The EF will query the template for the data model (retrieving it from the ftl tag), which will serve as the basis of the configuration. It contains some logic to generate default configurations based on the emulator environment, digital objects, configuration directories and software images, or it can ask the user to fill in the values.

Once a data model has been assigned values, it can be passed to the template to produce the required results, as shown below:

Data model:

```
{fixedDisks=[], floppyDisks=[{inserted=true, num=0, digobj=test
Data\IKPlus.d64, type=C64_1541}, {inserted=false, num=1, digobj=test
Data\arkanoid.d64, type=C64_1741}],
root=[{configFile=noConfFileDefined, digobj=test Data\IKPlus.d64,
configDir=.\cef\exec\3831e3da-afa6-4709-8f3b-a47878069dd0}
```

Output:

```
##Section: preamble##
##Section: body##
-autostart
"test Data\IKPlus.d64"
-drive8type
1541
-8
"test Data\IKPlus.d64"
+truedrive
-drive9type
1741
-9
"test Data\arkanoid.d64"
```

```
-truedrive
##Section: postscript##
```

This output will be parsed by the EF into three sections, of which only the body section will contain any data.

5.4.4 Packaged emulators

Several emulators have been included in the EF; for several others, there were issues which prevented these from successfully including them in the EF. The table below summarises the progress

Emulator	Operating System	Version	Included	Notes
Beebem[10]	Linux	0.0.13	Yes	
	Windows	4.13	Yes	
Dioscuri[11]	Linux Windows	0.5.0 0.6.0	Yes	
JavaCPC[12]	Linux Windows	6.7	No Yes	Linux version does not work due to bug in Java sound routines. Does not support automatically starting an image; images are attached properly but must be started manually in the emulator
QEMU[13]	Linux	0.13.0	Yes	Statically compiled
	Windows	0.9.0	Yes	
UAE[14]	Linux	0.8.29	Yes	
	Windows	1.6	Yes	
Vice[15]	Linux	2.2	Yes	
	Windows	2.2	Yes	

[10] http://www.mkw.me.uk/beebem/

[11] http://dioscuri.sourceforge.net/

[12] http://sourceforge.net/projects/javacpc/

[13] http://wiki.qemu.org/Main_Page

[14] http://www.amigaemulator.org/

[15] http://www.viceteam.org/

5.4.5 External Interfaces

5.5 General purpose packages

There are several packages that serve general purpose functionality. Most notably these are the Core package, which serves as a command-line front-end for running, testing and debugging the Core from the command line, and the Util package, which contains common file and XML utilities used in other packages.

The Core package makes use of BeanShell[16], a "small, free, embeddable, source level Java interpreter with object based scripting language features…BeanShell dynamically executes standard Java syntax and extends it with common scripting conveniences such as loose types, commands, and method closures like those in Perl and JavaScript"[17]

5.5.1 Core class Descriptions

The Core package consists of the following classes (all part of the eu.keep.core package):

Class name	Class type	Class description
CandidatesCompletionHandler	Class	JLine support class
ClassAndFileCompletor	Class	JLine support class
ClassMap	Class	A class that holds all classes, and their methods, in the classpath. Used for auto-completion
EFCliAutoComp	Class	A class to test the EF through a command line interface (CLI). The CLI accepts valid Java code in a dynamic, scripted fashion by using BeanShell's <code>Interpreter</code> class. The commands entered by the user are read using the 3rd party library JLine. Implements the CoreObserver interface

5.5.2 Util class Descriptions

The Util package consists of the following classes (all part of the eu.keep.util package):

Class name	Class type	Class description
DiskImage	Abstract	Common utilities used to generate a disk image
DiskUtilities	Class	Common shared disk image utilities, such as CHS calculation, etc
EmulatorSandbox	Class	Modified SecurityManager for running emulator in a sandbox environment where certain actions are disabled.

[16] http://www.beanshell.org/

[17] Ibid

FileUtilities	Abstract	Common shared file utilities, such as copy, unzip, etc
FloppyDiskImage	Class	Utilities to generate a floppy disk image. Currently supports very simplistic 1.44 MB FAT12 floppy disks. Information taken from http://en.wikipedia.org/wiki/FAT_16 and Linux's mkdosfs
FloppyDiskType	Enum	Enumeration of the different floppy disk types supported
VariableFixedDiskImage	Class	Utilities to generate a fixed disk image. Currently supports very simplistic FAT16 fixed disks, of varying size. Information taken from http://en.wikipedia.org/wiki/FAT_16 and Linux's mkdosfs
XMLUtilities	Class	Common shared XML utilities, such as validate, marshall/unmarshall, etc

5.6 Data Access Layer

There are several places where the Core accesses data.

Locally there is a database containing metadata used by the Core.

Externally, there are repositories that the Characteriser component accesses for retrieval of pathways and technical environment metadata. These databases, such as PRONOM/Planets Core Registry or UDFR, are expected to have an open API available that can be implemented in the Characteriser. These APIs are described in section 7.

There are also expected to be external databases of (certified) emulator packages, the Emulator Archive, and software images, the Software Archive. The Downloader will contact these to download the relevant data for use within the framework. APIs for these systems also described in section 7.

5.6.1 Core Database

This database contains data used within the Core. The database consists of several tables, listed below:

Table 1: Core Database tables

Table name	Description
Emulator_whitelist	'Whitelisted' emulators, those that are allowed to be used when rendering an environment
EF_PCR_fileformats	Conversion table from EF file format IDs to PCR file format IDs
Fileformats	File format IDs. A direct copy from the Software Archive file format table
PCR_fileformats	PCR file format ID table. Taken from PRONOM.
Registries	External registry metadata

Table 2: Emulator_whitelist table

Attribute	Data Type	Description
Emulator_ID	Large, unsigned integer	Unique emulator identifier from Emulator Archive
Emulator_descr	Variable length string (up to 500 characters)	Emulator description

Table 3: EF_PCR_fileformats table

Attribute	Data Type	Description
PCR_ff_ID	Variable length string (up to 16 characters)	Foreign key from PCR_fileformats table
EF_ff_ID	Variable length string (up to 16 characters)	Foreign key from EF_fileformats table

Table 4: Fileformats table

Attribute	Data Type	Description
Fileformat_ID	Variable length string (up to 16 characters)	File format ID
Name	Variable length string (up to 250 characters)	File format name

Table 5: PCR_Fileformats table

Attribute	Data Type	Description
Fileformat_ID	Variable length string (up to 16 characters)	File format ID
Name	Variable length string (up to 250 characters)	File format name

Table 6: Registries tables

Attribute	Data Type	Description
Registry_ID	Large, unsigned integer	Emulators primary key
Name	Variable length string (up to 255 characters)	Name of the registry
URL	Variable length string (up to 255 characters)	URL of the registry
Class_name	Variable length string (up to 255 characters)	Java .class file associated with registry
Translation_view	Variable length string (up to 500 characters)	Database view used for translating IDs
Enabled	Boolean	Registry is used during pathway lookup
Description	Variable length string (up to 255 characters)	Short description of the registry
Comment	Variable length string (up to 255	Short comment of the registry

	characters)	

6 Prototypes

The design of the EF is based around a sequence of prototypes, where each subsequent iteration includes more functionality. The final product should provide the functionality of the EF as described in [URD].

This approach was chosen to facilitate design, allow simple implementation to show where bottlenecks lie, and defer implementation until user requirements are properly elicited.

[DoW] mandates the release of two prototypes:

Sequence	Details	Delivery date
1	Prototype of working EF with set of emulator modules for hardware components of the selected target environments	M19 (August 2010)
2	Second prototype of EF with set of emulator modules for hardware components of the selected target environments	M24 (May 2011)

The Scrum Product Backlog [SPB] estimates a larger number of prototypes, as there have be around 10 four-week sprints, each resulting in a working prototype. It is expected a small number of these will be publicly released, of which two will be the [DoW] mandated prototype releases named above.

7 External components

7.1 Communication protocols

The communication with the external components (Emulator Archive, Software Archive) is implemented using web services; more specifically, they are defined on the Application Layer using Simple Object Access Protocol (SOAP). In code a description of the operations is given using the Web Services Description Language (WSDL).

The use of WSDLs leaves the implementation of the communication free to use any style of web services: SOA or REST.

7.2 Emulator Archive

The Emulator Archive is a database containing certified packages of emulators. The certification indicates it is known to run successfully in the Emulation Framework, and the package contains the required metadata for configuration and pathway generation.

Such an archive should support at least the following:

- a web-service that takes a blank input and returns a list of emulators, including their version and an identification number for each emulator.
- a web-service that takes a blank input and returns a list of supported hardware, including an identification number for each hardware type.
- a web-service that takes an identifier of a hardware type as an input and returns a list of emulators, including their version and an identification number, which supports the hardware type.
- a web-service that takes an identifier of an emulator as an input and returns a binary file containing the emulator and metadata.

7.2.1 Emulator Archive prototype

For proof-of-concept and testing purposes, a simple prototype was written that supported the above functions. The accompanying WSDL is included in Appendix A: Emulator Archive WSDL

7.3 Software Archive

The Software Archive is a database containing hardware-specific software images. These images contain specific applications and libraries, which allow the Core to build a technical environment (hardware, operating system, application) for a digital object by choosing an emulator (hardware) and selecting an appropriate image (application, operating system) from the Software Archive. Linking it to the digital object then completes the pathway.

At the time of writing, no public software archives were known to exist. However, memory institutions may have digital depots that provide this functionality. Such an archive should support at least the following:

Contents

- a web-service that takes a blank input and returns a list of file formats, including a version number and an identification number for each file format.

- a web-service that takes a blank input and returns a list of software packages, including a version number and an identification number for each package.
- a web-service that takes a blank input and returns a list of operating systems, including a version number and an identification number for each OS.

File formats

- a web-service that takes an identifier of a file format as an input and returns a binary file containing the complete software stack, consisting of some or all of the software package, software package plug-ins, operating system and metadata.

Software packages

- a web-service that takes an identifier of a software package as an input and returns a binary file containing the software package and metadata.

7.3.1 Software Archive prototype

For proof-of-concept and testing purposes, a simple prototype was written that supported image selection based on application, operating system and image format. The software archive prototype is a simple database (the H2 database was used), that, together with an Apache CXF front-end, can serve software images from a remote location.

The accompanying WSDL is included in Appendix B: Software Archive WSDL

7.3.2 Software archive database tables

Table 7: Fileformats table

Attribute	Data Type	Description
Fileformat_ID	Variable length string (up to 16 characters)	File format ID
Name	Variable length string (up to 250 characters)	File format name
Version	Variable length string (up to 250 characters)	File format version
Description	Variable length string (up to 1000 characters)	Description of the file format
Reference	Variable length string (up to 500 characters)	Any references for more information on the file format

Table 8: Opsys table

Attribute	Data Type	Description
Opsys_ID	Variable length string (up to 16 characters)	Operating System primary key
Name	Variable length string (up to 250 characters)	Name of the OS
Version	Variable length string (up to 250 characters)	Version of the OS
Description	Variable length string (up to 500 characters)	Short description of the Operating System
Creator	Variable length string (up to 500 characters)	Operating system creator

Release_date	Variable length string (up to 500 characters)	Operating system release date
License	Variable length string (up to 500 characters)	Operating system license
Language	Variable length string (up to 500 characters)	Operating system language
Reference	Variable length string (up to 500 characters)	Any references for more information on the operating system

Table 9: Apps table

Attribute	Data Type	Description
app_ID	Variable length string (up to 16 characters)	Application primary key
Name	Variable length string (up to 255 characters)	Name of the application
Version	Variable length string (up to 255 characters)	Version of the application
Description	Variable length string (up to 255 characters)	Short description of the application
Creator	Variable length string (up to 500 characters)	Application creator
Release_date	Variable length string (up to 500 characters)	Application release date
License	Variable length string (up to 500 characters)	Application license
Language	Variable length string (up to 500 characters)	Application language
Reference	Variable length string (up to 500 characters)	Any references for more information on the application

Table 10: Platforms table

Attribute	Data Type	Description
platform_ID	Variable length string (up to 16 characters)	Platform primary key
Name	Variable length string (up to 255 characters)	Name of the platform
Description	Variable length string (up to 255 characters)	Short description of the platform
Creator	Variable length string (up to 500 characters)	Platform creator
Production_start	Variable length string (up to 500 characters)	Platform production start date
Production_end	Variable length string (up to 500 characters)	Platform production end date

Reference	Variable length string (up to 500 characters)	Any references for more information on the platform

Table 11: Disk images table

Attribute	Data Type	Description
image_ID	Variable length string (up to 16 characters)	Disk image primary key
Description	Variable length string (up to 255 characters)	Short description of the disk image
imageformat_id	Variable length string (up to 16 characters)	Reference to imageformats table
platform_id	Variable length string (up to 16 characters)	Reference to platforms table
image	BLOB	Binary file of the disk image

Table 12: Disk image file system architecture (format) table

Attribute	Data Type	Description
imageformat_ID	Variable length string (up to 16 characters)	Image primary key
Name	Variable length string (up to 250 characters)	Name of the format

Table 13: Disk image BLOBs table

Attribute	Data Type	Description
image_ID	Foreign key	Reference to images table
image	BLOB	Disk image binary

Table 14: Fileformats to Application table

Attribute	Data Type	Description
fileformat_ID	Foreign key	Reference to fileformats table
app_id	Foreign key	Reference to apps table

Table 15: Fileformats to Operating Systems table

Attribute	Data Type	Description
fileformat_ID	Foreign key	Reference to fileformats table
opsys_id	Foreign key	Reference to opsys table

Table 16: Fileformats to Platforms table

Attribute	Data Type	Description
fileformat_ID	Foreign key	Reference to fileformats table
platform_id	Foreign key	Reference to platforms table

Table 17: Applications to Operating Systems table

Attribute	Data Type	Description
app_ID	Foreign key	Reference to apps table
opsys_id	Foreign key	Reference to opsys table

Table 18: Applications to Images table

Attribute	Data Type	Description
app_ID	Foreign key	Reference to apps table
image_id	Foreign key	Reference to images table

Table 19: Operating Systems to Platforms table

Attribute	Data Type	Description
opsys_ID	Foreign key	Reference to opsys table
platform_id	Foreign key	Reference to platform table

Table 20: Operating Systems to Images table

Attribute	Data Type	Description
opsys_ID	Foreign key	Reference to opsys table
image_id	Foreign key	Reference to images table

7.4 Technical registries

The Emulation Framework may use several registries to gather pathway information and/or technical environment metadata if the metadata provided with the digital object proves insufficient. These registries should support at least the following:

Contents

- a web-service that takes a blank input and returns a list of file formats, including a version number and an identification number for each format.
- a web-service that takes a blank input and returns a list of software packages, including a version number and an identification number for each package.
- a web-service that takes a blank input and returns a list of operating systems, including a version number and an identification number for each OS.
- a web-service that takes a blank input and returns a list of hardware types, including a version number and an identification number for each type.
- a web-service that takes a blank input and returns a list of emulators, including a version number and an identification number for each emulator.

File formats

- a web-service that takes an identifier of a file format as an input and returns a list of software packages, including their version and an identification number, which support the file format.
- a web-service that takes an identifier of a file format as an input and returns a list of operating systems, including their version and an identification number, which support the file format.
- a web-service that takes an identifier of a file format as an input and returns a list of hardware types, including their version and an identification number, which support the file format.
- a web-service that takes an identifier of a file format as an input and returns a list of emulators, including their version and an identification number, which support the file format.

- a web-service that takes an identifier of a file format as an input and returns a list of pathways, consisting of some or all of the software packages, hardware types, and emulators, including their version and an identification number, which support the file format.

Software packages

- a web-service that takes an identifier of a software package as an input and returns a list of operating systems, including their version and an identification number, which support the software package.
- a web-service that takes an identifier of a software package as an input and returns a list of hardware types, including their version and an identification number, which support the software package.
- a web-service that takes an identifier of a software package as an input and returns a list of emulators, including their version and an identification number, which support the software package.
- a web-service that takes an identifier of a software package as an input and returns a list of pathways, consisting of some or all of the hardware types and emulators, including their version and an identification number, which support the software package.

Operating systems

- a web-service that takes an identifier of an operating system as an input and returns a list of hardware types, including their version and an identification number, which support the operating system.
- a web-service that takes an identifier of an operating system as an input and returns a list of emulators, including their version and an identification number, which support the operating system.
- a web-service that takes an identifier of an operating system as an input and returns a list of pathways, consisting of some or all of the hardware types and emulators, including their version and an identification number, which support the operating system.

Hardware types

- a web-service that takes an identifier of a hardware type as an input and returns a list of emulators, including their version and an identification number, which support the hardware type.

Whether reverse lookups of the pathways are also required (e.g. given an emulator identifier, return the hardware types it supports) will be determined at a later stage.

7.4.1 PRONOM / Planets Core Registry

PRONOM is an on-line information system about data file formats and their supporting software products. Originally developed to support the access to and long-term preservation of electronic records held by the National Archives, PRONOM has been made available as a resource for anyone requiring access to this type of information.

The PRONOM API is described in several WSDLs[18]; however, the required methods above are in the process of implementation. When the API becomes available, it will be included here as a reference.

[18] http://gforge.planets-project.eu/gf/project/pronom/scmsvn/?action=browse&view=rev&revision=5

DROID (Digital Record Object Identification) is a software tool developed by The National Archives to perform automated batch identification of file formats. It is one of a planned series of tools utilising PRONOM to provide specific digital preservation services. DROID uses internal (byte sequence) and external (file extension) signatures to identify and report the specific file format versions of digital files. These signatures are stored in an XML signature file, generated from information recorded in the PRONOM technical registry.

7.4.2 UDFR

As of this writing, UDFR is still in the conceptual stage, and as such does not have a public API available yet.

7.5 Characterisation

7.5.1 FITS

The Core uses the open-source FITS tool. FITS in turn uses the following open source tools:

- Jhove[19]
- Exiftool[20]
- National Library of New Zealand Metadata Extractor[21]
- DROID[22]
- FFIdent[23]
- File Utility[24]

FITS also supplies two original tools

- FileInfo
- XmlMetadata

The FITS version used in the EF has been slightly modified: to enhance identification of (game) files, a modified DROID signature file including several formats was developed. This modified file only relies on the file extension for identification, and not on any other characteristics.

The following files reflect these changes:

- fits-0.4.1/xml/fits.xml
    - modified DROID signature file reference
- fits-0.4.1/tools/droid
    - added ‘d64’ file extension as Commodore C64 Disk Image

[19] http://hul.harvard.edu/jhove/

[20] http://www.sno.phy.queensu.ca/~phil/exiftool/

[21] http://meta-extractor.sourceforge.net/

[22] http://droid.sourceforge.net/

[23] http://schmidt.devlib.org/ffident/index.html

[24] http://unixhelp.ed.ac.uk/CGI/man-cgi?file

- added ‘t64’ file extension as Commodore C64 Tape Image
- added ‘adf’ file extension as Amiga Disk Image
- added ‘dsk’ and ‘sna’ as Amstrad Disk Image
- added ‘cdt’ as Amstrad Tape Image

8 Core API

As referenced in section 4.1.1, the Core should provide an API for communication. The user requirements state that the API should contain at least the following:

- a (web) service that takes a binary file and an XML metadata file as inputs and returns an acknowledgment on successful acceptance.
- a (web) service that takes a binary set of files and an XML metadata file as inputs and returns an acknowledgment on successful acceptance.

For user interaction, the Core will need to be controlled by a GUI. This will require some more specific methods than those mentioned above, and for those cases a more extensive API has been defined. However, as possible GUI interactions may include the methods above, both the machine and GUI API have been combined into one interface. This interface is fully specified in Appendix C: Core API

9 System-Wide Features

9.1.1 Language

As the EF is a back-end system, it is not intended to have a lot of language specific items in it. However, error messages from the Core may be passed to external systems, but these will consist of an identifier and a description, allowing for translation based on identifier outside the EF if necessary. The descriptions of errors, as well as any system logging, will be in English.

The EF also includes Observer functionality, modelled on the Observer design pattern to update observers of progress. Again these messages will be in English but include an identifier allowing for translation as outlined above.

9.1.2 Error-Handling

Any errors will be passed back to the external system for further handling and possible display. These error messages will consist of a unique identifier and an optional description.

9.1.3 Automated Debugging, Testing and Diagnostics

The major methods of classes will have their own automatic self-checking tests using the jUnit framework to enable frequent, extensive unit tests to be run throughout development. Each library will have a test harness user interface to enable it to be executed in isolation.

9.1.4 Speed and Capacity

As emulation can be very CPU-intensive, high-end hardware and optimisation of code can lead to better results. The EF is not concerned with these issues, however, as the emulators themselves are not part of the code to be developed and thus beyond the scope of this document.

Also, the EF is intended to run on a virtualised layer. Optimisations in this layer or during compilation of the EF for this layer will have far greater effect than optimising the EF code. Again, this virtual layer is beyond the scope of this document, so no special precautions will have to be made within the EF.

Connections to external systems will have time-out values associated with them to ensure the system returns within a specified time.

9.1.5 Statutory and Regulatory

The KEEP consortium is not aware of any Regulatory and Statutory requirements relating to this project. Legal studies regarding emulation have been carried out as part of the KEEP project and the design of the EF will adhere to these results. However, the EF is only a harness for running emulators and no specific investigation has been carried out to confirm the legality of it.

10 Development Environment

10.1 Development language

The choice of development language is limited to Java or C/C++ [DoW]. Both are both object-oriented libraries, and the difficult parts of the project seem equally difficult in both languages: controlling video/audio output of separate processes (external executables) seems to be very difficult, if not impossible for any of today's language/technology.

It is expected that C/C++ is easier to compile with a KEEP VM modified GCC. This would allow running the Core on a virtualisation layer at an earlier stage. C/C++ is a much lower level language than Java, however, and this means due to higher overheads it will require more effort to produce a full framework.

Although it is expected that porting Java to the KEEP VM is a more difficult task, this is one of the expected outcomes of the KEEP project [DoW]. Although it was noted that GCC is not yet entirely compatible with Java's graphic libraries (Swing/AWT), it is expected that a complete operating system such as Linux will be ported to the KEEP VM in the future, and this will have full support for Java. As such, a framework written in Java is also expected to run on the KEEP VM.

Moreover, Java has far more libraries available for web services, database access, characterisation, XML generation, etc. This low overhead means that it is far quicker to produce a framework with high functionality compared to C/C++. Coupled with the fact that the development team already had significant experience in Java, this would allow for far quicker development of the Core, delivering a rich framework at the end of development.

When the choice needed to be made, it was known that the KEEP VM would not support a full-fledged framework. The decision was then made that rather than limiting the framework's functionality so it could run on the KEEP VM in the short term, a full-fledged framework would be built that would not run on the KEEP VM until much later in the project.

Prior to being ported to the KEEP VM, Java has the added advantage that due to its high portability it can easily run on multiple platforms; compared to this, porting C/C++ is an intensive and tedious process. This allows demonstrations and dissemination of the EF on many platforms before the KEEP project is complete.

Given the above considerations, Java was chosen as the development language.

10.2 Internal database

The choice for internal database to be used within Java can be one of many simple databases such as H2[25], HSQL[26], Derby[27], or Firebird[28]. As complete emulator executables may be stored inside this database, it is important it has a good performance with large amounts of binary data, such as BLOBs.

Given its small footprint and integrated web-based database viewer, H2 was selected as internal database.

[25] http://www.h2database.com/html/main.html

[26] http://hsqldb.org/

[27] http://db.apache.org/derby/

[28] http://www.firebirdsql.org/

Nonetheless, a database abstraction layer should be developed that translates the development language calls to SQL; this should ensure that the code is independent of the choice of internal database, allowing for easy replacement if for any reason a particular database is preferred.

10.3 Choice of emulators

The requirements for the EF specify that it should be able to run at least two types of emulators, of which one is a Java-based emulator, and another is a KEEP VM based emulator.

Two emulators were chosen to fulfill these requirements: Dioscuri[29] and VICE[30].

Dioscuri was chosen as the team possessed extensive knowledge of the emulator. As it was expected that changes to the emulator may be needed to fully integrate it with the EF, this knowledge could prove invaluable. Furthermore, Dioscuri is written in Java, fulfilling the requirement to support Java-based emulator. It emulates the x86 platform, currently the most widely-used computer platform.

VICE was chosen to fulfill the remaining requirement to support a different type of emulator. VICE emulates the Commodore's 8-bit family of computers. From a usage perspective, the Commodore is widely supported in the emulation community, as well as being a platform for which very accurate emulators have been written. VICE is open source, so changes can be made to integrate it with the EF if necessary; furthermore it is written in C/C++, which makes it straightforward to (cross) compile, making it available for many platforms which may ease the porting to KEEP VM.

Support for several other emulators has been added: Qemu (x86), UAE (Amiga), BeebEm (BBC Micro), JavaCPC (Amstrad CPC).

10.4 Emulator configuration

Currently, to configure the emulators included in the EF, a separate emulator configuration is included in the (certified) emulator package for each emulator. This highly intensive task is not durable: it requires constant updates to the configuration when the emulator is modified. This method only works for configuration options already available in the emulators; although for open source emulators the source code can be modified, this will not work for closed source emulators.

To solve the problem of individual emulator configuration, a future solution would be to suggest that emulators included in the EF support a configuration API, i.e. each emulator implements a specified API that would allow the framework to call the methods in the API to control the emulator. This way, emulator developers have more control over the configuration, and the EF can rely on a generic API to control every emulator.

As configuration complexity differs between emulators, it is suggested that the configuration API is broken down into three levels of complexity: at the lowest, simplest level, only the very basic configuration parameters, such as mounting of disk and BIOS images, starting and stopping the emulator, are supported. Higher levels include all parameters of lower levels, and adding configuration of peripherals such as mouse/keyboard. At the highest complexity level, all the lower level parameters are supported, as well as video/chip timings, screen

[29] http://dioscuri.sourceforge.net/

[30] http://www.viceteam.org/

settings, etc. This hierarchy is displayed in Figure 8, which is a suggested configuration design using XML files and Java interfaces.

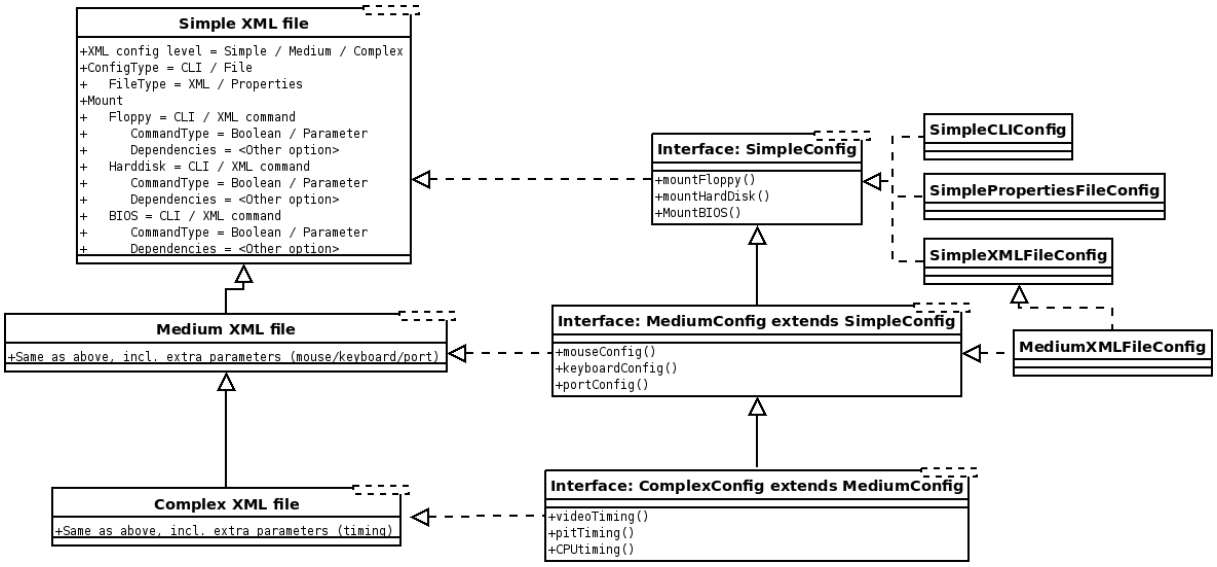


Figure 8: Emulator configuration design

This hierarchy of configuration would allow emulator developers to choose which level they can or want to implement, and so would allow the EF even for the simplest configuration to control the emulators.

As the EF and every emulator is expected to run on the KEEP VM in the future, the VM will act as an intermediary between the emulators and the EF, as the user will control the EF. The EF passes the required calls to the VM, which in turn will pass them on to the running emulator. In this respect, the API described above will be implemented between the VM and the EF as well as between the VM and the emulators.

10.5 Process control

The requirements state that the Emulation Framework should have control over any emulator it starts. However, early in the project it was discovered that controlling these emulators turned out to be a far more difficult task than first assumed. Several levels of complexity can be realized to control the emulators, ranging from difficult to implement but retaining a high level of control, to a simple implementation but relinquishing a lot of control.

- The EF has full control over the process
  The EF, similar to an OS, can spawn an emulator as a child process, and retain full control over it. Similar to an OS, the EF has full access to the input and output of the process, and thus can redirect the video, audio, etc. This scenario retains most control, and can therefore do nearly everything with the inputs/outputs of the emulator. This will also allow the EF to provide a generic control, regardless of emulator.
  However, this is the most complex scenario, and as stated previously, the EF will have to implement many process control features that are usually only found in full-fledged operating systems.
- EF controls process via host/KEEP VM
  A slightly less complex scenario, that trades of some control for simplicity, is that the EF asks the underlying host (this can be a common OS such as Windows/Linux, but also the KEEP VM) to spawn the emulator as a separate process. The EF then relies

on the host to control the process for it, e.g. the EF will request the host to perform certain features such as stopping, maximizing, minimizing, etc., the process. The EF is reliant on the host to perform these actions, and unless special privileges are given to the EF, the list of actions is likely to be fairly limited. Hence a certain amount of control is lost; it is unlikely the EF can ask the host access to the input/output streams of the separate emulator process

- EF communicates with process via a standard protocol (e.g. network sockets)
  The least complex (from an EF point of view) of the three scenarios, but also likely to retain the least control over the emulator. The EF spawns the emulator as a separate process on the host, and maintains a direct link to the emulator via a standard protocol such as network (Berkely/Java sockets), or TCP/IP. This does mean that only emulators that support this protocol can be controlled, and even then the expected amount of control is assumed to be limited, e.g. simple input and output data can be expected (e.g. keystrokes in and screenshots out), but more complex data such as sound or video out is likely to be too data intensive for any such protocol.
  This scenario is technically speaking the least complex but also allows little control, and is quite demanding on the modifications required by an emulator for it to be able to be controlled.

Glossary

Architectural Design Document

The high-level design document for the entire system (this document)

Atomic Digital Object

A digital object consisting of a single file

Compound Digital Object

A digital object consisting of a container format (such as ISO or ZIP), containing a collection of compound or atomic digital objects

Core

The part of the framework not including the GUI, nor the emulators that run in it, nor the transfer tools or the virtual layer on which it will run. These are all separate systems within the Emulation Framework. This document outlines the design of the Core.

Digital Object

An object composed of a set of bit sequences, e.g. a single document such as a PDF file, or an image of a (console) game, etc.

Emulation Framework

A framework that offers emulation services for digital preservation. Its main functionality is to allow a user to load a digital object and select one of a number of pathways in the framework to render the digital object in its original environment.

Metadata

Data about other data; more specifically, data describing properties of the digital object

Pathway

A structured description of the complete hardware and software stack needed to render a digital object. This usually consists of four layers (digital object, rendering application, OS, hardware platform), but may have more or less, e.g. some console game only have two: digital object and hardware platform

The terms pathway and viewpath can be considered to be used interchangeably

Software package

The software layers of the pathway, i.e. the rendering application and operating system

Preservation Fidelity factor

A user-generated indicator of how well a pathway supports the selected digital object

Appendix A: Emulator Archive WSDL

```
<?xml version="1.0" encoding="utf-8"?>

<wsdl:definitions name="EmulatorArchive"
                  targetNamespace="http://emulatorarchive.keep.eu"
                  xmlns:tns="http://emulatorarchive.keep.eu"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

    <!-- type/schema definition -->
    <wsdl:types>

        <xs:schema xmlns:tns="http://emulatorarchive.keep.eu"
                   targetNamespace="http://emulatorarchive.keep.eu"
                   xmlns:ea="http://emulatorarchive.keep.eu/EmulatorPackage"
                  xmlns:xmime="http://www.w3.org/2005/05/xmlmime">

            <!-- http://www.ibm.com/developerworks/xml/library/ws-tip-
imports.html -->
            <xs:import
namespace="http://emulatorarchive.keep.eu/EmulatorPackage"
                       schemaLocation="EmulatorPackageSchema.xsd"/>

                        <xs:element name="emuPackage"
type="ea:emulatorPackage"/>
            <xs:element name="emulatorPackageList">
                <xs:complexType>
                        <xs:sequence>
                            <xs:element name="emulatorPackage"
type="ea:emulatorPackage" minOccurs="0" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
            </xs:element>

            <xs:element name="hardwareID" type="xs:string"/>
            <xs:element name="hardwareIDs">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="id" type="xs:string"
minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>

                        <xs:element name="emulatorID_1" type="xs:int"/>
                        <xs:element name="emulatorID_2" type="xs:int"/>

            <xs:element name="packageFile" type="xs:base64Binary"

xmime:expectedContentTypes="application/octet-stream"/>

            <!--
            Dummy element used for notification operations
(http://www.w3.org/TR/wsdl#_notification)
```

```
            Apparently not supported by CXF (?)
            -->
            <xs:element name="dummyElement" type="xs:int"/>
            <!--
            The same element/message cannot be used for several operations
            Workaround: define multiple elements for each usage.
            See forum discussion here: http://old.nabble.com/%22Non-unique-
body-parts%22-error-when-trying-to-use-same-input-message-for-2-different-
operations-ts28387764.html
            -->
            <xs:element name="dummyElement_2" type="xs:int"/>

        </xs:schema>

    </wsdl:types>


    <!-- message definition -->
    <wsdl:message name="GetEmulatorPackageInput">
        <wsdl:part name="parameters" element="tns:emulatorID_1"/>
    </wsdl:message>
    <wsdl:message name="GetEmulatorPackageOutput">
        <wsdl:part name="emulatorPackage" element="tns:emuPackage"/>
    </wsdl:message>
    <wsdl:message name="GetEmulatorPackageListInput">
        <wsdl:part name="parameters" element="tns:dummyElement"/>
    </wsdl:message>
    <wsdl:message name="GetEmulatorPackageListOutput">
        <wsdl:part name="emulatorPackageList"
element="tns:emulatorPackageList"/>
    </wsdl:message>
    <wsdl:message name="DownloadEmulatorInput">
        <wsdl:part name="parameters" element="tns:emulatorID_2"/>
    </wsdl:message>
    <wsdl:message name="DownloadEmulatorOutput">
        <wsdl:part name="parameters" element="tns:packageFile"/>
    </wsdl:message>
    <wsdl:message name="GetSupportedHardwareInput">
                <wsdl:part name="parameters" element="tns:dummyElement_2"/>
       </wsdl:message>
    <wsdl:message name="GetSupportedHardwareOutput">
        <wsdl:part name="parameters" element="tns:hardwareIDs"/>
    </wsdl:message>
    <wsdl:message name="GetEmusByHardwareInput">
                <wsdl:part name="parameters" element="tns:hardwareID"/>
       </wsdl:message>
    <wsdl:message name="GetEmusByHardwareOutput">
        <wsdl:part name="parameters" element="tns:emulatorPackageList"/>
    </wsdl:message>


    <!-- portType definition -->
    <wsdl:portType name="EmulatorArchivePortType">
        <wsdl:operation name="DownloadEmulator">
           <wsdl:input message="tns:DownloadEmulatorInput"/>
           <wsdl:output message="tns:DownloadEmulatorOutput"/>
        </wsdl:operation>
        <wsdl:operation name="GetEmulatorPackage">
           <wsdl:input message="tns:GetEmulatorPackageInput"/>
           <wsdl:output message="tns:GetEmulatorPackageOutput"/>
        </wsdl:operation>
```

```
        <wsdl:operation name="GetEmulatorPackageList">
           <wsdl:input message="tns:GetEmulatorPackageListInput"/>
           <wsdl:output message="tns:GetEmulatorPackageListOutput"/>
        </wsdl:operation>
        <wsdl:operation name="GetSupportedHardware">
           <wsdl:input message="tns:GetSupportedHardwareInput"/>
           <wsdl:output message="tns:GetSupportedHardwareOutput"/>
        </wsdl:operation>
        <wsdl:operation name="GetEmusByHardware">
           <wsdl:input message="tns:GetEmusByHardwareInput"/>
           <wsdl:output message="tns:GetEmusByHardwareOutput"/>
        </wsdl:operation>
    </wsdl:portType>

    <!-- binding definition -->
    <wsdl:binding name="EmulatorArchiveBinding"
type="tns:EmulatorArchivePortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="DownloadEmulator">
            <soap:operation
soapAction="http://emulatorarchive.keep.eu/DownloadEmulator"
style="document"/>
            <wsdl:input>
               <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output>
               <soap:body use="literal"/>
           </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="GetEmulatorPackage">
            <soap:operation
soapAction="http://emulatorarchive.keep.eu/GetEmulatorPackage"
style="document"/>
            <wsdl:input>
               <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output>
               <soap:body use="literal"/>
           </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="GetEmulatorPackageList">
            <soap:operation
soapAction="http://emulatorarchive.keep.eu/GetEmulatorPackageList"
style="document"/>
            <wsdl:input>
               <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output>
               <soap:body use="literal"/>
           </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="GetSupportedHardware">
            <soap:operation
soapAction="http://emulatorarchive.keep.eu/GetSupportedHardware"
style="document"/>
            <wsdl:input>
               <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output>
               <soap:body use="literal"/>
```

```
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="GetEmusByHardware">
            <soap:operation
soapAction="http://emulatorarchive.keep.eu/GetEmusByHardware"
style="document"/>
            <wsdl:input>
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
               <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>

    </wsdl:binding>

    <!-- service definition -->
    <wsdl:service name="EmulatorArchiveService">
        <wsdl:port name="EmulatorArchivePort"
binding="tns:EmulatorArchiveBinding">
            <soap:address
location="http://localhost:9001/emulatorArchive/"/>
        </wsdl:port>
    </wsdl:service>


</wsdl:definitions>
```

Appendix B: Software Archive WSDL

```
<?xml version="1.0" encoding="utf-8"?>

<wsdl:definitions name="SoftwareArchive"
                  targetNamespace="http://softwarearchive.keep.eu"
                  xmlns:tns="http://softwarearchive.keep.eu"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:pw="http://softwarearchive.keep.eu/Pathway">

    <!-- type/schema definition -->
    <wsdl:types>

        <xs:schema xmlns:tns="http://softwarearchive.keep.eu"
                   targetNamespace="http://softwarearchive.keep.eu"

xmlns:swa="http://softwarearchive.keep.eu/SoftwarePackage"
                   xmlns:pw="http://softwarearchive.keep.eu/Pathway"

xmlns:xmime="http://www.w3.org/2005/05/xmlmime">

            <!-- http://www.ibm.com/developerworks/xml/library/ws-tip-
imports.html -->
            <xs:import
namespace="http://softwarearchive.keep.eu/SoftwarePackage"
                       schemaLocation="SoftwarePackageSchema.xsd"/>
            <xs:import namespace="http://softwarearchive.keep.eu/Pathway"
                       schemaLocation="PathwaySchema.xsd"/>

                        <xs:element name="softwarePackage"
type="swa:softwarePackage"/>
            <xs:element name="softwarePackageList">
                <xs:complexType>
                        <xs:sequence>
                            <xs:element name="softwarePackage"
type="swa:softwarePackage" minOccurs="0" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
            </xs:element>

            <xs:element name="pathwayList">
                <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="pw:pathway" minOccurs="0"
maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
            </xs:element>

            <!--
            The same element/message cannot be used for several operations
            Workaround: define multiple elements for each usage.
```

```
            See forum discussion here: http://old.nabble.com/%22Non-unique-
body-parts%22-error-when-trying-to-use-same-input-message-for-2-different-
operations-ts28387764.html
            -->
            <xs:element name="softwareID_1"     type="xs:string"/>
            <xs:element name="softwareID_2"     type="xs:string"/>

            <xs:element name="softwareFile"     type="xs:base64Binary"

xmime:expectedContentTypes="application/octet-stream"/>

            <xs:element name="fileFormat"        type="xs:string"/>

            <!--
            Dummy element used for notification operations
(http://www.w3.org/TR/wsdl#_notification)
            Apparently not supported by CXF (?)
            -->
            <xs:element name="dummyElement_1" type="xs:string"/>
        </xs:schema>
    </wsdl:types>


    <!-- message definition -->
        <wsdl:message name="GetSoftwarePackageInfoInput">
        <wsdl:part name="parameters" element="tns:softwareID_1"/>
    </wsdl:message>
    <wsdl:message name="GetSoftwarePackageInfoOutput">
        <wsdl:part name="emulatorPackage" element="tns:softwarePackage"/>
    </wsdl:message>
    <wsdl:message name="GetAllSoftwarePackagesInfoInput">
        <wsdl:part name="parameters" element="tns:dummyElement_1"/>
    </wsdl:message>
    <wsdl:message name="GetAllSoftwarePackagesInfoOutput">
        <wsdl:part name="softwarePackageList"
element="tns:softwarePackageList"/>
    </wsdl:message>
    <wsdl:message name="GetPathwaysByFileFormatInput">
        <wsdl:part name="parameters1" element="tns:fileFormat"/>
    </wsdl:message>
    <wsdl:message name="GetPathwaysByFileFormatOutput">
        <wsdl:part name="parameters" element="tns:pathwayList"/>
    </wsdl:message>
    <wsdl:message name="GetSoftwarePackagesByPathwayInput">
        <wsdl:part name="parameters1" element="pw:pathway"/>
    </wsdl:message>
    <wsdl:message name="GetSoftwarePackagesByPathwayOutput">
        <wsdl:part name="parameters" element="tns:softwarePackageList"/>
    </wsdl:message>
    <wsdl:message name="DownloadSoftwareInput">
        <wsdl:part name="parameters" element="tns:softwareID_2"/>
    </wsdl:message>
    <wsdl:message name="DownloadSoftwareOutput">
        <wsdl:part name="parameters" element="tns:softwareFile"/>
    </wsdl:message>

    <!-- portType definition -->
    <wsdl:portType name="SoftwareArchivePortType">
        <wsdl:operation name="GetSoftwarePackageInfo">
           <wsdl:input message="tns:GetSoftwarePackageInfoInput"/>
           <wsdl:output message="tns:GetSoftwarePackageInfoOutput"/>
```

```
        </wsdl:operation>
        <wsdl:operation name="GetAllSoftwarePackagesInfo">
           <wsdl:input message="tns:GetAllSoftwarePackagesInfoInput"/>
           <wsdl:output message="tns:GetAllSoftwarePackagesInfoOutput"/>
        </wsdl:operation>
        <wsdl:operation name="GetPathwaysByFileFormat">
           <wsdl:input message="tns:GetPathwaysByFileFormatInput"/>
           <wsdl:output message="tns:GetPathwaysByFileFormatOutput"/>
        </wsdl:operation>
        <wsdl:operation name="GetSoftwarePackagesByPathway">
           <wsdl:input message="tns:GetSoftwarePackagesByPathwayInput"/>
           <wsdl:output message="tns:GetSoftwarePackagesByPathwayOutput"/>
        </wsdl:operation>
        <wsdl:operation name="DownloadSoftware">
           <wsdl:input message="tns:DownloadSoftwareInput"/>
           <wsdl:output message="tns:DownloadSoftwareOutput"/>
        </wsdl:operation>
    </wsdl:portType>

    <!-- binding definition -->
    <wsdl:binding name="SoftwareArchiveBinding"
type="tns:SoftwareArchivePortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="GetSoftwarePackageInfo">
            <soap:operation
soapAction="http://softwarearchive.keep.eu/GetSoftwarePackageInfo"
style="document"/>
            <wsdl:input>
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
               <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="GetAllSoftwarePackagesInfo">
            <soap:operation
soapAction="http://softwarearchive.keep.eu/GetAllSoftwarePackagesInfo"
style="document"/>
            <wsdl:input>
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
               <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="GetPathwaysByFileFormat">
            <soap:operation
soapAction="http://softwarearchive.keep.eu/GetPathwaysByFileFormat"
style="document"/>
            <wsdl:input>
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
               <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="GetSoftwarePackagesByPathway">
            <soap:operation
soapAction="http://softwarearchive.keep.eu/GetSoftwarePackagesByPathway"
style="document"/>
```

```
            <wsdl:input>
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
               <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="DownloadSoftware">
            <soap:operation
soapAction="http://softwarearchive.keep.eu/DownloadSoftware"
style="document"/>
            <wsdl:input>
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
               <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>

    <!-- service definition -->
    <wsdl:service name="SoftwareArchiveService">
        <wsdl:port name="SoftwareArchivePort"
binding="tns:SoftwareArchiveBinding">
            <soap:address
location="http://localhost:9000/softwarearchive/"/>
        </wsdl:port>
    </wsdl:service>

</wsdl:definitions>
```

Appendix C: Core API

Method Summary	
EmulatorPackage	**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
Format	**autoSelectFormat**(List<Format> formats) Select a format from a list of formats.
Pathway	**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
SoftwarePackage	**autoSelectSoftwareImage**(List<SoftwarePackage> swPacks) Select a software image automatically from a list of software images.
List<Format>	**characterise**(File digObj) Characterise a digital object and returns information on format names, mime types and the reporting tools.
void	**cleanUp**() Clean up any temporary files and directories that were created by the Core Engine to unpack files, run emulators, etc.
Pathway	**extractPathwayFromFile**(File metadataFile) Retrieve the technical environment, i.e.
Properties	**getCoreSettings**() Get the Core Engine settings
Map<String,List<Map<String,String>>>	**getEmuConfig**(Integer conf) Get the configuration map of all available emulator parameters Useful for manual configuration of the emulator, to be used with setEmuConfig()
List<EmulatorPackage>	**getEmulatorsByPathway**(Pathway pathway) Returns a list of supported emulators that satisfy a given pathway.
List<EmulatorPackage>	**getEmuListFromArchive**() Get the list of all emulator packages available in the Emulator Archive
List<EmulatorPackage>	**getEmusByHWFromArchive**(String hardwareName) Get the list of emulator packages that support a hardware type in the emulator archive
Map<String,List<String>>	**getFileInfo**(File digObj)

	Characterise a digital object and returns file information
List<Pathway>	**getPathways**(Format format) Get pathways for a given file formatName.
List<DBRegistry>	**getRegistries**() Retrieve the list of technical registries
List<SoftwarePackage>	**getSoftwareByPathway**(Pathway pathway) Returns a list of supported software packages that satisfy a given pathway.
List<SoftwarePackage>	**getSoftwareListFromArchive**() Get all software packages available in the software archive
Set<String>	**getSupportedHardwareFromArchive**() Get the list of hardware supported by the Emulator Archive
Map<String,List<String>>	**getTechMetadata**(File digObj) Characterise a digital object and returns technical metadata information
String	**getTitle**() Get the Emulation Framework title from the jar manifest
String	**getVendor**() Get the Emulation Framework vendor from the jar manifest
String	**getVersion**() Get the Emulation Framework version from the jar manifest
List<EmulatorPackage>	**getWhitelistedEmus**() Select the whitelisted emulator IDs from the local database
boolean	**isPathwaySatisfiable**(Pathway pathway) Checks if a given pathway is satisfiable given the available emulators and software images
Map<EmulatorPackage, List<SoftwarePackage>>	**matchEmulatorWithSoftware**(Pathway pathway) Match emulators with a list of associated software images from a given pathway
Integer	**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()
void	**registerObserver**(CoreObserver coreObs) Register an observer
void	**removeObserver**(CoreObserver coreObs) Remove an observer
void	**runEmulationProcess**(Integer conf) Run the chosen emulation process An emulator must have already been selected and its configuration settings properly prepared.

void	**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()
List<DBRegistry>	**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
boolean	**start**(File file) Launches the emulation process automatically (i.e.
boolean	**start**(File file, File metadata) Launches the emulation process automatically (i.e.
boolean	**start**(File file, List<Pathway> pathways) Launches the emulation process given a digital object and a list of pathways to select from.
boolean	**start**(File file, Pathway pathway) Launches the emulation process given a digital object and a specific pathway.
boolean	**stop**() Stop the Core Emulator Framework engine
boolean	**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)
boolean	**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)

Method Detail

getVersion
String **getVersion**()

Get the Emulation Framework version from the jar manifest

Returns:

String representing the version

---

getTitle
String **getTitle**()

Get the Emulation Framework title from the jar manifest

Returns:

String representing the title

---

getVendor

`String` **`getVendor`**`()`

Get the Emulation Framework vendor from the jar manifest

Returns:

String representing the vendor

---

getCoreSettings

`Properties` **`getCoreSettings`**`()`

Get the Core Engine settings

Returns:

Properties the Java Properties object

---

registerObserver

`void` **`registerObserver`**`(CoreObserver coreObs)`

Register an observer

Parameters:

`coreObs` - A Core Emulation Framework observer

---

removeObserver

`void` **`removeObserver`**`(CoreObserver coreObs)`

Remove an observer

Parameters:

`coreObs` - A Core Emulation Framework observer

---

stop

`boolean` **`stop`**`()`
`throws IOException`

Stop the Core Emulator Framework engine

Returns:

True if engine stopped without error, false otherwise

Throws:

`IOException`

---

cleanUp

`void` **`cleanUp`**`()`

Clean up any temporary files and directories that were created by the Core Engine to unpack files, run emulators, etc.

---

characterise

`List<Format>` **`characterise`**`(File digObj)`
`throws IOException`

Characterise a digital object and returns information on format names, mime types and the reporting tools.

Parameters:

`digObj` - File representing the digital object

Returns:

List list of Format objects

Throws:

`IOException` - If a FITS characterisation error occurs

See Also:

`Format`

---

getTechMetadata

```
Map<String,List<String>> getTechMetadata(File digObj)
                                         throws IOException
```

Characterise a digital object and returns technical metadata information

Parameters:

`digObj` - File representing the digital object

Returns:

Map> a Map of item names (as keys) and an associated list of values

Throws:

`IOException` - If a FITS characterisation error occurs

---

getFileInfo

```
Map<String,List<String>> getFileInfo(File digObj)
                                     throws IOException
```

Characterise a digital object and returns file information

Parameters:

`digObj` - File representing the digital object

Returns:

Map> a Map of item names (as keys) and an associated list of values

Throws:

`IOException` - If a FITS characterisation error occurs

---

getPathways

```
List<Pathway> getPathways(Format format)
                          throws IOException
```

Get pathways for a given file formatName. This contacts the active technical registries and returns the pathways (digital object, rendering application, OS, hardware platform) found for the given formatName.

Parameters:

`format` - Format of digital object

Returns:

List List of available pathways objects

Throws:

`IOException` - If a Software Archive connection error occurs

---

isPathwaySatisfiable

```
boolean isPathwaySatisfiable(Pathway pathway)
                             throws IOException
```

Checks if a given pathway is satisfiable given the available emulators and software images

Parameters:

`pathway` - Pathway Configuration describing the environment

Returns:

true is pathway is satisfiable, false otherwise

Throws:

`IOException` - If an error occurs while connecting the Emulator/Software Archive

---

getEmuConfig

```
Map<String,List<Map<String,String>>> getEmuConfig(Integer conf)
                                                  throws IOException
```

Get the configuration map of all available emulator parameters Useful for manual configuration of the emulator, to be used with `setEmuConfig()`

Parameters:

`conf` - Integer representing an existing configuration

Returns:

Map<map>> Map of emulator parameters ordered by component and (multiple) parameter-value pairs

Throws:

`IOException` - If a template error occurs while configuring the parameters

---

setEmuConfig

```
void setEmuConfig(Map<String,List<Map<String,String>>> options,
                  Integer conf)
                  throws IOException
```

Set the emulator parameters Useful for manual configuration of the emulator to be used with `getEmuConfig()`

Parameters:

`options` - Map<map>> map of emulator parameters ordered by component and (multiple) parameter-value pairs

`conf` - Integer representing an existing configuration

Throws:

`IOException` - If a template error occurs while configuring the parameters

---

prepareConfiguration

```
Integer prepareConfiguration(File digObj,
                             EmulatorPackage emuPack,
                             SoftwarePackage swPack,
                             Pathway pathway)
                             throws IOException
```

Prepares the configuration settings for the selected emulation process The resulting configuration (emulator options) can be edited using `setEmuOptions()` and `getEmuOptions()`

Parameters:

`digObj` - File representing the digital object to the passed to the emulator configurator

`emuPack` - Emulator metadata package

`swPack` - Software metadata package

`pathway` - The Pathway that forms the basis for this configuration

Returns:

Integer An identification of a newly generated configuration

Throws:

`IOException` - If an error occurs while connecting the Emulator/Software Archive

---

matchEmulatorWithSoftware

```
Map<EmulatorPackage,List<SoftwarePackage>>
matchEmulatorWithSoftware(Pathway pathway)
                                                    throws
IOException
```

Match emulators with a list of associated software images from a given pathway

Parameters:

`pathway` - Pathway object to analyse

Returns:

Map> A map of emulators with their associated list of compatible software images

Throws:

`IOException` - If an error occurs while connecting the Emulator/Software Archive

---

autoSelectEmulator

```
EmulatorPackage autoSelectEmulator(List<EmulatorPackage> emuPacks)
                                   throws IOException
```

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

Parameters:

`emuPacks` - list of emulator packages

Returns:

EmulatorPackage The selected emulator package

Throws:

`IOException` - If an error occurs while connecting the Emulator Archive

autoSelectSoftwareImage

```
SoftwarePackage autoSelectSoftwareImage(List<SoftwarePackage> swPacks)
                                        throws IOException
```

Select a software image automatically from a list of software images. The selection process picks the first encountered software image from the list

Parameters:

`swPacks` - list of software packages

Returns:

SoftwarePackage The selected software package

Throws:

`IOException` - If an error occurs while connecting the Software Archive

---

autoSelectPathway

```
Pathway autoSelectPathway(List<Pathway> pathways)
                          throws IOException
```

Select a valid pathway automatically from a list of potential pathways The selection process simply picks the first encountered satisfiable pathway

Parameters:

`pathways` - List List of pathway objects

Returns:

selected Pathway object

Throws:

`IOException` - If an error occurs while analysing the given Pathways

---

autoSelectFormat

```
Format autoSelectFormat(List<Format> formats)
                        throws IOException
```

Select a format from a list of formats. This will simply pick the first format of the list which should correspond to the format identified by the highest number of tools within FITS

Parameters:

`formats` - list of file Format objects

Returns:

selected Format object

Throws:

`IOException` - If a Format exception occurs

See Also:

`Format`

---

runEmulationProcess

```
void runEmulationProcess(Integer conf)
                         throws IOException
```

Run the chosen emulation process An emulator must have already been selected and its configuration settings properly prepared.

Parameters:

`conf` - Integer representing an existing configuration

Throws:

`IOException` - If the configuration environment is not set up properly

---

extractPathwayFromFile

```
Pathway extractPathwayFromFile(File metadataFile)
                               throws IOException
```

Retrieve the technical environment, i.e. pathway from a metadata file (xml file) which must validate the xsd schema `PathwaySchema.xsd`.

Parameters:

`metadataFile` - File describing the pathway required to render a the digital object

Returns:

Pathway The pathway described by the XML file

Throws:

`IOException` - If a validation error occurs

---

start

```
boolean start(File file)
              throws IOException
```

Launches the emulation process automatically (i.e. no human intervention) given a digital object only. This method will characterise the given digital object, retrieve a list of pathways from the identified format and then call `start(File file, List<Pathway> pathways)`.

Parameters:

`file` - File representing the digital object to be rendered via emulation

Returns:

True if emulation process is launched without error, false otherwise

Throws:

`IOException` - If a characterisation error occurs

---

start

```
boolean start(File file,
              File metadata)
              throws IOException
```

Launches the emulation process automatically (i.e. no human intervention) given a digital object and its metadata which should contain all the necessary information to prepare/configure the emulation environment. This method will extract a the Pathway from the metadata using and then launch `start(File file, Pathway pw)`. If no

emulation pathway is defined in the metadata file, then the automatic emulation process is launched by calling `start(File file)`

Parameters:

`file` - File representing the digital object to be rendered via emulation

`metadata` - File representing the pathway required to render the digital object

Returns:

True if emulation process is launched without error, false otherwise

Throws:

`IOException` - If a validation error occurs

---

start

```
boolean start(File file,
              List<Pathway> pathways)
              throws IOException
```

Launches the emulation process given a digital object and a list of pathways to select from. A suitable pathway is then selected using an automatic selection method before calling `start(File file, Pathway pathway).`

Parameters:

`file` - File representing the digital object to be rendered via emulation

`pathways` - List of pathways

Returns:

True if emulation process is launched without error, false otherwise

Throws:

`IOException` - If a pathway error occurs

---

start

```
boolean start(File file,
              Pathway pathway)
              throws IOException
```

Launches the emulation process given a digital object and a specific pathway. Returns false if the given pathway is not satisfiable (emulator/software not supported/available).

Parameters:

`file` - File representing the digital object to be rendered via emulation

`pathway` - Pathway describing the environment required to render the digital object

Returns:

True if emulation process is launched without error, false otherwise

Throws:

`IOException` - If a pathway error occurs, or the emulation process cannot be launched successfully

---

getEmuListFromArchive

```
List<EmulatorPackage> getEmuListFromArchive()
                                              throws IOException
```

Get the list of all emulator packages available in the Emulator Archive

Returns:

List the list of all emulator packages

Throws:

`IOException` - If an Emulator Archive connection error occurs

---

getSupportedHardwareFromArchive

```
Set<String> getSupportedHardwareFromArchive()
                                              throws IOException
```

Get the list of hardware supported by the Emulator Archive

Returns:

Set Hardware names supported by the emulator archive

Throws:

`IOException` - If an Emulator Archive connection error occurs

---

getEmusByHWFromArchive

```
List<EmulatorPackage> getEmusByHWFromArchive(String hardwareName)
                                               throws IOException
```

Get the list of emulator packages that support a hardware type in the emulator archive

Parameters:

`hardwareName` - String describing the hardware type

Returns:

List Emulators that support the hardware type

Throws:

`IOException` - If an Emulator Archive connection error occurs

---

getEmulatorsByPathway

```
List<EmulatorPackage> getEmulatorsByPathway(Pathway pathway)
                                              throws IOException
```

Returns a list of supported emulators that satisfy a given pathway. Retrieves a list of emulators packages that satisfy (the hardware part) of the pathway

Parameters:

`pathway` - Pathway object describing the environment required to render a digital object

Returns:

list of emulator metadata packages

Throws:

`IOException` - If an Emulator Archive connection error occurs

---

getWhitelistedEmus

```
List<EmulatorPackage> getWhitelistedEmus()
                                          throws IOException
```

Select the whitelisted emulator IDs from the local database

Returns:

List a list of emulator metadata packages

Throws:

`IOException` - If an database connection error occurs

---

whiteListEmulator

```
boolean whiteListEmulator(Integer i)
                          throws IOException
```

Adds an emulator ID to the whitelist in the local database (list of emulators that will be used for rendering a digital object)

Parameters:

`i` - Unique ID of emulator

Returns:

True if ID successfully added to whitelist, false otherwise

Throws:

`IOException` - If an database connection error occurs

---

unListEmulator

```
boolean unListEmulator(Integer i)
                       throws IOException
```

Removes an emulator ID from the whitelist in the local database (list of emulators that will be used for rendering a digital object)

Parameters:

`i` - Unique ID of emulator

Returns:

True if ID successfully removed from whitelist, false otherwise

Throws:

`IOException` - If a database connection error occurs

---

getSoftwareByPathway

```
List<SoftwarePackage> getSoftwareByPathway(Pathway pathway)
                                           throws IOException
```

Returns a list of supported software packages that satisfy a given pathway. Retrieves a list of software metadat packages from the software database that satisfy (the operating system and application part) of the pathway

Parameters:

`pathway` - Pathway object

Returns:

list of software packages

Throws:

`IOException` - If an Software Archive connection error occurs

---

getSoftwareListFromArchive

```
List<SoftwarePackage> getSoftwareListFromArchive()
                                                  throws IOException
```

Get all software packages available in the software archive

Returns:

List the list of software packages in the archive

Throws:

`IOException` - If an Software Archive connection error occurs

---

getRegistries

```
List<DBRegistry> getRegistries()
                                throws IOException
```

Retrieve the list of technical registries

Returns:

List The list of technical registries in the local database

Throws:

`IOException` - If a database connection error occurs

---

setRegistries

```
List<DBRegistry> setRegistries(List<DBRegistry> listReg)
                                throws IOException
```

Insert registry information from list into the local database This replaces all existing registry information with the contents of the list

Parameters:

`listReg` - List of Registry

Throws:

`IOException` - If a database connection error occurs