System Maintenance Guide (SMG)

Computer emulator for digital preservation

Version : 1.2
Author : B. Lohman, J.R. van der Hoeven
Date : 05-07-2007
Project : Emulation project

Koninklijke Bibliotheek (National Library of the Netherlands)
Nationaal Archief of the Netherlands

I. Revision history

Revision number	Revision date	Author	Summary of changes
1.1	21-06-2007	J.R. van der Hoeven	Refined some bits
1.2	05-07-2007	B. Lohman	Incorporated new comments project manager

II. Related documents

Document name	Date	Author
User Requirements Document [URD]	21-02-2006	B. Lohman
Architectural Design Document [ADD]	03-03-2006	B. Lohman
Object Design Document [ODD]	25-06-2007	J.R. van der Hoeven
Emulation report by KB/NA [EMU]	20-06-2005	J.R. van der Hoeven
Dioscuri User Manual [DUM]	05-07-2007	B. Lohman J.R. van der Hoeven

III. Table of contents

1 Introduction

1.1 Purpose of this document

This document provides information about how the Dioscuri software is constructed. It is intended for anyone who needs to maintain the software. It does not describe how to use the system; this is covered by the Dioscuri user manual [DUM].

1.2 Scope of this document

This document covers the software developed for the Dioscuri emulation project for the Nationaal Archief of the Netherlands (NA) and the Koninklijke Bibliotheek, National Library of the Netherlands (KB). It includes updated high-level design decisions. Detailed design notes are included as comments in the project source files.

1.3 Definition of Terms

DUM	The Dioscuri User Manual, describes how to use the application.
SMG	The System Maintenance Guide, describes how to maintain the system.
ODD	Object Design Document, describes the modular architecture in detail.
ADD	Architectural Design Document, describes the design of the software.
SRD	The Software Requirements Document, specifies the behaviour of the software system.

1.4 Installation / uninstall

The complete installation/uninstallation procedure is described in the [DUM]. It should be sufficient to mention here that the software is packaged as a stand-alone jar (Java Archive) file, and only requires the installation of the Java Runtime Environment, version 1.5 or higher.

1.5 Installation Verification Procedure

The software will display the current version number in the console when run, along with the compilation date.

1.6 Quick Guide

The complete guide to running the software is described in the [DUM].
To start the software, run the Java Virtual Machine from the command line, indicating it should execute a jar file:

```
java -jar Dioscuri009.jar
```

The console will display some of the logging information, including the version number and compilation date, while the main application can be run from the newly created GUI window. A (boot) image can be selected in the Media menu, and the application will start executing once the “Start process (power on)” option has been selected from the Emulator menu.

To stop the application, open the Emulator menu and select “Stop process (shutdown)” to ensure all data is written to the selected images, after which “Quit” is used to exit the application (or by closing down the GUI window).

2 The Development Environment

2.1 IDE

The system was build using the freely available Eclipse Software Development Kit version 3.1.2 (http://www.eclipse.org/) . In the early stages, tests for JUnit were written as well, for which the Eclipse JUnit plugin v.3.8.1 was used.

The Java Runtime Environment is necessary to run the Java code. Any version of the JRE higher than 1.5 should be sufficient.

2.2 CVS

In the early stages of development a local CVS server was used, which was located at the Nationaal Archief, on the virtual network server //SVAPPL07. The CVS access string used to contact the repository was :pserver:<user>@SVAPPL07:/4717, where <user> is a valid user name.
From version 0.0.9 onwards, the repository was moved to Sourceforge. Details for Sourceforge CVS access can be found on the project page:

http://sourceforge.net/projects/dioscuri

To summarise, this states the following:
Anonymous CVS download, read only:
:pserver:anonymous@dioscuri.cvs.sourceforge.net:/cvsroot/dioscuri [no password]
Developer CVS, read/write access:
:extssh:<DEV_NAME>@dioscuri.cvs.sourceforge.net:/cvsroot/dioscuri [<DEV password>]

There is a sync delay of at most 1 hour between the development and anonymous CVS.
All user and password information is stored by the Dioscuri project administrators.

3 Design

The system design is based on the original computer architecture, following the modular structure of separate components fulfilling a distinct task. This design philosophy has resulted in a structure of packages implementing their hardware equivalent functionality, with a few separate packages filling in extra software functionality.
For full information on the design, refer to [ADD] and [ODD], which were created at the beginning of the project and include more details on each of the components / modules.

4 Implementation

The emulated computer components are based on the functionality of real hardware components. These components are implemented as modules of which each module is a representation of the hardware of that specific component (device). The structure of each module is generally the same as they all inherit from the superclass ‘module.java’. This superclass contains a list of general methods to ensure that the modules are able to communicate with each other.

From version 0.0.9 onwards, the namespace is:

```
nl.kbna.dioscuri.[package]
```

Other packages which do not directly implement a hardware component are the ‘emulator’ , ‘config’, ‘exception’, and ‘logging’ package. See for more information [ODD].

A complete summary of non-component modules is as follows:

Table 4.1: overview of non-component packages

Package [nl.kbna.dioscuri]	Contents	Description
emulator	Emulator.java GUI.java IO.java Modules.java	Main package of the DIOSCURI application, provides configuration parameters, sets up all modules, provides input/output and the entry to the graphical user interface.
config	AtaConfigDialog.java BiosConfigDialog.java BootConfigDialog.java ConfigController.java ConfigurationDialog.java CpuConfigDialog.java DioscuriXmlParams.java DioscuriXmlReader.java DioscuriXmlReaderToGui.java DioscuriXmlWriter.java FdcConfigDialog.java ModuleType.java RamConfigDialog.java SelectionConfigDialog.java SimpleConfigDialog.java XmlConnect.java	Provides configuration handling of the emulation process. All configuration settings are stored in an XML-document (based on an XML schema) and can be altered via the GUI tabs ‘configuration’ or in the XML file directly. During startup of the emulation process all settings are read from the XML file.
exception	CommandException.java CPUInstructionException.java ModuleException.java ModuleUnknownPort.java ModuleWriteOnlyPortException.java StorageDeviceException.java	Provides exception handling for the complete system.
logging	ConsoleFormatter.java FileFormatter.java	Provides logging for all classes.
module	Module.java ModuleBIOS.java ModuleClock.java	Provides classes for inheritance for all hardware components. All modules inherit from the basic class ‘Module’, and each of the distinct components (described below) will inherit from the Module[Type].java

	ModuleCPU.java ModuleDevice.java ModuleDMA.java ModuleFDC.java ModuleIDE.java ModuleKeyboard.java ModuleMemory.java ModuleMotherboard.java ModuleParallelPort.java ModulePIC.java ModulePIT.java ModuleRTC.java ModuleScreen.java ModuleSerialPort.java ModuleVideo.java	file for the appropriate module. This ensures a modular implementation for the emulator, and should allow for easy substitution with different implementations of each component, as outlined in [ADD], [ODD] and [SRD].

Each of the implemented hardware components is placed inside a separate package, to support the modularity of the application. The currently implemented components are as follows:

Table 4.2: overview of component (module) packages

Package [nl.kbna.dioscuri.module]	Contents	Description
bios	BIOS.java	Implementation of a BIOS chip. This package reads and loads the various system BIOSes such as the system BIOS (for booting) and the Video BIOS (providing video output routines). The BIOSes are read from image files included with the application, which were provided as open source implementations.
clock	Clock.java Timer.java	Provides an implementation of a quartz-crystal clock, supplying timer mechanisms to all components that need it. The timer class translates the crystal 'oscillations' into timers with adjustable interval lengths.
cpu	CPU.java Instruction.java Util.java plus 256 individual instructions	Implements a Central Processing Unit, including the instructions in the Intel opcode sets. Also implements all the necessary registers and flags. Each instruction is implemented as a separate class inheriting from 'Instruction.java', which are called from an array in the CPU class. The Util class implements some commonly used methods, such as added and subtracting 2 byte arrays, setting flags, etc.
dma	DMA.java DMA16Handler.java DMA8Handler.java DMAChannel.java DMAController.java DMAModeRegister.java	Provides an implementation of Direct Memory Access, at this point mainly used by the floppy drive to transfer data from/to memory. The implementation is generalised so each component can register a handler to let DMA take care of the transfer for them.
fdc	DMA8Handler.java Drive.java FDC.java Floppy.java	Floppy Drive Controller implementation. Provides a way to load floppy drive images (160K – 2.88M) in the emulator. Uses DMA to read/write data into memory.
ata	AscType.java Atpi.java CDROM.java DiskImage.java DMA8Handler.java	Implementation of a hard disk conforming to the ATA / IDE standard. Provides a way to attach hard disk images (given their geometry in terms of cylinders/heads/sectors per track) in the emulator. Transfers data directly (i.e. via CPU and I/O address space) from the disk to memory.

	IDE.java IDEChannel.java IDECommand.java IDEConstants.java IDEDrive.java IDEDriveController.java IDEDriveType.java IDEStatus.java IDETranslationType.java InterruptReason.java ModuleType.java SenseInfo.java SenseType.java	
keyboard	Keyboard.java KeyBoardController.java KeyboardInternalBuffer.java ScanCodeSets.java TheKeyboard.java	Keyboard input for the emulator. Emulates all components of the traditional keyboard, including an internal buffer and translation from keypress to scancode.
memory	Memory.java	Implementation of Random Access Memory. This is represented as one big array of bytes, with the appropriate getting and setting as required for the CPU instructions.
motherboard	DeviceDummy.java Devices.java Motherboard.java	Motherboard implementation, providing a connecting component for most other peripherals (devices). Handles most of the I/O ports input and output, and delegates the assigned ports to the correct device. Unassigned ports are handled by the dummy device, 'DeviceDummy.java'. Furthermore, it handles the timing mechanism by connecting devices that require a timer to the clock. Finally, it connects devices to the Programmable Interrupt Controller (PIC).
parallelport	ParallelPort.java	A stub for the parallel port implementation. Returns default values for I/O port requests.
pci	PCI.java	A stub for the Peripheral Component Interconnect implementation. Returns default values for I/O port requests.
pic	Init.java PIC.java TheProgrammableInterruptController.java	The Programmable Interrupt Controller implementation. Handles all interrupts generated by individual devices and delegates these to the CPU for scheduling.
pit	Clock.java Counter.java PIT.java	Implementation of a Programmable Interrupt Timer. Generates an interrupt after a set interval, which is used as a timing mechanism for some components, most notable the RTC. It offers three counters of which each counter can be set to a specific mode and interval.
rtc	RTC.java	Real Time Clock implementation. Contains all the data of the CMOS chip, including current system time, date, equipment information, status, boot sequence, etc.
screen	CodePage437.java Screen.java ScreenCanvas.java	An implementation of a CRT/LCD screen component, used in the GUI to display output of the emulator. Emulates a screen of mostly any size, both in text and graphics mode. A 16-byte implementation of a character set is included in the 'CodePage437.java' class.
serialport	SerialPort.java	A stub for the serial port implementation. Returns default values for I/O port requests.
video	AttributeController.java ColourRegister.java CRTControllerRegister.java GraphicsController.java	Implementation of a videocard with no onboard memory. Emulates the standard registers found in a simple VGA card, and translates any video data in memory to visual output on the screen.

	MiscellaneousOutputRegister.java ModeControlRegister.java Pixel.java SequencerRegister.java TextModeAttributes.java VGA.java VideoCard.java	

The above packages implement a full emulator, with the stubs as mentioned. The GUI class in nl.kbna.dioscuri.emulator package is the main class, as the GUI will provide the main method of interaction for the user. From the selected options available, the emulator will perform the requested function.

5 Creating a Release

5.1 Version of the release

First of all, make sure that the version number is correct. During the Dioscuri project, the versioning system maintains three numbers, denoted as: `x.y.z`

x = Major number. Is upgraded when significant jumps in functionality occurs. A zero (0) number denotes a pre-production version (alpha or beta). Higher versions (> 0) have to be stable and mature.
y = Minor number. Is upgraded when minor features are introduced or significant bug fixes have been added.
z = Revision number. Is upgraded when minor bugs are fixed.

See http://en.wikipedia.org/wiki/Software_versioning for more information.

5.2 Generating JAR

In Eclipse, a release can be created via the "File->Export…->JAR File menu". Select all the relevant files to be included (the minimum for a complete install is the bin folder); this then creates a stand-alone file that can be run from any computer where the Java Runtime Environment (version 1.5.0 or higher) is installed.

The main class, `nl.kbna.dioscuri.emulator.GUI` must be selected as an entry point to allow the JAR file to be run.

5.3 Finalising the release (includes)

As of this writing, there are certain files, that must be located in predefined directories that must be included for the release to work properly. These are:

Table 5.1: required files to include in the release

File	Directory	Description
logging.properties	./	Provides the logging settings for the logger
	./log	Provides a directory location for the log files (this can be changed in the logging.properties file)
DioscuriConfig.xml DioscuriConfig.xsd	./config	Provides the configuration settings for the emulator (based on an XML schema)
System BIOS Default: BIOS-bochs-latest)	<user selectable> Default: ./config	The system BIOS used for booting. This can be set via the GUI menus
VGA BIOS Default: VGABIOS-lgpl-latest	<user selectable> Default: ./config	The VGA BIOS containing video routines. This can be set via the GUI menus
Floppy/ hard disk image		A bootable floppy or hard disk image. This can be set via the GUI menus

6 Creating a Build Environment

In Eclipse, a build environment can be creating in multiple ways. Two of these are described below.

6.1 Creating a build environment from CVS

If a Concurrent Version System has been set up, Eclipse can download the source files from the server and put them into a project, allowing to build the application.

To add a new project from a CVS repository, go to File->New->Project->Checkout Projects from CVS. For the Dioscuri sourceforge server, enter the following location:

```
Host: dioscuri.cvs.sourceforge.net
Repository path: /cvsroot/dioscuri
Connection type: pserver
Port: <DEFAULT PORT>
User: anonymous
Password: <NONE>
```

6.2 Creating a build environment from local files

To create a build environment from local files, go to File->New->Project. Create a new project.
Once in the project, select File->Import and select the type of import related to the source of the local files to import these files into the project.

7 References

A list of reference for all information regarding a computer system, including any links to online sources, is kept in a document called “The PC Reference Guide”.

At the time of writing, this document, as well as all other important documentation, will be published on the Sourceforge project page, http://dioscuri.sourceforge.net