Ftklipse – Design and Implementation of an Extendable Computer Forensics Environment Specification Design Document

Marc-André Laverdière  Serguei A. Mokhov  Suhasini Tsapa
Djamel Benredjem

April 24, 2006
# Contents

1 Introduction
   1.1 Purpose .................................................. 1
   1.2 Scope .................................................... 1
   1.3 Definitions and Acronyms ............................... 2

2 System Overview
   2.1 Architectural Strategies ................................. 4
   2.2 System Architecture ..................................... 5
      2.2.1 Module View ......................................... 5
   2.3 Execution View ........................................... 6
      2.3.1 Runtime Entities .................................... 6
      2.3.2 Communication Paths ................................. 6
      2.3.3 Execution Configuration .............................. 7
   2.4 Coding Standards and Project Management .......... 7

3 Detailed System Design
   3.1 Class Diagrams ........................................... 8
   3.2 Data Storage Format ..................................... 9
      3.2.1 Entity Relationship Diagram ........................ 9
      3.2.2 External Systems and Databases .................. 11
      3.2.3 Log File Format .................................... 17
   3.3 Directory and Package Organization .................. 18
   3.4 Plug-Ins ................................................ 18
CONTENTS

3.5  User Interface Design  ....................................  22
     3.5.1  Appearance  ........................................  22

4  Conclusion  ..................................................  26
    4.1  Summary of Technologies Used  .......................  26
    4.2  Summary of Tools Added  ..............................  27
    4.3  Summary of Difficulties  .............................  27
    4.4  Limitations and Technological Restrictions ..........  27
    4.5  Future Work and Work-In-Progress ..................  28
    4.6  Acknowledgments  ....................................  28

References  .....................................................  30
Chapter 1

Introduction

This chapter briefly presents the purpose and the scope of the work on the Ftklipse project with a subset of relevant definitions and acronyms. All these aspects are detailed to some extent later through the document.

1.1 Purpose

To design and implement a plugin-based environment that allows to integrate forensic tools working together to support programming tasks and addition of new tools. Integration is done through GUI components.

1.2 Scope

The end-product environment must have user friendly GUI, configuration capabilities, plug-in capabilities to insert/inject new tools, case management, and chain of custody capabilities, along with evidence gathering capabilities, evidence preservation capabilities, and, finally report generation capabilities. A subset of these requirements has been implemented in Ftklipse, which is detailed throughout the rest of this document.
1.3 Definitions and Acronyms

**Cryptographic Hash Function** Function mapping input data of an arbitrary size to a fixed-sized output that is highly collision resistant.

**Digital evidence** Information stored or transmitted in binary form that may be relied upon in court.

**dcfldd** Enhanced DD imager with built-in hashing, works like *dd* from command line.

**Hashing on-the-fly** *dcfldd* can hash the input data as it is being transferred, helping to ensure data integrity.

**Status output** *dcfldd* can update the user of its progress in terms of the amount of data transferred and how much longer operation will take.

**Flexible disk wipes** *dcfldd* can be used to wipe disks quickly and with a known pattern if desired.

**Image/wipe verify** *dcfldd* can verify that a target drive is a bit-for-bit match of the specified input file or pattern.

**Multiple outputs** *dcfldd* can output to multiple files or disks at the same time.

**Split output** *dcfldd* can split output to multiple files with more configurability than the split command.

**Piped output and logs** *dcfldd* can send all its log data and output to commands as well as files natively.

**Documentation** Written notes, audio/videotapes, printed forms, sketches, and/or photographs that form a detailed record of the scene, evidence recovered, and actions taken during the search of the scene.

**JVM** The Java Virtual Machine. Program and framework allowing the execution of program developed using the Java programming language.
Magnetic media A disk, tape, cartridge, diskette, or cassette that is used to store data magnetically.

Steganography It simply takes one piece of information and hides it within another. Computer files (images, sounds recordings, even disks) contain unused or insignificant areas of data. Steganography takes advantage of these areas, replacing them with information (encrypted mail, for instance). The files can then be exchanged without anyone knowing what really lies inside of them. For example, an image of the space shuttle landing might contain a private letter to a friend. A recording of a short sentence might contain your company’s plans for a secret new product. Steganography can also be used to place a hidden “trademark” in images, music, and software, a technique referred to as watermarking.

SWT The Standard Widget Toolkit [Con06c], a set of graphical user interface components provided by the Eclipse framework.

Temporary and swap files Many computers use operating systems and applications that store data temporarily on the hard drive. These files, which are generally hidden and inaccessible, may contain information that the investigator finds useful.
Chapter 2

System Overview

In this chapter, we examine the architecture of our implementation of Ftk-lipse. We first introduce our architectural philosophy before informing the reader about the Siemens Four View Model, an architectural methodology for the conception of large-scale software systems. Afterwards, we examine each of the view, as architected for our system. Finally, we conclude with other software engineering matters that were of high importance in the development of our implementation.

2.1 Architectural Strategies

Our principles are:

Platform independence We target systems that are capable of running a JVM.

The Eclipse plug-in based environment slightly imitating the MVC (Model-view-Controller) pattern, to map the traditional input, processing, output roles into the GUI realm. In Eclipse model, a plug-in may be related to another plug-in by one of two relationships:

Dependency The roles in this relationship are dependent plug-in and prerequisite plug-in. A prerequisite plug-in supports the function
of a dependent plug-in.

**Extension** The roles in this relationship are host plug-in and extender plug-in. An extender plug-in extends the functions of a host plug-in.

**Database independent API** will allow us to swap database engines on-the-fly.

**Reasonable Efficiency** We will architect and implement an efficient system, but will avoid advanced programming tricks that improve the efficiency at the cost of maintainability and readability.

**Simplicity And Maintainability** We will target a simplistic and easy to maintain organization of the source.

**Architectural Consistency** We will consistently implement our architectural approach.

**Separation of Concerns** We will isolate separate concerns between modules and within modules to encourage reuse and code simplicity.

### 2.2 System Architecture

#### 2.2.1 Module View

**Layering**

We divided our application between layers. The top level has a front-end and a back-end. The frontend comprised a collection of GUI modules provided by and customized from eclipse as well as custom-designed by the team. The backend consists of supporting functionality and specifically database management, report generation, and external tool invocation.
Interface Design

Several interfaces had to be designed for the architecture to work. All the backend modules have an interface they expose to the frontend to use. Thus, there are interfaces between, GUI-to-External-Tools, GUI-to-Database, and GUI-to-Report-Generation. All these are designed to be swappable and highly modular so any component series can be replaced at any time with little or no change to the code. The interfaces (FtklipseCommonDatabaseBridge and IDatabaseAdapter, ITool and IToolExecutor, and IReportGenerator and ReportGeneratorFactory) are presented in the detailed design chapter.

2.3 Execution View

2.3.1 Runtime Entities

In the case of our application, there is hosting run-time environment that of Eclipse. The application can run within Eclipse IDE or be a stand-alone with a minimal subset of the Eclipse run-time. By nature, a JVM machine is executing all the environment and all GUI-based applications are multi-threaded to avoid blockage on user’s input. Additionally, depending on the database engine used behind the scenes, it may as well be multi-threaded to provide concurrent access and connection pooling.

2.3.2 Communication Paths

It was resolved that the modules would all communicate through message passing between methods. Communication to the database depends on the database adapter, and in our sample implementation is done through and in-process JDBC driver. Additionally, Java’s reflection is used to discover instantiation communication paths at run-time for pluggable modules.
2.3.3 Execution Configuration

Execution configuration in Ftklipse has to do with where its data directory is. The data directory is always local to where the application was ran from. The directory contains the main case database in the ftklipsedb.* files as well as numerical directories with case ID with imported evidence files. Additional configuration for application is located in plugin.properties and plugin.xml files.

2.4 Coding Standards and Project Management

In order to produce high-quality code, we decided to normalize on the OpenBSD style. We also decided to use javadoc source code documentation style for its completeness and the automated tool support. We used Subversion (svn) \cite{Co07} in order to manage the source code, makefile, and documentation revisions provided by SourceForge.net.
Chapter 3

Detailed System Design

- Case management: Investigations are organized by cases, which can contain one or more evidences. Each evidence can contain one or more file system images to analyze;

- Evidence Gathering using integrated and plug-in tools;

- Evidence Integrity validation using a hash function;

- Evidence Import from any media to an existing case;

- Logging of all operations performed on the evidence;

- Validation of integrity of evidence after each operation over it;

- Display of evidence in read-only mode either in ASCII, Unicode or Hex formats;

- Recording of investigative notes for each piece of evidence;

- Capability to extract a part of the evidence into another file;

- Capability to copy and rename the copy of the evidence;

- Generation of reports in PDF and LaTeX formats that includes listing of the evidence in the case, a printout of selected parts of the evidence,
the investigative notes related to selected parts of the evidence and a customized executive summary, introduction, and conclusion. It also integrates the chain of custody information for each part of the evidence displaying the principal, timestamp and operation performed on the evidence.

- An extendable set of tools through a plug-in architecture;
- General as well as tool-specific defaults and configuration screens;

### 3.1 Class Diagrams

We have a number of class diagrams representing the major modules and their relationships. Please located the detailed description of the modules in the generated HTML of javadoc or the javadoc comments themselves in the `doc/javadoc` directory.

The basic UI classes are in Figure 3.1. The prototype internal access control classes are in Figure 3.2. The main database abstraction is in Figure 3.3. Next, concrete database adapters are in Figure ??

Further, the database- and UI-independent database objects data structures are in Figure 3.5. The report generation-related API is in Figure 3.6. Finally, the external tools invocation framework is in Figure 3.7.

### 3.2 Data Storage Format

This section is about data storage issues and the details on the chosen underlying implementation and ways of addressing those issues.

#### 3.2.1 Entity Relationship Diagram

The ER diagram of the underlying SQL engine we chose is in Figure 3.8. The database is pretty simple as the `case.data` field is a BLOB to which the `Case` data structure is serialized. The `id.count` table is simply there to contain the maximum ID used across the database objects in the application.
CHAPTER 3. DETAILED SYSTEM DESIGN

Figure 3.1: Class Diagram for the basic User Interface
It is updated on application close, so when the application is loaded back again, it sets its internal ID from the database properly for newly created cases and other objects.

The database is slatted for extension with some code map data for the UI as well as log facilities later on for better reporting, like who, what, when, etc.

### 3.2.2 External Systems and Databases

The database engine the Ftklisper application talks to is abstracted away so that the actual engine particularities (e.g. SQL queries or XML atoms) are not visible to the application thus making it database-engine independent. The provision was made to have SQL, XML, JavaSpaces [Mam05], or raw object serialization databases. The actual external database engine used in the demo version of the toolkit, is the HSQLDB [The08] database, which is implemented in Java itself and has an in-process execution capability. This database engine is started automatically within the same process as an
Figure 3.3: Class Diagram for the Database Root Package
CHAPTER 3. DETAILED SYSTEM DESIGN

Figure 3.4: Class Diagram for the Database Adapters
Figure 3.5: Class Diagram for the Database Objects
CHAPTER 3. DETAILED SYSTEM DESIGN

Figure 3.6: Class Diagram for the Report Generation
Figure 3.7: Class Diagram for the Backend Tools Framework
CHAPTER 3. DETAILED SYSTEM DESIGN

Figure 3.8: Simple ER Diagram of the Internal Database

application when a first connection is made. It is shutdown when application exits. This choice is justified by simplicity and does not require an external database server to be set up. This external implementation of the engine is in lib/hsqldb.jar.

The database-produced files are stored in the data directory relative to the current execution environment. The files are ftklipsedb.properties and ftklipsedb.script. The former describes the global database settings and the latter is the serialized database itself, including DDL DML statements to reproduce the database. Both are managed by the HSQLDB engine itself. Originally when deploying the application, neither may present. They will be created if not present when Ftklipse starts.

Another external system we rely on in the form of library is the PDF generation library iText [LS06] [LS06], which is in lib/itext.jar. This library is used in PDFReportGenerator to produce a PDF copy of the case data stored in the database.

3.2.3 Log File Format

The log is saved in the ftklipse.application.log. As of this version, the file is produced with the help of the Logger class that has been imported from MARF [The09]. (Another logging facility that was considered but
not yet implemented is the Log4J tool [AGS+06], which has a full-fledged logging engine.) The log file produced by Logger has a classical format of [time stamp]: message. The logger intercepts all attempts to write to STDOUT or STDERR and makes a copy of them to the file.

### 3.3 Directory and Package Organization

In this section, we introduce the reader to the structure of the folders for ftkclipse. Please note that Java, by default, converts sub-packages into sub-folders, which is what we see in Figure 3.9.

Please also refer to Table 3.1 and Table 3.2 for description of the data contained in the folders and the package organization, respectively.

### 3.4 Plug-Ins

In order to allow tools to be plugged in, we use Eclipse’s default mechanism, which requires to define and export and extension point. The extension point Table 3.4 defines a set of properties that are mostly used to populate the user interface as well as providing the interfaces that must be implemented in order to contribute a plug-in to ftkclipse.

Any third party can contribute a plug-in tool in ftkclipse by creating an Eclipse plug-in project that chooses to extend ca.concordia.ciise.ftkle. Those plug-ins can afterwards be installed manually in the Eclipse folder’s sub-root, or using Eclipse’s built-in installer and updater. When installed properly, ftkclipse will detect them without the need to update any configuration file or perform other similar administrative works.

Each plug-in is responsible for implementing its own dialog(s) and may optionally define its own parameters persistence mechanism, although our API strongly suggests the use of Eclipse’s technology to do so.

In order that all tools can have access to information from the user interface, and that the user interface can have access to information about
| Folder     | Description                                                                 |
|------------|-----------------------------------------------------------------------------|
| bin        | Directory containing the compiled files. All package names described here are also present under this directory. |
| data       | Directory containing the case database as well as subdirectories for each of the cases. |
| doc        | Project’s documentation                                                     |
| example_evidence | Demo evidence that can be used in the projects                             |
| icons      | icons useable for branding and decorating the application                   |
| lib        | External libraries used by ftklipse                                          |
| META-INF   | Project’s meta-information that would be included in a JAR bundle           |
| references | Some useful references on the web on Eclipse development                    |
| schema     | Project’s extension point definitions                                       |
| src        | Directory containing the source code files. All package names described here are underneath this directory. |
| tools      | Precompiled tools to use. Also organized hierarchically.                    |

Table 3.1: Details on folder structure
| Package | Description |
|---------|-------------|
| ca.concordia.ciise.ftklipse | Ftklipse’s root package name |
| ca.concordia.ciise.ftklipse.accesscontrol | Ftklipse’s access control model |
| ca.concordia.ciise.ftklipse.database | Ftklipse’s database module |
| ca.concordia.ciise.ftklipse.database.adapters | Database adapters |
| ca.concordia.ciise.ftklipse.database.connection | Database connection objects |
| ca.concordia.ciise.ftklipse.database.objects | Object model that is saved and restored from the database |
| ca.concordia.ciise.ftklipse.database.reporting | Reporting sub-module |
| ca.concordia.ciise.ftklipse.database.util | Database utility classes |
| ca.concordia.ciise.ftklipse.junit | Some JUnit tests |
| ca.concordia.ciise.ftklipse.tools | Tool execution module, not including GUI screens |
| ca.concordia.ciise.ftklipse.tools.executors | Tool execution adapters for the underlying platform |
| ca.concordia.ciise.ftklipse.tools.linux | Tool adapters for Linux tools |
| ca.concordia.ciise.ftklipse.tools.windows | Tool adapters for Windows tools |
| ca.concordia.ciise.ftklipse.ui | Ftklipse’s user interface classes |
| ca.concordia.ciise.ftklipse.ui.actions | Eclise actions for the menu and right-click menu |
| ca.concordia.ciise.ftklipse.ui.tools | User interfaces for the tools provided by default |
| ca.concordia.ciise.ftklipse.util | Utility classes |

Table 3.2: Package organization
### Table 3.3: Extension Point for Third-Party Plug-Ins

| Attribute   | Type       | Summary                                                                 |
|-------------|------------|-------------------------------------------------------------------------|
| id          | string     | unique identifier for the tool                                          |
| name        | string     | name of the tool. Not currently used                                    |
| class       | ITool      | class implementing our standard interface for the tool execution       |
| type        | enumeration| one of collection, analysis or other. Used for structuring tools in menus|
| parameter   | string     | for future use, allowing a tool to register more than once but with different parameters that would let it act differently. |
| outputFile  | string     | for future use, allowing a tool to register and specify a default output file for its operation |
| category    | string     | for future use, in order to group tools for batch collection or batch analysis of data |
| platform    | enumeration| either win or unix. To specify on which platform the tool operates       |
| inBatchMenu | boolean    | whether the plug-in requires to be registered in batch processing menus |
| inRightClickMenu | boolean    | whether the plug-in requires to be registered in the right-click menu |
| friendlyName| string     | short name of the tool, for displaying the user                         |
| uiclass     | ITooUI     | class implementing our standard interface for the tool execution       |
all tools, we used a set of registry singletons which are responsible to conserve
single instances of the information.

Plug-in developers would thus find the WidgetRegistrySingleton to
be very helpful, as it notably returns a reference to the case and evidence
tree, which can be queried to find the active evidence and active projects.

As such, we do not implement a strict Model-View-Controller (MVC)
architecture, but merely a model that is similar to it, as the plug-ins are
trusted not to modify and user interface elements.

3.5 User Interface Design

3.5.1 Appearance

Ftklipse is implemented using JFace and SWT, technologies provided within
the Eclipse framework. It consists of a single window composed of a menu
bar on the top, a tree structure on the left-hand side, and a multiply-tabbed
area at the centre.

This central area displays information about the currently opened evi-
dence file or case information from the case database. Please refer to Fig-
ure 3.10 and Figure 3.11 for screenshots of the implementation.
Figure 3.9: Folder Structure of the Project
Figure 3.10: User Interface Showing the Case Introduction
Figure 3.11: User Interface Showing the Evidence Information and Notes
Chapter 4

Conclusion

Despite the technological difficulties and limitations the chosen approach seems very promising. Highly modular design allows also swapping module implementations from one technology to another if need be making it very extensible. Case management, very strong backend architecture for Tools, Database, and Report Generation. Eclipse UI integration are strong points of this project.

4.1 Summary of Technologies Used

The following were the most prominent technologies used throughout the project:

- Eclipse IDE \cite{E08}
- iText PDF generation library \cite{LS06}
- HSQLDB lightweight embedded Java SQL engine \cite{The08}
- Visual Editor for Eclipse \cite{Con06e}
4.2 Summary of Tools Added

The number of testing tools is not large and many more could be added from various resources [ht06], however, there were enough for many test cases given time limitations. The following Linux tools were used for testing and worked:

- stegdetect [Pro04], stegbreak, stegdeimage, magic2mime,
- file,
- strings,
- dcf1dd.

4.3 Summary of Difficulties

Learning curve for Eclipse plug-in and UI frameworks [Bol03, Con06b, Gal02, KFL02, Pro05, Bur06, Con06f, Con06d, Con06a] with large volumes of APIs and documentation was overwhelming at the beginning and making things like right- and double-click to work as well as SWT-based [Con06c]. UIs was sometimes non-trivial.

4.4 Limitations and Technological Restrictions

The Eclipse framework imposes some technological restrictions in user interface programming on two major areas that impacted our design.

The first restriction is that the menu items are populated by ‘Actions’, and that it is impractical to have a different Action instance for each menu item for each possible item the menu can interact with. For example, the right-click menu, although capable of being dynamically generated every time, requires to perform an action based on the currently selected item. Re-creating the menu on each right-click from new objects is expensive both in memory and computationally, risking to create an interface with a high
response time to the user, which impacts negatively on usability. Another option is to create a cache of such items and change internal data members related to the selected widget before displaying the menu. This approach increases complexity and was not considered to be a good solution in our context, due to the complexity of propagating this strategy to existing and future options. Finally, we considered having a central access point to the information on the selected items that would be opaque to the underlying data types creating the tree hierarchy. This last approach, albeit less ‘pure’ object-oriented design, was retained for its ease of use in prototyping new features, as well as the assumed atomicity of GUI operation (i.e. it should not be possible to change the selection while the handling of the right-click on the selection is running).

The second restriction is Eclipse’s all-or-nothing approach to plug-in development. As far as we understood the framework, it is possible to use Eclipse’s internal data types and existing advanced widgets only when extending the framework in our plug-in. A plug-in that would choose not to follow Eclipse’s organization (which is our case) could thus not have access to pre-existing file browsers and variety of editors. As such, the tree hierarchy, mouse handling, and data visualization needed to be reimplemented from lower-level SWT components.

### 4.5 Future Work and Work-In-Progress

Allow addition of tools dynamically though GUI Improve case management with full chain of custody (backend is done for this) Integration of the hexadecimal editor plugin \[\text{Pal06}\]

### 4.6 Acknowledgments

- Dr. Mourad Debbabi for the excellent course.
- Open Source community for Eclipse, HSQLDB, iText
• Dr. Peter Grogono for \LaTeX\ introductory tutorial [Gro01]
Bibliography

[AGS+06] N. Asokan, Ceki Gulcu, Michael Steiner, IBM Zurich Research Laboratory, and OSS Contributors. log4j, Hierarchical Logging Service for Java. apache.org, 2006. http://logging.apache.org/log4j/

[Bol03] Azad Bolour. Notes on the Eclipse Plug-in Architecture. eclipse.org, July 2003. http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html

[Bur06] Ed Burnette. Rich Client Tutorial. eclipse.org, February 2006. http://www.eclipse.org/articles/Article-RCP-1/tutorial1.html

[Col07] Inc. CollabNet. Subversion (SVN). tigris.org, 2007. http://subversion.tigris.org/

[Con06a] Contributors. Creating and using Extension Points. refractions.net, 2006. http://udig.refractions.net/confluence/display/DEV/1+Creating+and+Using+Extension+Points

[Con06b] Contributors. Eclipse Plugin Central - Forums. eclipseplugincentral.com, 2006. http://www.eclipseplugincentral.com/PNphpBB2+file-viewforum-f-74.html

[Con06c] Contributors. SWT: The Standard Widget Toolkit. eclipse.org, 2006. http://www.eclipse.org/swt/
[Con06d] Contributors. *User Guide: Building a Rich Client Platform application*. eclipse.org, 2006. http://help.eclipse.org/help31/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/rcp.htm

[Con06e] Contributors. *Visual Editor Project*. eclipse.org, 2006. http://wiki.eclipse.org/index.php/Visual_Editor_Project

[Con06f] Contributors. *Workbench User Guide: Plugging into the workbench*. eclipse.org, 2006. http://help.eclipse.org/help31/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/workbench.htm

[E+08] Eclipse contributors et al. Eclipse Platform. eclipse.org, 2000-2008. http://www.eclipse.org, last viewed April 2008.

[Gal02] David Gallardo. *Developing Eclipse plug-ins*. ibm.com, December 2002. http://www-128.ibm.com/developerworks/opensource/library/os-ecplug/?Open&ca=daw-ec-dr

[Gro01] Peter Grogono. *A $\LaTeX$2e Gallimaufry. Techniques, Tips, and Traps*. Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, March 2001. http://www.cse.concordia.ca/~grogono/Writings/gallimaufry.pdf, last viewed May 2008.

[htt06] http://www.dmares.com. *Software Links for Forensics Investigative Tasks*. 2006. http://www.dmares.com/maresware/SITES/tasks.htm

[KFL02] Dan Kehn, Scott Fairbrother, and Cam-Thu Le. *Internationalizing your Eclipse plug-in*. ibm.com, June 2002. http://www-128.ibm.com/developerworks/opensource/library/os-i18n/

[LS06] Bruno Lowagie and Paulo Soares. *iText, a Free Java-PDF library*. lowagie.com, 2006. http://www.lowagie.com/iText/
[Mam05] Qusay H. Mamoud. *Getting Started With JavaSpaces Technology: Beyond Conventional Distributed Programming Paradigms*. Sun Microsystems, Inc., July 2005. http://java.sun.com/developer/technicalArticles/tools/JavaSpaces/.

[Pal06] Marcel Palko. *Eclipse Hex Editor Plugin*. sourceforge.net, 2006. http://ehep.sourceforge.net/.

[Pro04] Niels Provos. Steganography detection with stegdetect, 2004. http://www.outguess.org/detection.php.

[Pro05] Emmanuel Proulx. *Eclipse Plugins Exposed*. onjava.com, February 2005. http://www.onjava.com/pub/a/onjava/2005/02/09/eclipse.html.

[The08] The hsqldb Development Group. HSQLDB – lightweight 100% Java SQL database engine v.1.8.0.10. hsqldb.org, 2001–2008. http://hsqldb.org/.

[The09] The MARF Research and Development Group. The Modular Audio Recognition Framework and its Applications. SourceForge.net, 2002–2009. http://marf.sf.net last viewed December 2008.
Index

API
  Case, 9
  FtkclipseCommonDatabaseBridge, 6
  IDatabaseAdapter, 6
  IReportGenerator, 6
  ITool, 6
  IToolExecutor, 6
  Logger, 17, 18
  PDFReportGenerator, 17
  ReportGeneratorFactory, 6

Data Storage Format, 9
  ER Diagram, 9
  External Systems and Databases, 11
  Log File Format, 17

Design, 8
  Class Diagrams, 9
  Data Storage Format, 9

Detailed System Design, 8

Directory Structure, 18

Files
  data, 7, 17
  doc/javadoc, 9
  ftkclipse.application.log, 17
  ftklipsedb.*, 7
  ftklipsedb.properties, 17
  ftklipsedb.script, 17
  lib/hsqldb.jar, 17
  lib/iText.jar, 17
  plugin.properties, 7
  plugin.xml, 7

Introduction, 1
  Definitions and Acronyms, 2
  Purpose, 1
  Scope, 1

Java, 11

Libraries
  HSQLDB, 11
  iText, 17
  JavaSpaces, 11
  MARF, 17

Package Organization, 18

System Overview, 4
  Architectural Strategies, 4

Tools
INDEX

dcfldd, 2, 27
dd, 2
file, 27
javadoc, 7
magic2mime, 27
MARP, 17
stegbreak, 27
stegdeimage, 27
stegdetect, 27
strings, 27
svn, 7