Cadmium – version 1.3

http://cadmium.x9c.fr

September 19, 2009

Contents

Chapter 1

Introduction

This document is structured in seven chapters. After this introductory chapter, the following ones explain how to build Cadmium, how to use Cadmium, and how to interface it with Java. Finally, the last chapters expose how to use Cadmium for scripting, and how to program servlets and GUI applications.

1.1 What is Cadmium ?

Cadmium is both a Java program and library that allow to combine Objective Caml[1] and Java[2]. As a program, Cadmium acts as a replacement for the `ocamlrun` program (the OCaml virtual machine); it thus allows to run Objective Caml bytecode files. As a library, Cadmium acts as the runtime support of Cafesterol-compiled files. Cafesterol, available at `http://cafesterol.x9c.fr`, is a compiler that is the Java counterpart of `ocamlc`/`ocamlopt`.

Cadmium is designed to be a *pure* Java program working on any JVM version 1.6 or above, it does not rely on JNI / native code. It is also different from JavaCaml[3] in that JavaCaml needs to use a special runtime when compiling with `ocamlc`. At the opposite, Cadmium can run an OCaml bytecode file compiled with traditional options. It strives to be as compatible as possible with `ocamlrun`.

1.2 Why is Cadmium useful ?

Although the project started to acquire some knowledge about the OCaml runtime environment, we regard the result as useful. From an OCaml point of view, it gives access to all Java libraries and their related manpower (to easily interface with Java libraries, one is advised to check the Nickel program available at `http://nickel.x9c.fr`). From a Java point of view, it gives access to the greater expressiveness of OCaml (higher-order functions, functors, stricter type discipline, *etc.*). On a side note, Cadmium also allows the deployment of OCaml applications running in a sandbox, using the security model underlying the JVM.

[1] The official Caml website can be reached at `http://caml.inria.fr` and contains the full development suite (compilers, tools, virtual machine, *etc.*) as well as links to third-party contributions.

[2] `http://java.sun.com`

[3] `http://www.ocaml-programming.de/javacaml/manual/`

Of course, Cadmium and Cafesterol programs are very slow compared to `ocamlopt`-compiled ones. Nevertheless, we think that OCaml could be used to glue some Java *low-level* fragments or for prototyping (and then rewriting the most critical parts). Following this path, the Cadmium/Cafesterol couple is maybe best compared with scripting languages, where it can compete in terms of speed.

1.3 Status

In its 1.3 version, Cadmium is already highly compatible with the original Objective Caml implementation. It supports all language constructs and the following libraries:

- standard library;
- bigarray library;
- dbm library;
- dynlink library (both bytecode and native versions);
- graph library;
- labltk library[4];
- num library;
- str library;
- both thread libraries;
- unix library;
- camlzip library[5].

The document "Cadmium – Primitives compatibility", available at `http://cadmium.x9c.fr/downloads.html` gives the compatibility of Cadmium with the original implementation on a per-primitive basis. Since version 1.3, it is possible to use the OCaml debugger; however, programs can only be debugged using eihter `ocamldebug.jar` or `ocamldebug-standalone.jar` (not the original `ocamldebug` debugger), and only manual mode is available for program loading.

Additional modules are shipped with Cadmium. The following modules provide access to Java packages and classes:

- `Cadmium` and `CadmiumObj` give access to some Java base elements, including reflection;
- `CadmiumSwiXml` and `CadmiumSwiXmlObj` provides bindings to the SwiXml library (*cf.* chapter 7);
- `CadmiumMath` provides acess to both `java.lang.Math` class and `java.math` package;
- `CadmiumJDBC` provides acess to the `java.sql` package;

[4]Based on both Jacl (version 1.4.1 available at `http://tcljava.sourceforge.net`) and Swank (version 2.1.4 available at `http://kenai.com/projects/swank`).

[5]Library for zip/gzip/jar manipulation – available at `http://cristal.inria.fr/~xleroy/software.html`.

- `CadmiumServlet` provides acess to both `javax.servlet` and `javax.servlet.http` packages (*cf.* chapter 6);
- `CadmiumXML` provides functions for XML parsing (both SAX and DOM models), transformations, and XPath queries;

For more information, one should read the ocamldoc-generated documentation for these modules.

As a technical aside, Cadmium is a big-endian 32-bit implementation of OCaml (which itself can be either big- or little-endian, and either 32- or 64-bit). It may be useful to the developer / user to know that, but OCaml gracefully handles architecture differences.

1.4 Contributing

In order to improve the project, I am primarily looking for testers and bug reporters. Pointing errors in documentation and indicating where it should be enhanced is also very helpful.
Bug reports can be made at `http://bugs.x9c.fr`.

1.5 Sketchy roadmap

1.x versions (late 2009 / early 2010): features, enhanced bindings to JDK packages

2.x versions (late 2010): work on performance issues

after the previous phases have been completed, the version number will be the one of the corresponding Objective Caml release

Chapter 2

Building Cadmium

2.1 Dependencies

The core of Cadmium has no dependency, but Cadmium extensions (resulting from the merge of the now-deprecated projects named Cadmium-Dbm, Cadmium-SwiXml, Cadmium-Servlet, and OCamlScripting) depend on external elements. Table 2.1 shows the dependencies of the various Cadmium extensions. Of course, a complete installation of Objective Caml 3.11.1 is also needed.

Cadmium extension	Dependencies
JDBM implementation for Dbm	JDBM classes in `lib/w3c-jdbm` directory
Servlet support	`servlet-api.jar` in `lib` directory
SwiXml support	`swixml.jar` in `lib` directory
Scripting support	all the jar files produced by the installation of Cafesterol, as well as all jar files it depends upon in `lib` directory

Table 2.1: Cadmium dependencies.

The compilation process is designed to try to compile an extension only if its depencies are satisfied. This means, for example, that the support for servlets will not be built if the `servlet-api.jar` file is missing from the `lib` directory.

More, one should notice that prebuilt Cadmium jar file are available at `http://cadmium.x9c.fr/downloads.html`.

2.2 Compilation of the Java part

The Java part of Cadmium can be built from sources using Ant[1] (at least version 1.7.1) under Java 1.6. Before invoking Ant, one is advised to edit `build.properties` to parametrize Ant targets; properties are not described here as they are self-explanatory. The following targets are available:

`clean-classes`, `clean-javadoc`, `clean` respectively deletes files from classes directory, javadoc directory, and both directories;

[1] Apache Ant is a build tool, available at `http://ant.apache.org`.

`compile` compiles source files into class files;

`deploy` (and other targets whose name begins with "deploy") compiles source files into class files and then creates the Cadmium jar files (in deploy directory);

`init` initializes properties for conditional compilation;

`install` copies the Cadmium jar into the Objective Caml directory;

`javadoc` generates javadoc in two flavours: `public` for ordinary user and `dev` for developper;

`style` generates Checkstyle[2] report.

2.3 Compilation of the Objective Caml part

To build the Objective Caml part of Cadmium library, the following `make` targets are available:

`all` compiles all files;

`library` compiles support library (C);

`cma` compiles support and bytecode libraries;

`cmxa` compiles support and native libraries;

`cmja` compiles support and Cafesterol libraries;

`html-doc` generates html documentation;

`clean-all` deletes all produced files (including documentation);

`clean` deletes all produced files (excluding documentation);

`clean-doc` deletes documentation files;

`install` copies library files to OCaml directory;

`depend` generates dependency files.

If Cafesterol is not installed, one should call make with the `library`, `cma`, `cmxa`, and `html-doc` targets instead of `all` to get an almost-complete compilation.

[2] http://checkstyle.sourceforge.net/

Chapter 3

Using Cadmium

3.1 Using Cadmium from the command-line

Cadmium can be used from the command-line by issuing the following command:

```
java -jar ocamlrun.jar <cadmium-options> <bytecode-file> <program-arguments>
```

Cadmium recognizes the following command-line switches:

`--gui`
show GUI dialog to set parameters

`--help`
show a message with all supported command-line switches

`--version`
show program version and exit

`--disassemble`
show disassembled program before execution

`--init-stack-size=<integer default:65536>`
set initial stack size, integer should be greater than 512

`--max-stack-size=<integer default:65536>`
set maximum stack size, integer should be greater than or equal to initial stack size

`--backtrace=<on|off default:off>`
whether exception backtrace should be printed

`--stop-jvm=<on|off default:off>`
whether program end should stop JVM

`--providers=<comma-separated-list>`
list elements are fully qualified class names to be used as providers (primitive providers are discussed in chapter 4)

`--compile-dispatchers=<on|off default:on>`
whether primitive dispatchers should be compiled (faster on some JVM implementations)

`--use-awt=<on|off default:off>`
whether Graphics library should use AWT (otherwise, uses Swing)

`--use-javax-sound=<on|off default:off>`
whether Graphics library should use javax.sound (otherwise, uses system mono-tone beep)

`--use-jdbm=<on|off default:off>`
whether Dbm library should use jdbm[1] (otherwise, uses classical Java collections)

`--os=<Unix|Cygwin|Win32|MacOS|Cadmium default:Unix>`
how underlying OS should be identified (as returned by `Sys.os_type`)

`--unix-emulation=<on|off default:off>`
whether unix primitives should be emulated by program execution (when emulated, a try is made to execute a command-line program – *e. g.* `/bin/ln` to create a link);

`--embedded=<on|off default:off>`
whether embedded mode[2] should be activated;

`--embedded-base=<fully-qualified class name>`
base class used by embedded mode when trying to load resources.

3.2 Using Cadmium inside a Java program

Given an `ocamlc`-compiled bytecode file, it is possible to run it from a Java program. Code sample 1 shows how to embed Cadmium execution inside a Java program. Advanced use (*e. g.* callbacks, value exchange between OCaml and Java, *etc.*) is detailed in chapter 4. Chapter 5 discuss scripting using the script engine shipped with Cadmium. More information is also available in the Cadmium javadoc files.

3.3 Building a standalone jar file

The directory `src/ant` in the Cadmium source distribution contains a file named `build-cadmium.xml` that provides two ant macros. These macros allow to build a standalone jar file containing *both* the Cadmium interpreter *and* the `ocamlc`-compiled bytecode. The macros differ in that one is used to create a Java application while the other is used to create a Java applet.

3.3.1 Building an application

When a standalone application is created, it can then be run using `java -jar <name-of-created-jar>`. Code sample 2 shows an Ant file whose only target create a file named `my-app.jar` from an OCaml bytecode file named `program`. The file named `params.properties` is a Java property file containing the Cadmium runtime parameters, as specified in the Cadmium

[1] The "jdbm" part of the w3c Jigsaw project can be reached file-by-file at http://dev.w3.org/cvsweb/java/classes/org/w3c/tools/dbm/ or packaged at http://aurora.regenstrief.org/~schadow/dbm-java/.

[2] When embedded mode is activated, if a program is about to open a file at a given path, a try is previously made to load a resource with the same path in the jar file. If this try fails, then the file is opened in a standard way. The embedded mode thus allows to deploy applications accessing files as standalone jar files.

Code sample 1 Embedding Cadmium.

```
File currentDir = new File(".");
File bcFile = new File("bytecode-file");
RandomAccessInputStream bytecode = new RandomAccessInputStream(bcFile);
ByteCodeParameters params =
    new ByteCodeParameters(new String[] {     /* program arguments */
                               "arg0",
                               "arg1" },
                           false,             /* backtrace */
                           true,              /* stop JVM upon exit */
                           System.in,         /* standard input */
                           System.out,        /* standard output */
                           System.err,        /* standard error */
                           false,             /* use awt */
                           false,             /* bare canvas */
                           true,              /* use javax.sound */
                           false,             /* don't use jdbm */
                           "Unix",            /* OS identifier */
                           false,             /* Unix emulation */
                           bcFile.toString(), /* program name */
                           false,             /* embedded mode */
                           null,              /* base of embedded mode */
                           false,             /* show dialog */
                           false,             /* show help */
                           false,             /* show version */
                           false,             /* show disassembled */
                           64 * 1024,         /* initial stack size */
                           64 * 1024,         /* maximum stack size */
                           new String[] {     /* primitive providers */
                               "prov0",
                               "prov1" },
                           true);             /* compile dispatchers */
Interpreter interp = new Interpreter(params, currentDir, bytecode);
interp.execute();
```

javadoc for the `ByteCodeParameters`[3] class; an empty file will make Cadmium use the default settings.

Code sample 2 Creating a standalone application jar.

```
<?xml version="1.0" encoding="iso-8859-1"?>

<project name="test-build-cadmium" default="build">

  <import file="/path/to/build-cadmium.xml"/>

  <target name="build" description="builds application from bytecode">
    <cadmium.make-application dest-file="my-app.jar"
                              program-dir="."
                              program-file="program"
                              program-parameters="params.properties"
                              cadmium-dir="/path/to/cadmium/deploy"
                              cadmium-file="ocamlrun.jar"/>
  </target>

</project>
```

3.3.2 Building an applet

When a standalone applet is created, it can be embedded into an HTML page. However, due to the Java features internally used by Cadmium, it is necessary for the applet to be signed in order to be successfully loaded into a web browser. Code sample 3 shows an Ant file whose only target create a file named `my-app.jar` from an OCaml bytecode file named `applet`.

The OCaml *main* code of an applet should use `register_applet` from either `Cadmium` or `CadmiumObj` in order to register functions that will subsequently be called by the actual Java applet as the `init`, `start`, `stop` and `destroy` methods. More information is available in the ocamldoc files for the `Cadmium` and `CadmiumObj` modules.

[3] `fr.x9c.cadmium.kernel.ByteCodeParameters`

Code sample 3 Creating a standalone applet jar.

```
<?xml version="1.0" encoding="iso-8859-1"?>

<project name="test-build-cadmium" default="build">

  <import file="/path/to/build-cadmium.xml"/>

  <target name="build" description="builds applet from bytecode">
    <cadmium.make-applet dest-file="my-app.jar"
                         applet-dir="."
                         applet-file="applet"
                         applet-parameters="params.properties"
                         cadmium-dir="/path/to/cadmium/deploy"
                         cadmium-file="ocamlrun.jar"/>
  </target>

</project>
```

Chapter 4

Interfacing Java with Cadmium

This chapter is the counterpart of the eighteenth chapter in the Objective Caml manual (title: "Interfacing C with Objective Caml"). The reader is advised to read the chapter of the original Objective Caml manual first. The information contained in the present chapter also applies to Cafesterol-compiled programs. Additional information can be found in the javadoc files.

4.1 Overview and compilation information

4.1.1 Declaring primitives

In both Cadmium-interpreted and Cafesterol-compiled programs, primitives are declared exactly as in standard Objective Caml.

4.1.2 Implementing primitives

All primitives, whatever their arities, are implemented by methods taking n `Value`[1] parameters (where n is the arity of the primitive), preceded by a parameter of type `CodeRunner`[2], and returning a `Value` instance. A `CodeRunner` instance contains all context information that is related to a given program; it is necessary because several Cadmium programs may be running in the same JVM. Additionally, the implementing method should be static and annotated by `@Primitive`[3], in a class annotated by `@PrimitiveProvider`[4]. This is the fully-qualified name of the class annotated by `@PrimitiveProvider` that should be passed to Cadmium as an additional primitive provider to ensure that it will find the methods implementing the primitives.

As an example, the following class provides a method implementing the `caml_ml_input` primitive:

```
@PrimitiveProvider
public class Implementation {
    @Primitive
    public static Value caml_ml_input(CodeRunner ctxt,
                                      Value channel,
                                      Value buffer,
```

[1] `fr.x9c.cadmium.kernel.Value`
[2] `fr.x9c.cadmium.kernel.CodeRunner`
[3] `fr.x9c.cadmium.kernel.Primitive`
[4] `fr.x9c.cadmium.kernel.PrimitiveProvider`

```
                                        Value offset,
                                        Value length) {
        ...
    }
}
```

4.2 The `Value` class

All Objective Caml entities are represented by instances of the `Value` class. Such an instance can contain:

- a *long* (in the Objective Caml terminology) value, that is a Java `int`;
- an offset value (code offset under Cadmium, closure index under Cafesterol), as a Java `int`;
- a `Block`[5] instance.

4.2.1 Integer values

Integer values encode 31-bit signed integers, stored as Java `int`s.

4.2.2 Blocks

Each block includes a header that contains the size of the block as well as the tag of the block. The tag gives information about the block contents. Tags (accessed by the `getTag()` method) have the same meaning as in standard Objective Caml. The contents of a custom block is usually accessed by the `asCustom()` and `setCustom(Object)` methods, the custom value being a bare `java.lang.Object` value.

4.3 Representation of Caml data types

Type representation is the same as in standard Caml except in the following cases:

- an `int32` value is represented by a custom block, but for performance reasons, the `asInt32()` and `setInt32(int)` methods are used (the actual value is also stored as a bare `int` rather than wrapped in an `Object`);
- an `int64` value is represented by a custom block, but for performance reasons, the `asInt64()` and `setInt64(long)` methods are used (the actual value is also stored as a bare `long` rather than wrapped in an `Object`);
- a `nativeint` value is represented by a custom block, but for performance reasons, the `asNativeInt()` and `setNativeInt(int)` methods are used (the actual value is also stored as a bare `int` rather than wrapped in an `Object`);

[5] `fr.x9c.cadmium.kernel.Block`

The Java counterpart of the `Val_int` macro is the `createFromInt(int)` method of the `Value` class. The `Value` class defines the following constants:

- `UNIT` for `()` value;
- `FALSE` for `false` value;
- `TRUE` for `true` value;
- `EMPTY_LIST` for `[]` value;
- `ZERO` for `0` value;
- `ONE` for `1` value;
- `TWO` for `2` value;
- `MINUS_ONE` for `-1` value;
- `MINUS_TWO` for `-2` value.

One should also notice that values between `-128` and `255` (inclusive) are pooled. It is thus safe to compare them using `==` on the Java side.

The following code constructs a list whose head is `h` and tail is `t` (it first constructs the block and then wraps it into a value):

```
Block b = Block.createBlock(Block.TAG_CONS, h, t);
Value v = Value.createFromBlock(b);
```

The Java counterpart of the `hash_variant` function is the `hashVariant` static method[6].

4.4 Operations on values

Beside the methods listed in this section, `fr.x9c.cadmium.support.Helper` contains some methods easing the process of converting values between Java and Objective Caml.
Additionally, the `fr.x9c.cadmium.support.values` package provides classes that can be used to represent OCaml values; these classes implements the `ToValue` interface which means that they provide a `toValue()` method returning a `Value` instance.

4.4.1 Kind tests

The `Value` class provides the following methods:

- `isLong()` is true if the value is an integer;
- `isCodeOffset()` is true if the value is an offset;
- `isBlock()` is true if the value is a holding a block.

[6] Located in the `fr.x9c.cadmium.primitives.stdlib.Hash` class.

4.4.2 Operations on values

The `Value` class provides the following methods:

- `asLong()` returns the integer value;
- `asCodeOffset()` returns the offset value;
- `asBlock()` returns the underlying block;
- `getRawValue()` returns the long value (either integer or code offset value), with no conversion;
- `createFromLong(int)` builds a value from an integer;
- `createFromCodeOffset(int)` builds an offset from an integer;
- `createFromBlock(Block)` builds a value holding a block;
- `createFromRawValue(int)` builds a long value (either integer or code offset value), with no conversion.

4.4.3 Accessing blocks

The `Block` class provides the following methods:

- `getTag()` returns the tag of the block;
- `sizeValues()` returns the number of values in the block;
- `get(int)` returns the value at the passed index;
- `set(int, Value)` sets the value at the passed index with the given value;
- `getCode()` returns the code offset of the block;
- `setCode(int)` changes the code offset of the block;
- `sizeBytes()` returns the number of bytes in the block;
- `getByte(int)` returns the byte at the passed index;
- `setByte(int, int)` changes the byte at the passed index;
- `getBytes()` returns the bytes as an array of bytes;
- `asString()` returns the string value of the block;
- `asDouble()` returns the double value of the block;
- `setDouble(double)` changes the double value of the block;
- `sizeDoubles()` returns the number of doubles in the block;
- `getDouble(int)` returns the double at the passed index;
- `setDouble(int, double)` changes the double at the passed index with the given value;

- `asCustom()` returns the custom value of the block;
- `asInt32()` returns the `int32` value of the block;
- `asInt64()` returns the `int64` value of the block;
- `asNativeInt()` returns the `nativeint` value of the block.

4.4.4 Allocating blocks

Blocks can be created using methods from the `Block` class whose names start with `create` (detailed information is given in the javadoc files):

- `createAtom(int)` creates an atom (empty block);
- `createBlock(int, ...)` creates block with passed tag and values;
- `createCustom(int, Custom.Operations)` creates block of passed size for a given custom type (custom value should subsequently be set using `setCustom(Object)`);
- `createDouble(double)` creates a double block;
- `createDoubleArray(double[])` creates a double array block with given values;
- `createDoubleArray(int)` creates a double array block of passed length;
- `createString(byte[])` creates a string block;
- `createString(int)` creates a string block of passed length;
- `createString(int, byte[])` creates a string block;
- `createString(int, String)` creates a string block;
- `createString(String)` creates a string block;
- `createStringArray(String[])` creates a string array.

4.4.5 Raising exceptions

Cadmium/Cafesterol programs can throw three kinds of exceptions:

- `CadmiumException`[7] is raised by the runtime when a *high-level* error occurs (*e. g.* invalid bytecode file, invalid parameter, *etc.*);
- `Fatal.Exception`[8] is raised when an unsupported feature is encountered (*e. g.* 64-bit features) or an unrecoverable error occurs (*e. g.* stack underflow);
- `Fail.Exception`[9] is the Java counterpart of all exceptions raised by OCaml code. The `Fail` class provides methods for raising exceptions, including the predefined ones.

[7] `fr.x9c.cadmium.kernel.CadmiumException`
[8] `fr.x9c.cadmium.kernel.Fatal.Exception`
[9] `fr.x9c.cadmium.kernel.Fail.Exception`

4.5 Living in harmony with the garbage collector

There is nothing to do to live in harmony with the garbage collector as Cadmium/Cafesterol relies upon the garbage collector of the hosting JVM.

4.6 Advanced topic: callbacks from Java to OCaml

Callbacks are registered on the Caml side using functions from the `Callback` module. Plain values, functions, as well as exceptions are registered exactly as in standard Objective Caml.

If the Caml program has been executed by an `Interpreter`[10] a callback can be executed using its `execute(String, Value...)` method, where the string is the callback identifier and the values are the parameters to the callback function. If the callback throws an exception, a `CadmiumException` is raised whose cause is the `Fail.Exception` instance raised by the callback. It is the programmer's responsibility to call `execute` with the correct number of values, as well as to ensure that these values have correct types.

4.7 Advanced topic: custom blocks

When creating a custom block, `Block.createCustom` is passed a `Custom.Operations` instance. This instance defines the implementation for the custom type. When implementing a new custom type, one can either implement the `Custom.Operations` interface or extend the `Custom.AbstractOperations` class. The following methods define a custom type:

- `getIdentifier()` should return the custom type identifier (should follow the Objective Caml conventions);
- `finalize(Value)` is called when a value is garbage collected;
- `isComparable()` returns whether values can be compared;
- `compare(Value, Value, AtomicBoolean)` compares two values, the atomic boolean being set if values are unordered;
- `isHashable()` returns whether values can be hashed;
- `hash(Value)` hashes the passed value;
- `isSerializable()` returns whether serialization is supported;
- `serialize(DataOutput, Value)` writes the value to the passed data stream;
- `deserialize(DataInput, Block)` reads the value from the passed data stream.

[10] `fr.x9c.cadmium.kernel.Interpreter`

Chapter 5

Scripting with Cadmium

Cadmium provides an implementation of the Java[1] JSR223 Scripting API[2] (this API is defined by the `javax.script` package) for the Objective Caml language. It thus provides a script engine that allows execution of Objective Caml code snippets in Java applications. It does so by embedding a modified OCaml toplevel. The engine also supports compilation through Cafesterol.

5.1 Script engines

The script engine comes in two flavors:

- the all-in-one `ocamlrun-scripting-standalone.jar` jar file is the fusion of all the needed jar files. It also contains all the required Objective Caml files. This vesion does not need any additional file, and can even be run on a system where Objective Caml is not installed;
- the `ocamlrun-scripting.jar` jar file contains only the elements specific to scripting, it depends uopn `ocamlrun.jar` as well as upon the jar files produced by the installation of Cafesterol[3] (these files correspond to the Objective Caml libraries). To compile scripts two additional files are needed: `baristaLibrary.jar` (from the Barista project[4]) and `ocamljava.jar` (ocamljava-compiled version of ocamljava). It is also necessary to either have Objective Caml installed in its standard location, or to set the `OCAMLLIB` (alternatively `CAMLLIB`) environment variable to the install directory of Objective Caml.

5.2 Evaluating scripts

Code sample 4 shows how to use the script engine from Java. First, a script engine manager is created. Then, it is used to retrieve the script engine that can run `ml` files. Finally, a script is executed. It is also possible to compile the script before execution, as shown in code sample 5 (as script compilation is very slow, one is advised to use it only for scripts whose execution is long). It is also possible to run scripts using `eval` variants that take a context and/or a list of bindings. In this case, one is advised to use instances of `OCamlContext`, and `OCamlBindings` (*cf.* the javadoc-generated documentation).

[1] The official website for the Java Technology can be reached at `http://java.sun.com`.
[2] `http://www.jcp.org/en/jsr/detail?id=223`
[3] `http://cafesterol.x9c.fr`
[4] `http://barista.x9c.fr`

Code sample 4 Running an Objective Caml script.

```
ScriptEngineManager manager = new ScriptEngineManager();
ScriptEngine engine = manager.getEngineByExtension("ml");
engine.eval("print_endline \"printing from ocaml\"");
```

Code sample 5 Compiling (and then running) an Objective Caml script.

```
ScriptEngineManager manager = new ScriptEngineManager();
ScriptEngine engine = manager.getEngineByExtension("ml");
CompiledScript script =
  ((Compilable) engine).compile("print_endline \"printing from ocaml\"");
script.eval();
```

When using bindings, the developer should be aware of the *magic* that occurs when translating the Java values into Objective Caml values. If bindings are `Value` or `Block`[5] instances, they are not translated. Otherwise, the mapping given by table 5.1 is applied (mapping for primitives is given for readability, as the binding mechanism implies that primitive values are wrapped). If the value has a type that do not appear in the table, it is encapsulated into an Objective Caml custom type whose identifier is *cadmium_java_object* (it is thus a `Cadmium.java_object` instance - *cf.* the ocamldoc-generated documentation). The programmer is in charge of using bindings with their correct types on the script side, as no check is done by the framework.
One is advised to check both the `Helper`[6] class, and the `fr.x9c.cadmium.support.values` package for easy creation of OCaml value from the Java side.

The return value of `eval(-)` methods is `java.lang.Object` to match the API specification but returned values are actually instances of the class `Value`. It is the responsibility of the programmer to know the Objective Caml type of the returned value and to interpret it accordingly using the facilities provided by the Cadmium library. In brief, no *magic* occurs on returned values.

Code sample 6 shows the execution of a script using a bound value. To access a bound value, it is necessary to use the Objective Caml `Cadmium` module (or `CadmiumObj` module). Its enclosing library is loaded using the `loadLibrary(-)` method, while the `addLibraryPath(-)` method allows to add a path to the library path (using the same convention as the Objective Caml compilers and toplevel).

Code sample 7 shows how to register a function on the Objective Caml side, and how to invoke it from Java. The Objective Caml script executed by the engine only registers a function with name "say". After script evaluation, it is possible to call this function from Java. As function invocation is not among the facilities provided by a simple scripting engine, one has to cast the engine instance to `Invocable`. In this example, the value returned by `invokeFunction` is ignored; in general, such a value follows the same behaviour as ones returned by `eval(-)` methods.

[5] These classes are used by the Cadmium runtime to represent Objective Caml values.
[6] `fr.x9c.cadmium.support.Helper`

Java type	OCaml mapped type
boolean	bool
java.lang.Boolean	bool
byte	int
java.lang.Byte	int
char	int
java.lang.Character	int
double	float
java.lang.Double	float
float	float
java.lang.Float	float
int	int32
java.lang.Integer	int32
long	int64
java.lang.Long	int64
short	int
java.lang.Short	int
java.lang.String	string

Table 5.1: Mapping of Java *primitive* types.

5.3 Limitations

The following methods of `ScriptEngine` are not supported (they always throw an exception):

- `invokeMethod(-)`;
- `getInterface(-)` (both forms).

In addition, `invokeFunction(-)` can only be used to call Objective Caml functions that have been registered using the Objective Caml callback mechanism. One should also notice that the parameters to the function are transformed in the same way bindings values are. As for bindings, it is the programmer responsibility to ensure that passed values have the good types, and also that the number of passed values matches the function arity.

Code sample 6 Running an Objective Caml script with bindings.

```
ScriptEngineManager manager = new ScriptEngineManager();
ScriptEngine engine = manager.getEngineByExtension("ml");
((OCamlScriptEngine) engine).addLibraryPath("+cadmium");
((OCamlScriptEngine) engine).loadLibrary("cadmiumLibrary");
engine.getBindings(ScriptContext.ENGINE_SCOPE).put("n", 5);
String script =
  "let n = Int32.to_int (Cadmium.get_binding \"n\") in\n" +
  "let a = Array.init n (fun i -> i * i) in\n" +
  "let sum = Array.fold_left (+) 0 a in\n" +
  "Printf.printf \"sum([0;%d[) = %d\\n\" n sum";
engine.eval(script);
```

Code sample 7 Running an Objective Caml function registered as callback.

```
ScriptEngineManager manager = new ScriptEngineManager();
ScriptEngine engine = manager.getEngineByExtension("ml");
String script =
  "Callback.register \"say\" (fun s -> Printf.printf \"OCaml says '%s'\" s)"
engine.eval(script);
((Invocable) engine).invokeFunction("say", "hello");
```

Chapter 6

Programming servlets with Cadmium

Since version 1.1, Cadmium includes bindings to servlets. The servlet framework defined by Sun allows the programming of web applications that are subsequently run using a servlet container. Before using the Cadmium bindings, one is advised to check the Sun documentation to acquire some knowledge about servlets themselves. The official source for the servlet technology is http://java.sun.com/products/servlet/.

6.1 Servlet source and descriptor

The behaviour of a servlet is defined by registering the callbacks handling the various kind of requests it should handle. Code sample 8 shows a typical OCaml program defining a servlet that handles only *get* requests. The `web.xml` descriptor for this application is given by code sample 9 (the replacement value for `fully.qualified.servletClass` is discussed below). The servlet is mapped to `/*`, which explains why the `handle_get` function matches over the path info in order to know the name of the requested service.

To use servlet listeners, it is sufficient to use the functions from `Servlet.Listeners` to register OCaml functions, and to add the elements shown by code sample 10 to the `web.xml` file.

The are two kinds of servlets that can be created: bytecode ones (compiled using `ocamlc`), and native ones (compiled using `ocamljava`). The following sections describe the building process for both cases. When the build process is over, a `war` file has been created that can then be deployed to a servlet container for execution.

6.2 Building a bytecode servlet

Code sample 11 shows a Makefile that creates a bytecode file named `servlet`. Once this file has been created, executing the `build.xml` file shown by code sample 12 allows to produce a web application named `bytecode.war` that can be deployed to a servlet container. The `cadmium.make-servlet` Ant macro is defined in the `src/ant/build-servlet.xml` file. The `web.xml` file is similar to the one reproduced above except that `fully.qualified.servletClass` should be replaced by `fr.x9c.cadmium.support.servlet.ByteCodeServlet`. The file named `parameters.properties` is a Java property file containing the Cadmium runtime parameters,

Code sample 8 source.ml.

```
open CadmiumServlet

let handle_get srv req resp =
  let out = Response.get_output_stream resp in
  match Request.get_path_info req with
  | "/Service1" ->
        ...
  | "/Service2" ->
        ...
  | _ ->
      output_string out "<html><head><title>ERROR</title></head><body>";
      output_string out "<b>UNHANDLED REQUEST</b>";
      output_string out "</body></html>"

let () =
  Servlet.register { Servlet.destroy = ignore;
                     Servlet.init = ignore;
                     Servlet.info = "my first servlet";
                     Servlet.do_delete = None;
                     Servlet.do_get = Some handle_get;
                     Servlet.do_head = None;
                     Servlet.do_options = None;
                     Servlet.do_post = None;
                     Servlet.do_put = None;
                     Servlet.do_trace = None;
                     Servlet.get_last_modified = None; }
```

Code sample 9 web.xml.

```
<?xml version="1.0" encoding="iso-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                             http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5">

  <description>Servlet test</description>
  <display-name>Servlet test</display-name>

  <servlet>
    <servlet-name>Servlet</servlet-name>
    <servlet-class>fully.qualified.servletClass</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Servlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>

</web-app>
```

Code sample 10 Defining listeners in web.xml.

```
<listener>
  <listener-class>fr.x9c.cadmium.support.servlet.SessionActivationListener</listener-class>
</listener>

<listener>
  <listener-class>fr.x9c.cadmium.support.servlet.SessionAttributeListener</listener-class>
</listener>

<listener>
  <listener-class>fr.x9c.cadmium.support.servlet.SessionBindingListener</listener-class>
</listener>

<listener>
  <listener-class>fr.x9c.cadmium.support.servlet.SessionListener</listener-class>
</listener>
```

as specified in the Cadmium javadoc for the `ByteCodeParameters`[1] class; an empty file will make Cadmium use the default settings.

Code sample 11 Makefile (bytecode version).

```
compile:
        ocamlc -c -I +cadmium source.ml

link:
        ocamlc -o servlet -I +cadmium cadmiumLibrary.cma cadmiumServletLibrary.cma \
          source.cmo
```

6.3 Building a native servlet

Code sample 13 shows a Makefile that creates a `native.war` file that is a web application that is ready to be deployed to a servlet container. The `web.xml` file is similar to the one reproduced above except that `fully.qualified.servletClass` should be replaced by `testserv.cafesterolServlet`, as `testsrv` is the chosen package name, and `cafesterolServlet` is the name of the class generated when the `-servlet` command-line switch is passed to `ocamljava`.

[1] `fr.x9c.cadmium.kernel.ByteCodeParameters`

Code sample 12 Ant file (bytecode version).

```
<?xml version="1.0" encoding="iso-8859-1"?>

<project name="test-servlet" default="test">

  <import file="/path/to/build-servlet.xml"/>

  <target name="test" description="war file generation">
    <cadmium.make-servlet dest-file="bytecode.war"
                          servlet-dir="."
                          servlet-file="servlet"
                          servlet-parameters="parameters.properties"
                          servlet-webxml="web-bytecode.xml"
                          cadmium-dir="/path/to/ocamlrun.jar/"
                          cadmium-file="ocamlrun.jar"
                          cadmiumservlet-dir="/path/to/ocamlrun-servlet.jar/"
                          cadmiumservlet-file="ocamlrun-servlet.jar"/>
  </target>

</project>
```

Code sample 13 Makefile (native version).

```
OPTIONS=-java-package testserv \
        -classpath /path/to/ocamlrun-servlet.jar \
        -I +cadmium \
        -provider fr.x9c.cadmium.primitives.cadmiumservlet.Servlets

compile:
        ocamljava $(OPTIONS) -c -I +cadmium source.ml

link:
        ocamljava $(OPTIONS) -o native.war -standalone \
          -additional-jar /path/to/ocamlrun-servlet.jar \
          -servlet web.xml cadmiumLibrary.cmja cadmiumServletLibrary.cmja source.cmj
```

Chapter 7

Programming GUI with Cadmium

Since version 1.1, Cadmium includes bindings to the SwiXml library. SwiXml is a powerful and very convenient library that allows to render a GUI from an XML description. Although it is not the only Java library designed for this purpose, SwiXml is so handy that is appears a good choice. The home page of SwiXml is http://www.swixml.org.

7.1 Building a bytecode GUI

Code sample 14 shows how to invoke a SwiXml-based bytecode program from command-line, where:

`<ocamlrun.jar>` designates the full path to the Cadmium jar file;

`<ocamlrun-swixml.jar>` designates the full path to the SwiXml bindings jar file;

`<swixml.jar>` designates the full path to the swixml jar file available from http://www.swixml.org;

`<jdom.jar>` designates the full path to a JDOM implementation that is compatible with SwiXml (to ensures that, the best is to use the JDOM jar included in the SwiXml distribution);

`<bytecode>` designates the full path to the program (bytecode file) to execute;

`<arguments>` designates the arguments to the program.

Code sample 14 Running a SwiXml-based compiled program.

```
java -cp <ocamlrun.jar>:<ocamlrun-swixml.jar>:<swixml.jar>:<jdom.jar>
    fr.x9c.cadmium.Main --providers=fr.x9c.cadmium.primitives.cadmiumswixml.SwiXml
    <bytecode> <arguments>
```

Code sample 15 shows how to use SwiXml bindings in an embedded way, where:

`"."` designates the path to be used by the interpreted program as the starting working directory;

`"bytecode-file"` designates the path of the bytecode file to be interpreted;

`"arg0"`, `"arg1"` and so on designate the arguments to the interpreted program;

`"prov0"`, `"prov1"` and so on designate the additional primitive providers besides the SwiXml one.

Code sample 15 Embedding a SwiXml-based program.

```
File currentDir = new File(".");
File bcFile = new File("bytecode-file");
RandomAccessInputStream bytecode = new RandomAccessInputStream(bcFile);
ByteCodeParameters params =
    new ByteCodeParameters(new String[] {     /* program arguments */
                               "arg0",
                               "arg1" },
                           false,             /* backtrace */
                           true,              /* stop JVM upon exit */
                           System.in,         /* standard input */
                           System.out,        /* standard output */
                           System.err,        /* standard error */
                           false,             /* use awt */
                           false,             /* bare canvas */
                           true,              /* use javax.sound */
                           false,             /* don't use jdbm */
                           "Unix",            /* OS identifier */
                           false,             /* Unix emulation */
                           bcFile.toString(), /* program name */
                           false,             /* embedded mode */
                           null,              /* base of embedded mode */
                           false,             /* show dialog */
                           false,             /* show help */
                           false,             /* show version */
                           false,             /* show disassembled */
                           64 * 1024,         /* initial stack size */
                           64 * 1024,         /* maximum stack size */
                           new String[] {     /* primitive providers */
                               "fr.x9c.cadmium.primitives.cadmiumswixml.SwiXml",
                               "prov0",
                               "prov1" },
                           true);             /* compile dispatchers */
Interpreter interp = new Interpreter(params, currentDir, bytecode);
interp.execute();
```

7.2 Building a native GUI

It is possible to use `ocamljava` to compile a program using SwiXml. Code sample 16 shows how to compile such a program. The resulting Java jar file can be run by `java -jar prog.jar`.

Code sample 16 Compiling a SwiXml-based program with Cafesterol.

```
ocamljava -standalone
    -I +cadmium cadmiumLibrary.cmja cadmiumSwiXmlLibrary.cmja
    -additional-jar <swixml.jar>
    -additional-jar <jdom.jar>
    -o prog.jar source.ml
```

7.3 Example

The `source.xml` and `source.ml` sources, reproduced as code samples 17 and 18, demonstrate how to use SwiXml.

First, reflection (through `Cadmium` module) is used to retrieve the information needed to build the action listeners. The `make_listener` function is defined such that if it is passed a `unit -> unit` function , it returns a `java_object` instance that is a valid action listener[1]. The `show` function is then built upon it to construct a listener that displays the passed string in a message dialog.

We also define an auxiliary `register_listeners` function that takes a list of *(component-identifier, listener)* couples and performs the registration of each listener with its associated component.

The actual main part of the program consists in 7 steps:

1. SwiXml engine construction;
2. rendering of the UI from the `source.xml` file;
3. definition of `quit` that just terminates the program;
4. definition of `connect` that only sets the label of the `changeButton`;
5. registration of listeners with components using their identifiers as defined in the XML file;
6. exposition of the UI;
7. little *wait* trick to ensure that the program does not terminate at once.

For a comprehensive use of SwiXml, it is necessary to grasp the concepts behind both SwiXml and Java Swing. To this intent, one should refer to the SwiXml and Java Swing documentations.

One may also notice that the example is written using `Cadmium` and `SwiXml` modules while the exact same result could be achieved using `CadmiumObj` and `SwiXmlObj` modules that provide the same functionalities as their conterparts, in an object-oriented fashion.

More, although reflection can be used as seen in the example to access to Java elements, users are strongly encouraged to use Nickel[2] in order to access the full power of Java Swing with

[1] This is achieved by a proxy created by the `Cadmium.register` function.
[2] Nickel, the Objective Caml / Java bridge generator for Cadmium, is available at `http://nickel.x9c.fr`.

an increased ease. To be convinced of the added value of Nickel, one has to consider that its use would roughly reduce the example program to the code below the `(* main *)` comment, at the expense of OCaml/Java bindings generation.

Code sample 17 source.xml.

```
<?xml version="1.0" encoding="iso-8859-1"?>

<frame title="SwiXml test" size="640,400"
       DefaultCloseOperation="JFrame.EXIT_ON_CLOSE">
  <menubar>
    <menu Text="Cadmium/SwiXml test">
      <menuitem Text="What is Cadmium ?" id="cadmiumMenu"/>
      <menuitem Text="What is SwiXml ?" id="swixmlMenu"/>
      <menuitem Text="What is Nickel ?" id="nickelMenu"/>
      <separator/>
      <menuitem Text="What is SwiXml ?" id="cadmiumswixmlMenu"/>
    </menu>
  </menubar>

  <panel constraints="BorderLayout.NORTH">
    <label Text="login"/>
    <textfield Columns="10" Text="login"/>
    <label Text="password"/>
    <passwordfield Columns="10" Text="pwd"/>
    <button id="changeButton" Text="connect" constraints="BorderLayout.SOUTH"/>
  </panel>
  <scrollpane constraints="BorderLayout.CENTER">
   <textarea Autoscrolls="true">Basic interactive test for SwiXml
UI is described in XML
Listeners are written in OCaml and are passed to Java using Cadmium proxies
</textarea>
  </scrollpane>
  <panel layout="BorderLayout" constraints="BorderLayout.SOUTH">
    <glue constraints="BorderLayout.CENTER"/>
    <button id="quitButton" constraints="BorderLayout.EAST" Text="Quit"
            TooltipText="click to quit"/>
  </panel>
</frame>
```

Code sample 18 source.ml.

```
open Cadmium
open CadmiumSwiXml

let lock = Object.make ()

(* frames *)
let frame_class = Class.for_name "javax.swing.JFrame"
let show_method = Class.get_method frame_class "show" []

(* action listeners and events *)
let action_listener_class = Class.for_name "java.awt.event.ActionListener"
let action_event_class = Class.for_name "java.awt.event.ActionEvent"
let action_performed_method = Class.get_method action_listener_class "actionPerformed"
    [Class action_event_class]
let make_listener f =
  let g _ = f (); Object_val (get_null ()) in
  register [action_listener_class] [(action_performed_method, g)]

(* buttons *)
let button_class = Class.for_name "javax.swing.JButton"
let set_label_method = Class.get_method button_class "setLabel" [String]

(* option panes *)
let option_pane_class = Class.for_name "javax.swing.JOptionPane"
let show_message_dialog_method = Class.get_method option_pane_class "showMessageDialog"
    [(Class (Class.for_name "java.awt.Component"));
     (Class (Class.for_name "java.lang.Object"))]
let show msg =
  make_listener (fun () -> Method.invoke show_message_dialog_method (get_null ())
      [Object_val (get_null ()); (String_val msg)])

(* main *)
let register_listeners e =
  List.iter (fun (id, lst) ->
    let res = set_action_listener_by_name e id lst in
    if not res then raise Not_found)
let () =
  let engine = make_engine () in
  let frame = render_from_file engine "source.xml" in
  let quit () = Object.notify lock in
  let connect () = Method.invoke set_label_method (find engine "changeButton")
      [String_val "logged"] in
  register_listeners engine
    [ ("quitButton", (make_listener quit));
      ("changeButton", (make_listener connect));
      ("cadmiumMenu", (show "OCaml Interpreter and Runtime for Java\ncadmium.x9c.fr"));
      ("swixmlMenu", (show "Rendering of XML-defined GUI\nwww.swixml.org"));
      ("nickelMenu", (show "OCaml/Java Bridge Generator\nnickel.x9c.fr"));
      ("cadmiumswixmlMenu", (show "Cadmium bindings for SwiXml\ncadmium.x9c.fr"))];
  ignore (Method.invoke show_method frame []);
  Object.wait lock;
  exit 0
```