Keeping Emulation Environments Portable

FP7-ICT-231954

Guideline document for emulator adoption in the Emulation Framework

Deliverable number	I5.5
Nature	Report
Dissemination level	PU (public)
Delivery date	**Due**: M35 (December 2011) **Actual**: M37 (February 2012)
Status	Final
Work package number	WP5
Lead beneficiary	TSSP
Author(s)	Bram Lohman (TSSP)

Document history

Revisions

Version	Date	Author	Changes
0.1	21-04-2011	Bram Lohman (TSSP)	Initial version
1.0	27-02-2012	Bram Lohman (TSSP)	Updated content following review

Reviews

Date	Name	Result
24-02-2012	Jeffrey van der Hoeven	Approved with minor changes

Signature/Approval

Approved by (signature)	Date

Accepted by at European Commission (signature)	Date

Executive Summary

This document contains guidelines for emulator developers on how to develop their emulators for easy integration within the Emulation Framework.

The Emulation Framework (EF) uses an Emulator Archive containing executables of emulators that are used for rendering the hardware requirements of a selected computer environment. These emulators are developed independently of the EF by third parties, mainly from the open source community. For the EF to be able to run an emulator, it has to be added to the Emulator Archive, a configuration file must be added so it can be set up correctly, and the EF must be informed of the addition.

This document addresses this process, and outlines how emulator developers can help to streamline this and what to do to integrate their emulator easily into the EF.

A description of the Emulator Archive database schema is given, describing all tables and contents of the Emulator Archive, showing what information needs to be filled in when adding an emulator.

To include an emulator into the EF Emulator Archive, the following should be taken into account:

1. The packaged emulator must be self-contained, meaning that the package must contain all the files required by the emulator and does not require further installation (i.e. all library dependencies are included and no external configuration, e.g. in the Windows registry).
2. A Freemarker data model used to configure the emulator within the Emulation Framework is described, listing all parameters and their description. This then leads into the generation of a template file for converting the data model into a configuration file, including an example showing the data model and values, template file, and output.

An appendix containing a complete template file for reference purposes is also included.

List of Related Documents

User Requirements Document [URD]	http://www.keep-project.eu/ezpub2/index.php?/eng/content/download/7918/39623/file/KEEP_WP2_D2.2_complete.pdf
Architectural Design Document [ADD]	http://emuframework.sourceforge.net/docs/Architectural-Design-Document_1.1.pdf
System User Guide [SUG]	http://emuframework.sourceforge.net/docs/System-User-Guide_1.1.pdf
System Maintenance Guide [SMG]	http://emuframework.sourceforge.net/docs/System-Maintenance-Guide_1.1.pdf
EF Howto Remote Emulation [HRE]	http://emuframework.sourceforge.net/docs/EF-howto-remoteemulation-1.0.pdf

Abbreviations

EA	Emulator Archive
EF	Emulation Framework
SWA	Software Archive

Table of Contents

1. Introduction

This document contains guidelines for emulator developers on how to develop their emulators for easy integration within the Emulation Framework.

The Emulation Framework uses an Emulator Archive containing executables of emulators that are used for rendering the hardware requirements of a selected environment. These emulators are developed independently of the Emulation Framework by third parties, mainly from the open source community. For the Emulation Framework to be able to run an emulator, it has to be added to the Emulator Archive, a configuration file must be added so it can be set up correctly, and the Emulation Framework must be informed of the addition.

1.1. Objectives and scope

The objective of this document is to inform emulator developers which steps they need to take to ensure their emulators can be successfully added to the Emulator Archive while keeping this integration process as simple as possible. Moreover, it outlines what functionality the developers can add to ensure their emulators work seamlessly within the Emulation Framework, and can (in the future), be controlled by the Emulation Framework.

This document is written for version 2.0.0 of the Emulation Framework. It is based on the database schemas and settings of the corresponding Emulator Archive. Although all effort is made to ensure backward and forward compatibility, future versions of the Emulation Framework and Emulator Archive may change and this document may need to be updated accordingly.

1.2. About the Emulation Framework

The Emulation Framework (EF) allows rendering of digital files and computer programs in their native environment. It offers the potential to view these files using their original ‘look and feel’, independent from the current view design of computer systems. The spectrum of computer platforms and applications that can be supported is practically unlimited.

Release 2.0.0 of the EF[1] supports emulation of the x86, Commodore 64, Amiga, Amstrad CPC and Thomson T07 computer platforms. Emulation is done by using existing (open source) emulators that are carefully selected on their capability to mimic the functionality of these platforms.

The EF 2.0.0 consists of three parts:

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

The **Core EF** is the technical heart of the system, performing the workflow steps (automatic identification of file formats, selecting the required software and automatically configuring the emulation environment) required for rendering the original environment. The Core EF interacts with the Software Archive and the Emulator Archive for selecting the appropriate emulators and software.

[1] Emulation Framework software release 2.0.0, available at: http://emuframework.sf.net

The **Software Archive** is a separate web service that contains the software (applications and operating systems) available for the EF. The download package comes with three open source operating systems, although an included wizard allows any software to be added.

The included operating systems are:

- FreeDOS – an open source MS DOS look-a-like operating system
- Damn Small Linux – a small Linux kernel with limited functionality
- Puppy Linux – a Linux based operating system with small footprint

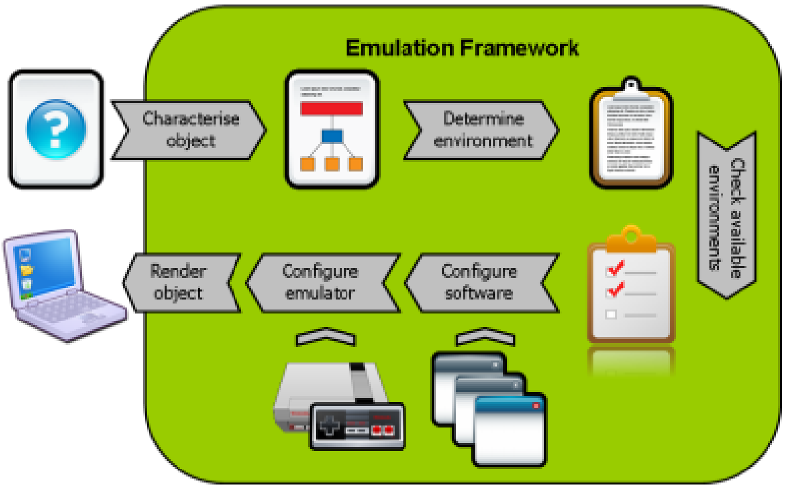
The **Emulator Archive** is a separate web service that contains the emulators available for the EF. The Emulator Archive also contains a wizard for more emulators to be added, and this document outlines the preparation required for an emulator to be added to the Emulator Archive.

The download package comes with the following open source emulators:

1. Dioscuri – x86 Java-based emulator capable of running MS DOS and Linux.
2. QEMU – x86 capable of running MS Windows and Linux.
3. VICE – Commodore 64 emulator
4. UAE – Amiga emulator
5. Java CPC – Amstrad emulator
6. BeebEm – BBC Micro emulator
7. Thomson – Thomson T07 emulator

The Core EF, Software Archive and Emulator Archive are developed by Tessella[2] with support from the National Library of the Netherlands (Koninklijke Bibliotheek, KB)[3].

The EF provides an automated workflow for running emulators, as well as the application and operating system software required for rendering files. The following illustration shows the steps taken when providing an unknown digital file to be rendered in an emulated computer environment.



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

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

The workflow consists of the following steps:

1. Characterise object – based on a given file (selected by a user) the EF identifies which file format it is using the tool FITS. Note that this step is not required if a user only wants to render an environment without a given file;
2. Determine environment – looks up which software and hardware is required to run the file or create the environment;
3. Check available environment – matches the required environment with the best environment available in the EF;
4. Configure software – retrieves selected software from the software archive and (optionally) wraps the given file into a disk image;
5. Configure emulator – retrieves selected emulator from the emulator archive and configures it using emulator specific templates. Attaches software and disk image containing the required software and (optionally) selected file;
6. Render object – launch the prepared emulation environment.

1.3. About the KEEP project

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

More information can be found on the KEEP website[4].

1.4. Outline of this document

Chapter 2 describes the requirements for adding an emulator to the Emulator Archive, including a description of the Emulator Archive database schema, emulator package restrictions and Freemarker templates.

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

2. Requirements for adding an emulator to the Emulation Framework

The Emulation Framework is designed to be scalable. Although the installation kit comes with a basic set of emulators that can be used out of the box, it is also possible to add new emulators and enrich the set of supported computer platforms by the EF. If an emulator developer would like to add his/her emulator to the EF, the following is required:

1. technical information about the emulator and computer platforms, software and file formats it can run;
2. an emulator package containing:
    a. a preinstalled emulator without any dependencies (e.g. libraries) with the executing system;
    b. a template file defining how the emulator must be invoked and configured.

If all this information is available, the emulator can be easily added using the Emulator Archive wizard available in the administrator GUI of the EF. For more information on how to use the wizard, please see [SUG].

The following sections explain the contents of the Emulator Archive, including the required fields; the contents and restrictions of an emulator package; an explanation of the emulator configuration templates; and how to add the emulator to the Emulator Archive.

2.1. Emulator Archive database schema

The Emulation Framework uses the Emulator Archive, a database that can be contacted via a web service, to retrieve emulators. The Emulator Archive stores the emulator binary, along with associated metadata, into various database tables, as outlined in the following sections.

Database table: emulators

Name	Description	Example
emulator_id	Unique numerical ID, used for internal database purposes	1
name	(Simple) name of the emulator	Dioscuri
Version	Version of the emulator	0.7
exec_type	Type of executable	jar (platform independent), exe (DOS/Windows), ELF (Linux binary), etc.
exec_name	Name of the executable	Dioscuri-0.7.0.jar
exec_dir	Relative path of the executable in the installation package	e.g. '.' or './binary'
description	(Extensive) description of the emulator	Dioscuri, the modular emulator
language_id	Language of the emulator interface. Has to be one of the languages defined in the language table	En
package_name	Name of the zipped package containing all emulator files	Dioscuri_070Package.zip

package_type	Compression of package	Zip
package_version	Version of package. This is different from the emulator version as emulators with the same version can be in different packages	1
package	BLOB of package	
user_instructions	Instructions for using the emulator	Set at least the following parameters from the "Configure->Edit Config" menu: BIOS, etc.

Emulators are associated with hardware platforms in the 'emus_hardware' table, which links the emulator ID to the hardware ID. A one-to-many relationship is possible.

Database table: hardware

Name	Description	Example
hardware_id	Unique numerical ID, used for internal database purposes	1
name	Name of the hardware platform	X86

The emulators are required to be associated with an image format, which is the format of the files it can read. This is usually defined as the format of the disk it reads, e.g. for x86 platforms disks are usually in FAT16, FAT32, NTFS or something similar. For console emulators, this may be the image format of the cartridge, e.g. ROM or NDS for Nintendo games.

Emulators are associated with image formats in the 'emus_imageformats' table, which links the emulator ID to the image format ID. A one-to-many relationship is possible.

For emulators requiring a software image to run, e.g. an operating system, this table links the software images it can run to the software image type in the Software Archive. If no software image is required other than the digital object, then it is sufficient to provide the hardware platform that is required by the digital object (e.g. d64 objects require C64 hardware to run, but no further software is required to run the object).

Database table: imageformats

Name	Description	Example
imageformat_id	Unique numerical ID, used for internal database purposes	1
name	Name of the image format	FAT16

2.2. Emulator package

The emulator package described in the previous section is a container file containing all the files required by the emulator to run. This can be a simple 'tar' of all the files, or compression formats such as ZIP/RAR[5]. The files will be extracted and placed in the directory structure defined by the container file on the fly by the EF.

Note that the EF requires the emulator (and the associated files in the container) to complete, i.e. the package must contain all the files required by the emulator and does not require further installation. This means that all library dependencies need to be included, and no external configuration, such as Windows registry settings can be used. This limitation is a result of the workflow process being as simple as possible for the end user.

The emulator package must also contain one or more Freemarker template configuration files for the Emulation Framework to configure the emulator to run. These templates are described in the following section.

2.3. Emulator configuration using templates

Templates, a processing element that can be combined with a data model and processed by a template engine to produce a result document, offer a flexible and generic way of creating custom configurations for emulators, without having to write any language-specific code.

This allows emulator developers to write emulator-specific templates in pseudo-code (rather than forcing them to use Java), and offers greater flexibility by reading these templates at run-time when unpacking and configuring an emulator.

Figure 1 illustrates the processing flow of a template engine[6]. 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.

[5] Note that the only the ZIP format is currently supported, but it is trivial to add other formats

[6] The Freemarker template engine was chosen for the EF: http://freemarker.sourceforge.net/

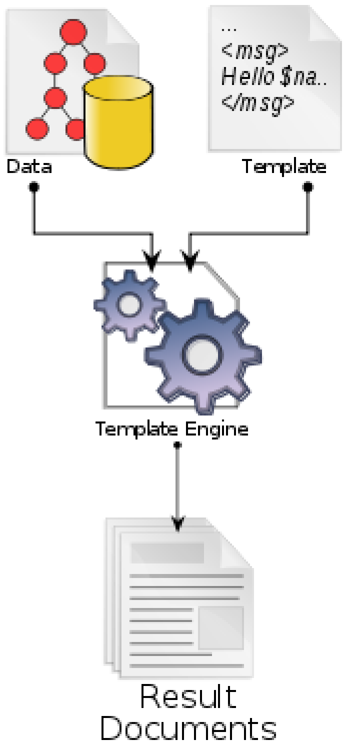


Figure 1: A diagram illustrating all of the basic elements and processing flow of a template engine

In the EF, each emulator package contains 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.

Using templates different classes of template builders can be created that are able to generate a configuration for an emulator. With these builders configurations can be created that contain settings such as 'autorun', 'floppy disks', ‘memory size', 'cpu bits', etc.

Figure 2 shows a visualisation of the data model used in the EF.

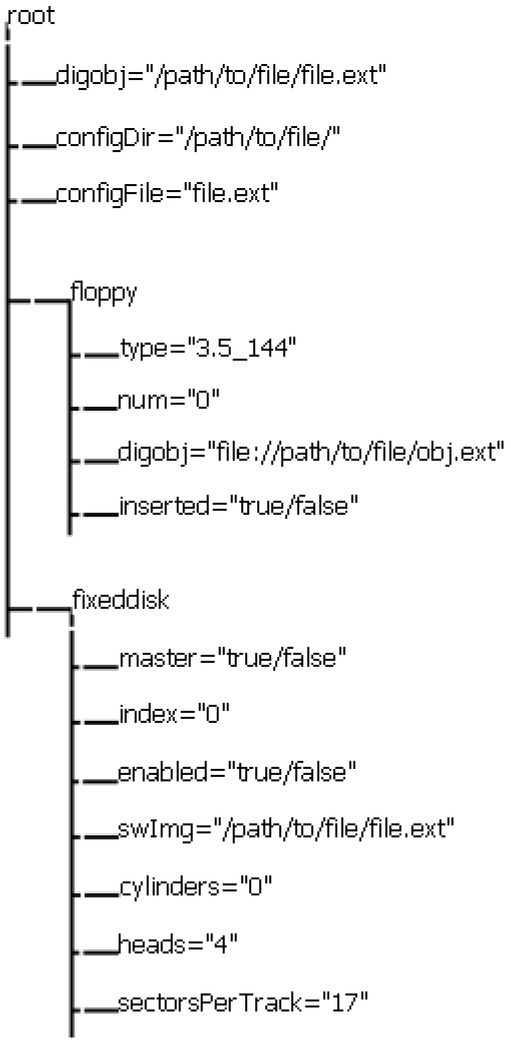


Figure 2: Visualisation of Emulation Framework template data model

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) that 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.

The model is extensible, and the diagram above shows the initial, simple, version. The following tables describe the items that can be used as configuration parameters for an emulator:

Basic template data model

Parameter	Description	Example
digobj	The digital object to be attached to the environment	file.txt
configDir	The configuration directory in which the generated property or XML file is to be placed	config/
configFile	The name of the configuration file that is	config.xml

	generated by the template	
floppy.type	The type of floppy disk that is attached to the drive. This can only be one of the types defined in the EF, listed on the right	XXX floppy types
floppy.num	The floppy drive number. The EF will automatically increment this number (starting at 0) based on the number of files provided to the configuration. The template file must substitute this number with the convention expected by the emulator	0
floppy.digobj	The name of the file that will be attached to the floppy drive	disk1.img
floppy.inserted	A boolean (true/false) indicating whether the floppy disk is inserted (readable) or waiting to be inserted	true / false
fixeddisk.master	A boolean (true/false) indicating whether the fixed disk is the master disk or a slave disk	true / false
fixeddisk.index	The fixed disk number. The EF will automatically increment this number (starting at 0) based on the number of files provided to the configuration. The template file must substitute this number with the convention expected by the emulator	0
fixeddisk.enabled	A boolean (true/false) indicating whether the fixed disk is enabled (readable) or not	true/false
fixeddisk.swImg	The name of the file that will be attached to the fixed disk	fixeddisk1.img
fixeddisk.cylinders	The number of cylinders, part of the geometry of the fixed disk indicating the address of a physical block of data	4
fixeddisk.heads	The number of heads, part of the geometry of the fixed disk indicating the address of a physical block of data	2
fixeddisk.sectorsPerTrack	The number of sectors per track, part of the geometry of the fixed disk indicating the address of a physical block of data	17

The data model is extensible in such a way that future versions can add new parameters that can be used to configure emulators, such as CPU variables, memory settings, etc. As not all emulators are required to use all the parameters, it is always possible to fall back on a more basic version to provide a simple configuration of an emulator.

Each emulator should have at least a CLI (command line interface) template file, which will create the command line parameters necessary to start the emulator (and perhaps configure it), and in addition (optionally) an XML or properties template file, that completely describes the emulator's configuration.

2.3.1. Template generation example

A simple example aids in illustrating the working of the template engine merging data and template. In this example, the following colour coding is used:

Colour	**Description**	**Usage**

🟪 (pink)	Variables read from data file	The template will read the values of these variables from the data file
🟨 (yellow)	Comments	Explanation in the template file. This is ignored by the template processor
🟦 (turquoise)	Freemarker pseudo code	Code (e.g. if/then/else, macro, etc.) used by the template engine for simple processing. See the Freemarker user guide for a full explanation.
🟫 (dark yellow)	Internally defined map (key-value pairs)	Maps (key-value pairs) defined inside the template (and not in the data file) that can be used within the template for value substitution. See the Freemarker user guide for a full explanation.
🟦 (blue)	Variable substitution (from data file)	Variables enclosed within the ${...} constructs will be replaced by their values, read from the data file
🟩 (green)	Constants (literal text)	Literal text that is copied verbatim from the template file to the output file

Given the following data file:

```
{masterobj:"file1.txt"
floppyDisks:{type:C645_25_170, num:0, digobj:"file2.txt", inserted:true
;            type:C645_25_340, num:1, digobj:"file3.txt", inserted:false}}
```

The template (below) on the left will generate the output on the right (ignore any whitespace, this purely for illustrative purposes):

Template

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

<#-- Drive type definition -->
<#assign driveTypes = {"C645_25_170":"1541", "C645_25_340":"1571"}>

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

##Section: preamble##
##Section: body##
-autostart
${masterobj}



<#list floppyDisks as floppy>
 <@floppyDisk item=floppy/>
</#list>






##Section: postscript##
```

Output

```
##Section: preamble##
##Section: body##
-autostart
file1.txt

-drive8type
1541
-8
file2.txt
+truedrive
-drive9type
1571
-9
file3.txt
-truedrive

##Section: postscript##
```

This example contains two definitions, which are used as substitution lists: *floppyDriveLetter*, which will substitute the correct drive number starting from index 0, as C64 drives start at 8; and *driveTypes*, which will substitute the naming convention used by the EF, into the naming convention the emulator uses. These are called from within the macro (see below), and the parameter passed to it is substituted by the corresponding value, e.g. *C645_25_170* becomes *1541*.

Further there is one macro, the *floppyDisk* macro. It contains some logic that generates command-line floppy drive settings given the *floppyDisk* parameters in the data file. Note that this macro is called twice, once for each of the *floppyDisk* lists defined in the data file, each containing four values. Each list is passed to the macro in turn as a variable called *item*, whose values can be read by calling their name, e.g. *item.inserted*

Note that there is text which has been transcribed literally (e.g. *-autostart*); all items contained in ${...} are substituted by the value from the variables from the data file (e.g. *${masterobj}* is replaced by *file1.txt*).

The different section separators in the configuration file are used to guarantee order for specific configuration options (such as XML headers/footers, command line arguments, etc.). The EF parses the output and splits it into 3 sections based on these text literals. For this example, all information is contained in the body section so the others are unused, but that differs per emulator.

The EF uses the output, prepended with the executable name of the emulator (which is picked up from the database, e.g. VICE.exe), which results a command-line invocation:

```
C:/>VICE.exe -autostart file1.txt -drive8type 1541 -8 file2.txt +truedrive
-drive9type 1571 -9 file3.txt –truedrive
```

Note that for clarity and illustration purposes, the *attributes* header, defining its structure in the *#ftl* tag corresponding to the data model and containing all the parameters that can be used in the template with their default values, is not included in this example. Note that not all emulators will use all of the parameters in this header, even though they are listed. Some parameters are in sub-maps, see for example *fixedDisks* and *floppyDisks*. Attributes that are not used can be set or left undefined; the template engine will ignore them either way.

2.4. Submitting the emulator to the Emulation Framework

For full details on how to submit an emulator to the Emulation Framework, please see [SUG]

Appendix A: Complete template files for the Dioscuri emulator

2.5. Command line template (templateCLI.ftl)[7]

```
<#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"}}>
<#-- Dioscuri 0.4.3 - 0.6.0 configuration template (CLI) -->

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

<#-- Start of preamble -->
<@separator section="preamble"/>
java
-jar
<#-- Start of body -->
<@separator section="body"/>
-r
-c
${configFile}
<@separator section="postscript"/>
<#-- Start of postscript -->
```

2.6. XML template[8]

```
<#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"}}>
<#-- Dioscuri 0.4.3 - 0.6.0 configuration simple template (CLI) -->

<#-- Floppy drive letter definition -->
<#assign floppyDriveLetter = {"0":"A", "1":"B"}>

<#-- Drive type definition -->
<#assign driveTypes = {"FAT3_5_720":"720K", "FAT3_5_1440":"1.44M",
"C645_25_170":"unsupported drive type", "C645_25_340":"unsupported drive
type"}>

<#-- Seperator macro -->
```

[7] http://emuframework.svn.sourceforge.net/viewvc/emuframework/EmulatorArchive/trunk/packages/templates/Dioscuri/templateCLI.ftl?revision=7

[8] http://emuframework.svn.sourceforge.net/viewvc/emuframework/EmulatorArchive/trunk/packages/templates/Dioscuri/templateXML.ftl?revision=299

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

<#-- Floppy disk macro -->
<#macro floppyDisk item>
                <floppy>
<#if item.inserted == "true">
                  <enabled>true</enabled>
                  <inserted>true</inserted>
<#else>
                  <enabled>false</enabled>
                  <inserted>false</inserted>
</#if>
                  <driveletter>${floppyDriveLetter[item.num]}</driveletter>
                  <diskformat>${driveTypes[item.type]}</diskformat>
                  <writeprotected>false</writeprotected>
                  <imagefilepath>${item.digobj}</imagefilepath>
                </floppy>
</#macro>

<#-- Fixed disk macro -->
<#macro fixedDisk item>
                <harddiskdrive>
<#if item.enabled == "true">
                  <enabled>true</enabled>
<#else>
                  <enabled>false</enabled>
</#if>
                  <channelindex>${item.index}</channelindex>
<#if item.master == "true">
                  <master>true</master>
<#else>
                  <master>false</master>
</#if>
                  <autodetectcylinders>true</autodetectcylinders>
                  <cylinders>${item.cylinders}</cylinders>
                  <heads>${item.heads}</heads>

<sectorspertrack>${item.sectorsPerTrack}</sectorspertrack>
                  <imagefilepath>${item.swImg}</imagefilepath>
                </harddiskdrive>
</#macro>

<#-- Start of preamble -->
<@separator section="preamble"/>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<emulator debug="false">
    <architecture name="Von Neumann">
        <modules>
            <bios>
                <sysbiosfilepath>images/bios/BIOS-bochs-
latest</sysbiosfilepath>
                <vgabiosfilepath>images/bios/VGABIOS-lgpl-
latest</vgabiosfilepath>

<ramaddresssysbiosstartdec>983040</ramaddresssysbiosstartdec>

<ramaddressvgabiosstartdec>786432</ramaddressvgabiosstartdec>
                <bootdrives>
                    <bootdrive0>Hard Drive</bootdrive0>
                    <bootdrive1>None</bootdrive1>
                    <bootdrive2>None</bootdrive2>
                </bootdrives>
                <floppycheckdisabled>false</floppycheckdisabled>
            </bios>
            <cpu debug="false">
                <cpu32bit>true</cpu32bit>
                <speedmhz>5</speedmhz>
            </cpu>
            <memory debugaddressdecimal="9295" debug="false">
                <sizemb>16</sizemb>
            </memory>
            <pit debug="false">
                <clockrate>5</clockrate>
            </pit>
            <keyboard debug="false">
                <updateintervalmicrosecs>200</updateintervalmicrosecs>
            </keyboard>
            <mouse debug="false">
                <enabled>false</enabled>
                <mousetype>serial</mousetype>
            </mouse>
            <video debug="false">
                <updateintervalmicrosecs>40000</updateintervalmicrosecs>
            </video>
<#-- Start of body -->
<@separator section="body"/>
            <fdc debug="false">
                <updateintervalmicrosecs>250</updateintervalmicrosecs>
                <#list floppyDisks as floppy>
                  <@floppyDisk item=floppy/>
                </#list>
            </fdc>
            <ata debug="false">
                <updateintervalmicrosecs>100000</updateintervalmicrosecs>
                <#list fixedDisks as fixed>
                  <@fixedDisk item=fixed/>
                </#list>
            </ata>
<@separator section="postscript"/>
<#-- Start of postscript -->
        </modules>
    </architecture>
</emulator>
```