

OCaml-Java Cheat Sheet

<http://www.ocamljava.org>

Xavier Clerc, July 2014

Tools

| | |
|-------------------------|---|
| <code>ocaml</code> | classical toplevel |
| <code>ocamlbuild</code> | compilation manager (ocamljava-aware) |
| <code>ocamlc</code> | compiler producing OCaml bytecode |
| <code>ocamldebug</code> | debugger for ocamlc-compiled programs |
| <code>ocamldep</code> | dependency analyzer |
| <code>ocamldoc</code> | documentation generator (ocamljava-aware) |
| <code>ocamlj</code> | toplevel using Java bytecode |
| <code>ocamljar</code> | post-compilation optimizer |
| <code>ocamljava</code> | compiler producing Java bytecode |
| <code>ocamlrun</code> | interpreter for ocamlc-compiled programs |
| <code>ocamltop</code> | classical toplevel, as a windowed application |
| <code>ocamlwrap</code> | generator of Java interfaces to OCaml code |

File extensions

| | <code>ocamlc</code> | <code>ocamltop</code> | <code>ocamljava</code> |
|-------------------------------|---------------------|-----------------------|------------------------|
| <i>interface:</i> source | <code>.mli</code> | <code>.mli</code> | <code>.mli</code> |
| compiled | <code>.cmi</code> | <code>.cmi</code> | <code>.cmi</code> |
| <i>implementation:</i> source | <code>.ml</code> | <code>.ml</code> | <code>.ml</code> |
| compiled | <code>.cmo</code> | <code>.cmx</code> | <code>.cmj</code> |
| object | <code>-</code> | <code>.o</code> | <code>.jo</code> |
| <i>library:</i> compiled | <code>.cma</code> | <code>.cmxa</code> | <code>.cmja</code> |
| object | <code>-</code> | <code>.a</code> | <code>.ja</code> |
| <i>executable</i> | <code>.out</code> | <code>.out</code> | <code>.jar</code> |
| <i>plugin</i> | <code>-</code> | <code>.cmxs</code> | <code>.cmjs</code> |

Compilation and link

General

compile an interface: `ocamljava -c m.mli`

compile an implementation: `ocamljava -c m.ml`

produce a library: `ocamljava -a -o l.cmja m.cmj ...`

additional command-line switches:

| | |
|-------------------------------|----------------------------------|
| <code>-classpath c</code> | set classpath |
| <code>-cp c</code> | add to classpath |
| <code>-java-extensions</code> | activate typer extensions |
| <code>-java-package p</code> | set package for compiled modules |

Applications

link as executable: `ocamljava -o e.jar m.cmj ...`

Applets

link as applet: `ocamljava -applet k -o a.jar m.cmj ...`
where `k` is the kind of applet (`awt`, `swing`, or `graphics`)

Servlets

compile as servlet: `ocamljava -servlet k -c m.ml`
where `k` is the kind of servlet (`http`, or `generic`)

link as servlet: `ocamljava -war f -o s.war m.cmj ...`
where `f` is the file to be used as the webapp descriptor

ocamlbuild (extended)

recognizes the `ocamljava`-specific extensions and tags for the additional command-line switches, plus:

| | |
|-----------------------------|----------------------------|
| <code>use_javalib</code> | for the Java library |
| <code>use_concurrent</code> | for the concurrent library |

Post-compilation optimization

A compiled `jar` file can be optimized through

`ocamljar [options] in.jar out.jar`

possible options include:

| | |
|-----------------------------------|--|
| <code>-no-backtrace v</code> | to set backtrace support |
| <code>-no-debug v</code> | to set debug support |
| <code>-no-dynlink v</code> | to set dynlink support |
| <code>-no-runtime-lock v</code> | to set runtime lock use |
| <code>-no-signals v</code> | to set signals support |
| <code>-no-unused-globals v</code> | to set removal of unused globals |
| <code>-unsafe v</code> | to set use of <i>unsafe</i> data containers |
| <code>-war</code> | if passed file is a <code>war</code> archive |

where `v` can be either `false` or `true`

Wrappers generation

Wrappers for elements of a module can be generated by:

```
ocamljava -c m.mli
ocamljava -c m.ml
ocamljava -o p.jar m.cmj
ocamlwrap m.cmi
```

resulting in a file named `MWrapper.java` allowing to access the OCaml elements

Typer extension

Mapping of types

| Java type | OCaml type | note |
|-------------------------|---------------------------------------|------|
| <code>boolean</code> | <code>bool</code> | |
| <code>byte</code> | <code>int</code> | |
| <code>char</code> | <code>int</code> | |
| <code>double</code> | <code>float</code> | |
| <code>float</code> | <code>float</code> | |
| <code>int</code> | <code>int32</code> | |
| <code>long</code> | <code>int64</code> | |
| <code>short</code> | <code>int</code> | |
| <code>pack.Class</code> | <code>pack'Class java_instance</code> | (1) |
| | <code>pack'Class java_extends</code> | (2) |

(1) used to designate exactly an instance of `pack.Class`

(2) used to designate an instance of `pack.Class` or any subtype

Instance creation

`let obj = Java.make "pack.Class(sign)" params`

Method calls

`Java.call "pack.Class.meth(sign)" inst params`

`Java.call "pack.Class.stat(sign)" params`

Field accesses

`let val = Java.get "pack.Class.field:type" inst`

`Java.set "pack.Class.field:type" inst val`

`let val = Java.get "pack.Class.stat:type" ()`

`Java.set "pack.Class.stat:type" val`

Type checks

`let cls = Java.get_class inst`

`let bool_val = Java.instanceof "pack.Class" inst`

`let inst' = Java.cast "pack.Class" inst`

Sugar

Any type in a signature can be replaced with an underscore ("`_`") as long as there is no ambiguity; a dash ("`-`") can be used instead of a whole signature as long as there is no ambiguity

`open Package'pack` is equivalent to `import pack.*;`, allowing to use simple class names instead of fully-qualified class names

Proxies

```
Java.proxy "pack.Interface" (object
  method m1 ... = ...
  method m2 ... = ...
end)
```

builds an instance implementing the interface declared as:

```
package pack;
public interface Interface {
  ... m1(...);
  ... m2(...);
}
```

Exceptions

`exception Java_exception of java'lang'Exception java_instance`
`exception Java_exception of java'lang'Error java_instance`
are used to respectively represent Java exceptions and error;
both can be caught as regular OCaml exceptions

`Java.throw inst` is used to raise a Java exception; `inst` must be an instance of `java.lang.Throwable`

Main modules of javalib.cmja

| | |
|----------------------------|---|
| <code>Java</code> | basic functions |
| <code>JavaString</code> | <code>String</code> -like interface to Java strings |
| <code>JavaXyzArray</code> | arrays of <code>Xyz</code> values (one for each primitive type plus one for references) |
| <code>JavaArray</code> | generic representation of arrays |
| <code>JavaIOStreams</code> | conversion between Java streams and OCaml channels |
| <code>JavaApplet</code> | type definitions for the various applet kinds |
| <code>JavaServlet</code> | type definitions for the various servlet kinds |