Armed Bear Common Lisp User Manual

Mark Evenson Erik Hülsmann Rudolf Schlatte Alessio Stalla
Ville Voutilainen

Version 1.8.0
October 27, 2020

Contents

0.0.1 Preface to the Second Edition

ABCL 1.1 now contains (A)MOP. We hope you enjoy! –The Mgmt.

0.0.2 Preface to the Third Edition

The implementation now contains a performant and conformant implementation of (A)MOP to the point of inclusion in CLOSER-MOP's test suite.

0.0.3 Preface to the Fourth Edition

ABCL 1.3 now implements an optimized implementation of the `org.armedbear.lisp.LispStack` abstraction thanks to Dmitry Nadezhin which runs on ORCL JVMs from 1.[5-8] conformantly.

0.0.4 Preface to the Fifth Edition

ABCL 1.4 consolidates eighteen months of production bug-fixes, and substantially improves the support for invoking external processes via `SYS:RUN-PROGRAM`.

0.0.5 Preface to the Sixth Edition

With the sixth major release of the implementation, we make the following explicit revision of our compatibility to the underlying JVM. Since we are an open source implementation, we insist on possible open access to the sources from with an JDK may both be built and run upon. This requirement is no longer met by Java 5, so henceforth with the release of ABCL 1.5, we will support JAVA 6, JAVA 7 and JAVA 8 runtimes.

0.0.6 Preface to the Seventh Edition

Long overdue, we turn our Java to 11.

Reflecting the management's best estimates as to implementation most easily available to the potential ABCL 1.6 User, the Seventh release implementation works best with JAVA 8 or JAVA 11 runtimes. Since freely available implementations of jdk6 and jdk7 exist, we still strive to maintain compatibility with the Java 6 and Java 7 runtime environments but those environments are less tested. The User may need to use the facilities of the ABCL-BUILD contrib to recompile the implementation locally in more exotic runtimes (see Section 5.6 page 54).

0.0.7 Preface to the Eighth Edition

Since the implementation now runs comfortably on openjdk6, openjdk7, openjdk8, openjdk11, and openjdk14, we take advantage of the presence of the `java.nio` package introduced in JAVA 5. We have overhauled the implementation to use these abstractions for arrays specialized on commonly used unsigned-byte types, adding two additional keyword arguments useful in their construction to `cl:make-array`.[1].

0.0.8 Preface to the Ninth Edition

With the Ninth Edition of the implementation we now support building and running with the openjdk15. This is intended as the last major release to support the openjdk6, openjdk7, and openjdk8 platforms.

The implementation of the `EXT:JAR-PATHNAME` and `EXT:URL-PATHNAME` subtypes of `cl:PATHNAME` has been fixed to the point that arbitrary references to zipfile archives within zipfile now work for most read-only operations (`CL:PROBE-FILE`, `CL:TRUENAME`, `CL:OPEN`, `CL:LOAD`, `CL:FILE-WRITE-DATE`, `CL:DIRECTORY`, and `CL:MERGE-PATHNAMES`). The previous version relied on the ability for `java.net.URL` to open streams of an archive within an archive, behavior that was silently dropped after Java 5, and consequently hasn't worked on common platforms supported by the Bear in a long time. This restores the feasiblity of accessing fasls from within jar files [2].

[1] See 4.10 on page 49

[2] Examine the ASDF-JAR contrib in section 5.2 on page 52 for a recipe for packaging and accessing such artifacts.

Chapter 1

Introduction

Armed Bear Common Lisp (ABCL) is an implementation of Common Lisp that runs on the Java Virtual Machine (JVM). ABCL compiles Common Lisp to Java byte-code[1], with an efficiency that varies upon the hosting JVM implementation. ABCL supports building and running on the Java 6, Java 7, Java 8, Java 11, Java 13, Java 14 and Java 15 openjdk platformsJVM implementations[2].

Armed Bear provides the following integration methods for interfacing with Java code and libraries:

- Lisp code can create Java objects and call their methods (see Section 3.1, page 13).
- Java code can call Lisp functions and generic functions, either directly (Section 3.2.1, page 15) or via `JSR-223` (Section 3.3, page 17).
- `jinterface-implementation` creates Lisp-side implementations of Java interfaces that can be used as listeners for Swing classes and similar.
- `java:jnew-runtime-class` can inject fully synthetic Java classes–and their objects– into the current JVM process whose behavior is specified via closures expressed in Common Lisp. [3]

ABCL is supported by the Lisp library manager Quicklisp[4] and can run many of the programs and libraries provided therein out-of-the-box.

1.1 Conformance

1.1.1 ANSI Common Lisp

ABCL is currently a (non)-conforming ANSI Common Lisp implementation due to the following known issues:

- The generic function signatures of the `CL:DOCUMENTATION` symbol do not match the specification.
- The `CL:TIME` form does not return a proper `CL:VALUES` environment to its caller.

[1] The class files produced by the compiler have a byte-code version of "49.0".

[2] As of April 2020, the AdoptOpenJDK.net community `https://adoptopenjdk.net/` provides perhaps the easiest installation of unencumbered openjdk implementations

[3] Parts of the current implementation are not fully finished, so the status of some interfaces here should be treated with skepticism if you run into problems.

[4] `http://quicklisp.org/`

- When merging pathnames and the defaults point to a `EXT:JAR-PATHNAME`, we set the `DEVICE` of the result to `:UNSPECIFIC` if the pathname to be be merged does not contain a specified `DEVICE`, does not contain a specified `HOST`, does contain a relative `DIRECTORY`, and we are not running on a MSFT Windows platform.[5]

Somewhat confusingly, this statement of non-conformance in the accompanying user documentation fulfills the requirements that ABCL is a conforming ANSI Common Lisp implementation according to the Common Lisp Hyper-Spec [P+96]. Clarifications to this point are solicited.

ABCL aims to be be a fully conforming ANSI Common Lisp implementation. Any other behavior should be reported as a bug.

1.1.2 Contemporary Common Lisp

In addition to ANSI conformance, ABCL strives to implement features expected of a contemporary Common Lisp, i.e. a Lisp of the post-2005 Renaissance.

The following known problems detract from ABCL being a proper contemporary Common Lisp.

- An incomplete implementation of interactive debugging mechanisms, namely a no-op version of `STEP`[6], the inability to inspect local variables in a given call frame, and the inability to resume a halted computation at an arbitrarily selected call frame.
- Incomplete streams abstraction, in that ABCL needs a suitable abstraction between ANSI and Gray streams with a runtime switch for the beyond conforming behavior[7].
- Incomplete documentation: source code is missing doc-strings from all exported symbols from the `EXTENSIONS`, `SYSTEM`, `JAVA`, `MOP`, and `THREADS` packages. This user manual is currently in draft status.

1.2 License

ABCL is licensed under the terms of the GPL v2 of June 1991 with an added "classpath-exception" clause (see the file `COPYING` in the source distribution[8] for the license, term 13 in the same file for the classpath exception). This license broadly means that you must distribute the sources to ABCL, including any changes you make, together with a program that includes ABCL, but that you are not required to distribute the sources of the whole program. Submitting your changes upstream to the ABCL development team is actively encouraged and very much appreciated, of course.

1.3 Contributors

- Dmitry Nadezhin
- Philipp Marek `Thanks for the markup, and review of the Manual`
- Douglas Miles `Thanks for the whacky IKVM stuff and keeping the flame alive in the dark years.`

[5] The intent of this rather arcane sounding deviation from conformance is so that the result of a merge won't fill in a `DEVICE` with the wrong "default device for the host" in the sense of the fourth paragraph in the CLHS description of MERGE-PATHNAMES (see in [P+96] the paragraph beginning "If the PATHNAME explicitly specifies a host and not a deviceâĂę"). A future version of the implementation may return to conformance by using the `HOST` value to reflect the type explicitly. See 4.2 on page 45 for further information.

[6] Somewhat surprisingly allowed by ANSI

[7] The streams could be optimized to the JVM NIO [**?**] abstractions at great profit for binary byte-level manipulations.

[8] See `http://abcl.org/svn/trunk/tags/1.8.0/COPYING`

- Alan Ruttenberg `Thanks for JSS.`
- Olof-Joachim Frahm
- piso
- and of course *Peter Graves*

Chapter 2

Running ABCL

ABCL is packaged as a single jar file usually named either `abcl.jar` or possibly something like `abcl-1.8.0.jar` if using a versioned package on the local file-system from your system vendor. This jar file can be executed from the command line to obtain a REPL[1], viz:

```
cmd$ java -jar abcl.jar
```

N.b. for the proceeding command to work, the `java` executable needs to be in your path.

To facilitate the use of ABCL in tool chains such as SLIME [sli] (the Superior Lisp Interaction Mode for Emacs), we provide both a Bourne shell script and a DOS batch file. If you or your administrator adjusted the path properly, ABCL may be executed simply as:

```
cmd$ abcl
```

Probably the easiest way of setting up an editing environment using the EMACS editor is to use QUICKLISP and follow the instructions at `http://www.quicklisp.org/beta/#slime`.

2.1 Options

ABCL supports the following command line options:

`--help` displays a help message.

`--noinform` Suppresses the printing of startup information and banner.

`--noinit` suppresses the loading of the `~/.abclrc` startup file.

`--nosystem` suppresses loading the `system.lisp` customization file.

`--eval FORM` evaluates FORM before initializing the REPL.

`--load FILE` loads the file FILE before initializing the REPL.

`--load-system-file FILE` loads the system file FILE before initializing the REPL.

`--batch` evaluates forms specified by arguments and in the initialization file `~/.abclrc`, and then exits without starting a REPL.

All of the command line arguments following the occurrence of `--` are passed unprocessed into a list of strings accessible via the variable `EXT:*COMMAND-LINE-ARGUMENT-LIST*` from within ABCL.

[1] Read-Eval Print Loop, a Lisp command-line

2.2 Initialization

If the ABCL process is started without the `--noinit` flag, it attempts to load a file named `.abclrc` in the user's home directory and then interpret its contents.

The user's home directory is determined by the value of the JVM system property `user.home`. This value may or may not correspond to the value of the `HOME` system environment variable, at the discretion of the JVM implementation that ABCL finds itself hosted upon.

Chapter 3

Interaction with the Hosting JVM

The Armed Bear Common Lisp implementation is hosted on a Java Virtual Machine. This chapter describes the mechanisms by which the implementation interacts with that hosting mechanism.

3.1 Lisp to Java

ABCL offers a number of mechanisms to interact with Java from its Lisp environment. It allows calling both instance and static methods of Java objects, manipulation of instance and static fields on Java objects, and construction of new Java objects.

When calling Java routines, some values will automatically be converted by the FFI[1] from Lisp values to Java values. These conversions typically apply to strings, integers and floats. Other values need to be converted to their Java equivalents by the programmer before calling the Java object method. Java values returned to Lisp are also generally converted back to their Lisp counterparts. Some operators make an exception to this rule and do not perform any conversion; those are the "raw" counterparts of certain FFI functions and are recognizable by their name ending with `-RAW`.

3.1.1 Low-level Java API

This subsection covers the low-level API available after evaluating `(require :java)`. A higher level Java API, developed by Alan Ruttenberg, is available in the `contrib/jss` directory and described later in this document, see Section 5.3 on page 52.

Calling Java Object Methods

There are two ways to call a Java object method in the low-level (basic) API:

- Call a specific method reference (which was previously acquired)
- Dynamic dispatch using the method name and the call-specific arguments provided by finding the best match (see Section 3.1.1).

`JAVA:JMETHOD` is used to acquire a specific method reference. The function takes two or more arguments. The first is a Java class designator (a `JAVA:JAVA-CLASS` object returned by `JAVA:JCLASS` or a string naming a Java class). The second is a string naming the method.

Any arguments beyond the first two should be strings naming Java classes, with one exception as listed in the next paragraph. These classes specify the types of the arguments for the method.

[1] Foreign Function Interface is the term of art for the part of a Lisp implementation which implements calling code written in other languages, typically normalized to the local C compiler calling conventions.

When `JAVA:JMETHOD` is called with three parameters and the last parameter is an integer, the first method by that name and matching number of parameters is returned.

Once a method reference has been acquired, it can be invoked using `JAVA:JCALL`, which takes the method as the first argument. The second argument is the object instance to call the method on, or `NIL` in case of a static method. Any remaining parameters are used as the remaining arguments for the call.

Calling Java object methods: dynamic dispatch

The second way of calling Java object methods is by using dynamic dispatch. In this case `JAVA:JCALL` is used directly without acquiring a method reference first. In this case, the first argument provided to `JAVA:JCALL` is a string naming the method to be called. The second argument is the instance on which the method should be called and any further arguments are used to select the best matching method and dispatch the call.

Dynamic dispatch: Caveats

Dynamic dispatch is performed by using the Java reflection API[2]. Generally the dispatch works fine, but there are corner cases where the API does not correctly reflect all the details involved in calling a Java method. An example is the following Java code:

```
ZipFile jar = new ZipFile("/path/to/some.jar");
Object els = jar.entries();
Method method = els.getClass().getMethod("hasMoreElements");
method.invoke(els);
```

Even though the method `hasMoreElements()` is public in `Enumeration`, the above code fails with

```
java.lang.IllegalAccessException: Class ... can
not access a member of class java.util.zip.ZipFile\$2 with modifiers
"public"
       at sun.reflect.Reflection.ensureMemberAccess(Reflection.java:65)
       at java.lang.reflect.Method.invoke(Method.java:583)
       at ...
```

This is because the method has been overridden by a non-public class and the reflection API, unlike `javac`, is not able to handle such a case.

While code like that is uncommon in Java, it is typical of ABCL's FFI calls. The code above corresponds to the following Lisp code:

```
(let ((jar (jnew "java.util.zip.ZipFile" "/path/to/some.jar")))
  (let ((els (jcall "entries" jar)))
    (jcall "hasMoreElements" els)))
```

except that the dynamic dispatch part is not shown.

To avoid such pitfalls, all Java objects in ABCL carry an extra field representing the "intended class" of the object. That class is used first by `JAVA:JCALL` and similar to resolve methods; the actual class of the object is only tried if the method is not found in the intended class. Of course, the intended class is always a super-class of the actual class – in the worst case, they coincide. The intended class is deduced by the return type of the method that originally returned the Java object; in the case above, the intended class of `ELS` is `java.util.Enumeration` because that is the return type of the `entries` method.

While this strategy is generally effective, there are cases where the intended class becomes too broad to be useful. The typical example is the extraction of an element from a collection, since methods in the collection API erase all types to `Object`. The user can always force a more specific intended class by using the `JAVA:JCOERCE` operator.

[2]The Java reflection API is found in the `java.lang.reflect` package

Calling Java class static methods

Like non-static methods, references to static methods can be acquired by using the `JAVA:JMETHOD` primitive. Static methods are called with `JAVA:JSTATIC` instead of `JAVA:JCALL`.

Like `JAVA:JCALL`, `JAVA:JSTATIC` supports dynamic dispatch by passing the name of the method as a string instead of passing a method reference. The parameters should be values to pass in the function call instead of a specification of classes for each parameter.

Parameter matching for FFI dynamic dispatch

The algorithm used to resolve the best matching method given the name and the arguments' types is the same as described in the Java Language Specification. Any deviation should be reported as a bug.

Instantiating Java objects

JAVA objects can be instantiated (created) from LISP by calling a constructor from the class of the object to be created. The `JAVA:JCONSTRUCTOR` primitive is used to acquire a constructor reference. Its arguments specify the types of arguments of the constructor method the same way as with `JAVA:JMETHOD`.

The obtained constructor is passed as an argument to `JAVA:JNEW`, together with any arguments. `JAVA:JNEW` can also be invoked with a string naming the class as its first argument.

Accessing Java object and class fields

Fields in Java objects can be accessed using the getter and setter functions `JAVA:JFIELD` and `(SETF JAVA:JFIELD)`. Static (class) fields are accessed the same way, but with a class object or string naming a class as first argument.

Like `JAVA:JCALL` and friends, values returned from these accessors carry an intended class around, and values which can be converted to Lisp values will be converted.

3.2 Java to Lisp

This section describes the various ways that one interacts with LISP from JAVA code. In order to access the LISP world from JAVA, one needs to be aware of a few things, the most important ones being listed below:

- All Lisp values are descendants of `LispObject`.
- Lisp symbols are accessible either via static members of the `Symbol` class, or by dynamically introspecting a `Package` object.
- The Lisp dynamic environment may be saved via `LispThread.bindSpecial(Binding)` and restored via `LispThread.resetSpecialBindings(Mark)`.
- Functions can be executed by invoking `LispObject.execute(args [...])`

3.2.1 Calling Lisp from Java

Note: the entire ABCL LISP system implementation in JAVA is resident in the `org.armedbear.lisp` package, but the following code snippets do not show the relevant import statements in the interest of brevity. An example of the import statement would be

```
import org.armedbear.lisp.*;
```

to potentially import all the JVM symbol from the `org.armedbear.lisp` namespace.

There can only ever be a single Lisp interpreter per JVM instance. A reference to this interpreter is obtained by calling the static method `Interpreter.createInstance()`.

```
Interpreter interpreter = Interpreter.createInstance();
```

If this method has already been invoked in the lifetime of the current Java process it will return `null`, so if you are writing JAVA whose life-cycle is a bit out of your control (like in a JAVA servlet), a safer invocation pattern might be:

```
Interpreter interpreter = Interpreter.getInstance();
if (interpreter == null) {
  interpreter = Interpreter.createInstance();
}
```

The Lisp `eval` primitive may simply be passed strings for evaluation:

```
String line = "(load␣\"file.lisp\")";
LispObject result = interpreter.eval(line);
```

Notice that all possible return values from an arbitrary Lisp computation are collapsed into a single return value. Doing useful further computation on the `LispObject` depends on knowing what the result of the computation might be. This usually involves some amount of `instanceof` introspection, and forms a whole topic to itself (see Section 3.2.1, page 17).

Using `eval` involves the Lisp interpreter. Lisp functions may also be directly invoked by Java method calls as follows. One simply locates the package containing the symbol, obtains a reference to the symbol, and then invokes the `execute()` method with the desired parameters.

```
interpreter.eval("(defun␣foo␣(msg)" +
  "(format␣nil␣\"You␣told␣me␣'~A'~%\"␣msg))");
Package pkg = Packages.findPackage("CL-USER");
Symbol foo = pkg.findAccessibleSymbol("FOO");
Function fooFunction = (Function)foo.getSymbolFunction();
JavaObject parameter = new JavaObject("Lisp␣is␣fun!");
LispObject result = fooFunction.execute(parameter);
// How to get the "naked string value"?
System.out.println("The␣result␣was␣" + result.writeToString());
```

If one is calling a function in the CL package, the syntax can become considerably simpler. If we can locate the instance of definition in the ABCL Java source, we can invoke the symbol directly. For instance, to tell if a `LispObject` is (Lisp) `NIL`, we can invoke the CL function `NULL` in the following way:

```
boolean nullp(LispObject object) {
  LispObject result = Primitives.NULL.execute(object);
  if (result == NIL) {
    return false;
  }
  return true;
}
```

Note, that the symbol `nil` is explicitly named in the JAVA namespace as `Symbol.NIL` but is always present in the local namespace in its unadorned form for the convenience of the User.

Multiple Values

After a call to a function that returns Lisp multiple values, the values are associated with the executing `LispThread` until the next call into Lisp. One may access the values object as a list of `LispObject` instances via a call to `getValues()` on that thread reference as evidenced by the following code:

```
org.armedbear.lisp.Package cl = Packages.findPackage("CL");
Symbol valuesSymbol = cl.findAccessibleSymbol("VALUES");
LispObject[] valuesArgs = {
  LispInteger.getInstance(1), LispInteger.getInstance(2)
};
// equivalent to ``(values 1 2)''
LispObject result = valuesSymbol.execute(valuesArgs);
LispObject[] values = LispThread.currentThread().getValues();
for (LispObject value: values) {
  System.out.println("value␣==>␣" + value.printObject());
}
```

Introspecting a LispObject

We present various patterns for introspecting an arbitrary `LispObject` which can hold the result of every Lisp evaluation into semantics that Java can meaningfully deal with.

LispObject as `boolean` If the `LispObject` is to be interpreted as a generalized boolean value, one can use `getBooleanValue()` to convert to Java:

```
LispObject object = Symbol.NIL;
boolean javaValue = object.getBooleanValue();
```

Since in Lisp any value other than `NIL` means "true", Java equality can also be used, which is a bit easier to type and better in terms of information it conveys to the compiler:

```
boolean javaValue = (object != Symbol.NIL);
```

LispObject as a list If `LispObject` is a list, it will have the type `Cons`. One can then use the `copyToArray` method to make things a bit more suitable for Java iteration.

```
LispObject result = interpreter.eval("'(1␣2␣4␣5)");
if (result instanceof Cons) {
  LispObject array[] = ((Cons)result.copyToArray());
  ...
}
```

A more Lispy way to iterate down a list is to use the 'cdr()' access function just as like one would traverse a list in Lisp:;

```
LispObject result = interpreter.eval("'(1␣2␣4␣5)");
while (result != Symbol.NIL) {
  doSomething(result.car());
  result = result.cdr();
}
```

3.3 Java Scripting API (JSR-223)

ABCL can be built with support for JSR-223 [Gro06], which offers a language-agnostic API to invoke other languages from Java. The binary distribution download-able from ABCL's homepage is built with JSR-223 support. If you're building ABCL from source on a pre-1.6 JVM, you need to have a JSR-223 implementation in your classpath (such as Apache Commons BSF 3.x or greater) in order to build ABCL with JSR-223 support; otherwise, this feature will not be built.

This section describes the design decisions behind the ABCL JSR-223 support. It is not a description of what JSR-223 is or a tutorial on how to use it. See http://abcl.org/trac/browser/trunk/abcl/examples/jsr-223 for example usage.

3.3.1 Conversions

In general, ABCL's implementation of the JSR-223 API performs implicit conversion from Java objects to Lisp objects when invoking Lisp from Java, and the opposite when returning values from Java to Lisp. This potentially reduces coupling between user code and ABCL. To avoid such conversions, wrap the relevant objects in `JavaObject` instances.

3.3.2 Implemented JSR-223 interfaces

JSR-223 defines three main interfaces, of which two (`Invocable` and `Compilable`) are optional. ABCL implements all the three interfaces - `ScriptEngine` and the two optional ones - almost completely. While the JSR-223 API is not specific to a single scripting language, it was designed with languages with a more or less Java-like object model in mind: languages such as Javascript, Python, Ruby, which have a concept of "class" or "object" with "fields" and "methods". Lisp is a bit different, so certain adaptations were made, and in one case a method has been left unimplemented since it does not map at all to Lisp.

The ScriptEngine

The main interface defined by JSR-223, `javax.script.ScriptEngine`, is implemented by the class `org.armedbear.lisp.scripting.AbclScriptEngine`. `AbclScriptEngine` is a singleton, reflecting the fact that ABCL is a singleton as well. You can obtain an instance of `AbclScriptEngine` using the `AbclScriptEngineFactory` or by using the service provider mechanism through `ScriptEngineManager` (refer to the `javax.script` documentation).

3.3.3 Start-up and configuration file

At start-up (i.e. when its constructor is invoked, as part of the static initialization phase of `AbclScriptEngineFactory`) the ABCL script engine attempts to load an "init file" from the classpath (`/abcl-script-config.lisp`). If present, this file can be used to customize the behavior of the engine, by setting a number of variables in the `ABCL-SCRIPT` package. Here is a list of the available variables:

`*use-throwing-debugger*` controls whether ABCL uses a non-standard debugging hook function to throw a Java exception instead of dropping into the debugger in case of unhandled error conditions.

- Default value: `T`
- Rationale: it is more convenient for Java programmers using Lisp as a scripting language to have it return exceptions to Java instead of handling them in the Lisp world.
- Known Issues: the non-standard debugger hook has been reported to misbehave in certain circumstances, so consider disabling it if it doesn't work for you.

`*launch-swank-at-startup*` If true, Swank will be launched at startup. See `*swank-dir*` and `*swank-port*`.

- Default value: `NIL`

`*swank-dir*` The directory where Swank is installed. Must be set if `*launch-swank-at-startup*` is true.

`*swank-port*` The port where Swank will listen for connections. Must be set if `*launch-swank-at-startup*` is true.

- Default value: 4005

Additionally, at startup the AbclScriptEngine will `(require 'asdf)` - in fact, it uses asdf to load Swank.

3.3.4 Evaluation

Code is read and evaluated in the package `ABCL-SCRIPT-USER`. This packages `USE`s the `COMMON-LISP`, `JAVA` and `ABCL-SCRIPT` packages. Future versions of the script engine might make this default package configurable. The `CL:LOAD` function is used under the hood for evaluating code, and thus the behavior of `LOAD` is guaranteed. This allows, among other things, `IN-PACKAGE` forms to change the package in which the loaded code is read.

It is possible to evaluate code in what JSR-223 calls a "ScriptContext" (basically a flat environment of name→value pairs). This context is used to establish special bindings for all the variables defined in it; since variable names are strings from Java's point of view, they are first interned using `READ-FROM-STRING` with, as usual, `ABCL-SCRIPT-USER` as the default package. Variables are declared special because CL's `LOAD`, `EVAL` and `COMPILE` functions work in a null lexical environment and would ignore non-special bindings.

Contrary to what the function `LOAD` does, evaluation of a series of forms returns the value of the last form instead of T, so the evaluation of short scripts does the Right Thing.

3.3.5 Compilation

`AbclScriptEngine` implements the `javax.script.Compilable` interface. Currently it only supports compilation using temporary files. Compiled code, returned as an instance of `javax.script.CompiledScript`, is read, compiled and executed by default in the `abcl-script-user` package, just like evaluated code. In contrast to evaluated code, though, due to the way the ABCL compiler works, compiled code contains no reference to top-level self-evaluating objects (like numbers or strings). Thus, when evaluated, a piece of compiled code will return the value of the last non-self-evaluating form: for example the code "`(do-something) 42`" will return 42 when interpreted, but will return the result of (do-something) when compiled and later evaluated. To ensure consistency of behavior between interpreted and compiled code, make sure the last form is always a compound form - at least `(identity some-literal-object)`. Note that this issue should not matter in real code, where it is unlikely that a top-level self-evaluating form will appear as the last form in a file (in fact, the Common Lisp load function always returns `t` upon success; with JSR-223 this policy has been changed to make evaluation of small code snippets work as intended).

3.3.6 Invocation of functions and methods

AbclScriptEngine implements the `javax.script.Invocable` interface, which allows to directly call Lisp functions and methods, and to obtain Lisp implementations of Java interfaces. This is only partially possible with Lisp since it has functions, but not methods - not in the traditional OO sense, at least, since Lisp methods are not attached to objects but belong to generic functions. Thus, the method `invokeMethod()` is not implemented and throws an `UnsupportedOperationException` when called. The `invokeFunction()` method should be used to call both regular and generic functions.

3.3.7 Implementation of Java interfaces in Lisp

ABCL can use the Java reflection-based proxy feature to implement Java interfaces in Lisp. It has several built-in ways to implement an interface, and supports definition of new ones. The `JAVA:JMAKE-PROXY` generic function is used to make such proxies. It has the following signature:

`jmake-proxy interface implementation &optional lisp-this ==> proxy`

`interface` is a Java interface metaobject (e.g. obtained by invoking `jclass`) or a string naming a Java interface. `implementation` is the object used to implement the interface - several built-in methods of jmake-proxy exist for various types of implementations. `lisp-this` is an object passed to the closures implementing the Lisp "methods" of the interface, and defaults to `NIL`.

The returned proxy is an instance of the interface, with methods implemented with Lisp functions.

Built-in interface-implementation types include:

- a single Lisp function which, upon invocation of any method in the interface, will be passed the method name, the Lisp-this object, and all the parameters. Useful for interfaces with a single method, or to implement custom interface-implementation strategies.
- a hash-map of method-name → Lisp function mappings. Function signature is `(lisp-this &rest args)`.
- a Lisp package. The name of the Java method to invoke is first transformed in an idiomatic Lisp name (`javaMethodName` becomes `JAVA-METHOD-NAME`) and a symbol with that name is searched in the package. If it exists and is fbound, the corresponding function will be called. Function signature is as the hash-table case.

This functionality is exposed by the class `AbclScriptEngine` via the two methods `getInterface(Class)` and `getInterface(Object, Class)`. The former returns an interface implemented with the current Lisp package, the latter allows the programmer to pass an interface-implementation object which will in turn be passed to the `jmake-proxy` generic function.

3.4 Implementation Extension Dictionaries

As outlined by the CLHS ANSI conformance guidelines, we document the extensions to the Armed Bear Common Lisp implementation made accessible to the user by virtue of being an exported symbol in the `java`, `threads`, or `extensions` packages. Additional, higher-level information about the extensions afforded by the implementation can be found in 4 on page 43.

3.4.1 The JAVA Dictionary

The symbols exported from the the `JAVA` package constitute the primary mechanism to interact with Java language constructs within the hosting virtual machine.

Modifying the JVM CLASSPATH

The `JAVA:ADD-TO-CLASSPATH` generic functions allows one to add the specified pathname or list of pathnames to the current classpath used by ABCL, allowing the dynamic loading of JVM objects:

```
CL-USER> (add-to-classpath "/path/to/some.jar")
```

N.b `ADD-TO-CLASSPATH` only affects the classloader used by ABCL (the value of the special variable `JAVA:*CLASSLOADER*`. It has no effect on Java code outside ABCL.

Creating a synthetic Java Class at Runtime

For details on the mechanism available to create a fully synthetic Java class at runtime can be found in `JAVA:JNEW-RUNTIME-CLASS` on 3.4.1.

— Variable: ***java-object-to-string-length*** [**java**]

Length to truncate toString() PRINT-OBJECT output for an otherwise unspecialized JAVA-OBJECT. Can be set to NIL to indicate no limit.

— Variable: +**false**+ [**java**]

The JVM primitive value for boolean false.

— Variable: +**null**+ [**java**]

The JVM null object reference.

— Variable: +**true**+ [**java**]

The JVM primitive value for boolean true.

— Generic Function: **add-to-classpath** [**java**]

Add JAR-OR-JARS to the JVM classpath optionally specifying the CLASS-LOADER to add.

JAR-OR-JARS is either a pathname designating a jar archive or the root directory to search for classes or a list of such values.

— Macro: **chain** [**java**]

Performs chained method invocations.

TARGET is either the receiver object when the first call is a virtual method call or a list in the form (:static <jclass>) when the first method call is a static method call.

OP and each of the OPS are either method designators or lists in the form (<method designator> &rest args), where a method designator is either a string naming a method, or a jmethod object. CHAIN will perform the method call specified by OP on TARGET; then, for each of the OPS, CHAIN will perform the specified method call using the object returned by the previous method call as the receiver, and will ultimately return the result of the last method call. For example, the form:

(chain (:static "java.lang.Runtime") "getRuntime" ("exec" "ls"))

is equivalent to the following Java code:

java.lang.Runtime.getRuntime().exec("ls");

— Macro: **define-java-class** [**java**]

not-documented

— Function: **describe-java-object** [**java**] *object stream*

Print a human friendly description of Java OBJECT to STREAM.

— Function: **dump-classpath** [**java**] *&optional classloader*

not-documented

— Function: **ensure-java-class** [**java**] *jclass*

Attempt to ensure that the Java class referenced by JCLASS exists in the current process of the implementation.

— Function: **ensure-java-object** [**java**] *obj*

Ensures OBJ is wrapped in a JAVA-OBJECT, wrapping it if necessary.

— Function: **get-current-classloader** [**java**]
not-documented

— Function: **get-default-classloader** [**java**]
not-documented

— Function: **jarray-component-type** [**java**] *atype*
Returns the component type of the array type ATYPE

— Function: **jarray-from-list** [**java**] *list*
Return a Java array from LIST whose type is inferred from the first element.
For more control over the type of the array, use JNEW-ARRAY-FROM-LIST.

— Function: **jarray-length** [**java**] *java-array*
Returns the length of a Java primitive array.

— Function: **jarray-ref** [**java**] *java-array &rest indices*
Dereferences the Java array JAVA-ARRAY using the given INDICES, coercing the result into a Lisp object, if possible.

— Function: **jarray-ref-raw** [**java**] *java-array &rest indices*
Dereference the Java array JAVA-ARRAY using the given INDICES. Does not attempt to coerce the result into a Lisp object.

— Function: **jarray-set** [**java**] *java-array new-value &rest indices*
Stores NEW-VALUE at the given INDICES in JAVA-ARRAY.

— Class: **java-class** [**java**]
not-documented

— Class: **java-exception** [**java**]
not-documented

— Function: **java-exception-cause** [**java**] *java-exception*
not-documented

— Class: **java-object** [**java**]
not-documented

— Function: **java-object-p** [**java**] *object*
Returns T if OBJECT is a JAVA-OBJECT.

— Function: **jcall** [**java**] *method-ref instance &rest args*
Invokes the Java method METHOD-REF on INSTANCE with arguments ARGS, coercing the result into a Lisp object, if possible.

— Function: **jcall-raw** [**java**] *method-ref instance &rest args*
Invokes the Java method METHOD-REF on INSTANCE with arguments ARGS. Does not attempt to coerce the result into a Lisp object.

— Function: **jclass [java]** *name-or-class-ref &optional class-loader*

Returns a reference to the Java class designated by NAME-OR-CLASS-REF. If the CLASS-LOADER parameter is passed, the class is resolved with respect to the given ClassLoader.

— Function: **jclass-array-p [java]** *class*

Returns T if CLASS is an array class

— Function: **jclass-constructors [java]** *class*

Returns a vector of constructors for CLASS

— Function: **jclass-field [java]** *class field-name*

Returns the field named FIELD-NAME of CLASS

— Function: **jclass-fields [java]** *class &key declared public*

Returns a vector of all (or just the declared/public, if DECLARED/PUBLIC is true) fields of CLASS

— Function: **jclass-interface-p [java]** *class*

Returns T if CLASS is an interface

— Function: **jclass-interfaces [java]** *class*

Returns the vector of interfaces of CLASS

— Function: **jclass-methods [java]** *class &key declared public*

Return a vector of all (or just the declared/public, if DECLARED/PUBLIC is true) methods of CLASS

— Function: **jclass-name [java]** *class-ref &optional name*

When called with one argument, returns the name of the Java class designated by CLASS-REF. When called with two arguments, tests whether CLASS-REF matches NAME.

— Function: **jclass-of [java]** *object &optional name*

Returns the name of the Java class of OBJECT. If the NAME argument is supplied, verifies that OBJECT is an instance of the named class. The name of the class or nil is always returned as a second value.

— Function: **jclass-superclass [java]** *class*

Returns the superclass of CLASS, or NIL if it hasn't got one

— Function: **jclass-superclass-p [java]** *class-1 class-2*

Returns T if CLASS-1 is a superclass or interface of CLASS-2

— Function: **jcoerce [java]** *object intended-class*

Attempts to coerce OBJECT into a JavaObject of class INTENDED-CLASS. Raises a TYPE-ERROR if no conversion is possible.

— Function: **jconstructor [java]** *class-ref &rest parameter-class-refs*

Returns a reference to the Java constructor of CLASS-REF with the given PARAMETER-CLASS-REFS.

— Function: **jconstructor-params [java]** *constructor*

Returns a vector of parameter types (Java classes) for CONSTRUCTOR

— Function: **jequal [java]** *obj1 obj2*

Compares obj1 with obj2 using java.lang.Object.equals()

— Function: **jfield [java]** *class-ref-or-field field-or-instance &optional instance value*

Retrieves or modifies a field in a Java class or instance.

Supported argument patterns:

Case 1: class-ref field-name: Retrieves the value of a static field.

Case 2: class-ref field-name instance-ref: Retrieves the value of a class field of the instance.

Case 3: class-ref field-name primitive-value: Stores a primitive-value in a static field.

Case 4: class-ref field-name instance-ref value: Stores value in a class field of the instance.

Case 5: class-ref field-name nil value: Stores value in a static field (when value may be confused with an instance-ref).

Case 6: field-name instance: Retrieves the value of a field of the instance. The class is derived from the instance.

Case 7: field-name instance value: Stores value in a field of the instance. The class is derived from the instance.

— Function: **jfield-name [java]** *field*

Returns the name of FIELD as a Lisp string

— Function: **jfield-raw [java]** *class-ref-or-field field-or-instance &optional instance value*

Retrieves or modifies a field in a Java class or instance. Does not attempt to coerce its value or the result into a Lisp object.

Supported argument patterns:

Case 1: class-ref field-name: Retrieves the value of a static field.

Case 2: class-ref field-name instance-ref: Retrieves the value of a class field of the instance.

Case 3: class-ref field-name primitive-value: Stores a primitive-value in a static field.

Case 4: class-ref field-name instance-ref value: Stores value in a class field of the instance.

Case 5: class-ref field-name nil value: Stores value in a static field (when value may be confused with an instance-ref).

Case 6: field-name instance: Retrieves the value of a field of the instance. The class is derived from the instance.

Case 7: field-name instance value: Stores value in a field of the instance. The class is derived from the instance.

— Function: **jfield-type [java]** *field*

Returns the type (Java class) of FIELD

— Function: **jinput-stream** [**java**] *pathname*

Returns a java.io.InputStream for resource denoted by PATHNAME.

— Function: **jinstance-of-p** [**java**] *obj class*

OBJ is an instance of CLASS (or one of its subclasses)

— Function: **jinterface-implementation** [**java**] *interface &rest method-names-and-defs*

Creates and returns an implementation of a Java interface with methods calling Lisp closures as given in METHOD-NAMES-AND-DEFS.

INTERFACE is either a Java interface or a string naming one.

METHOD-NAMES-AND-DEFS is an alternating list of method names (strings) and method definitions (closures).

For missing methods, a dummy implementation is provided that returns nothing or null depending on whether the return type is void or not. This is for convenience only, and a warning is issued for each undefined method.

— Function: **jmake-invocation-handler** [**java**] *function*

not-documented

— Generic Function: **jmake-proxy** [**java**]

Returns a proxy Java object implementing the provided interface(s) using methods implemented in Lisp - typically closures, but implementations are free to provide other mechanisms. You can pass an optional 'lisp-this' object that will be passed to the implementing methods as their first argument. If you don't provide this object, NIL will be used. The second argument of the Lisp methods is the name of the Java method being implemented. This has the implication that overloaded methods are merged, so you have to manually discriminate them if you want to. The remaining arguments are java-objects wrapping the method's parameters.

— Function: **jmember-protected-p** [**java**] *member*

MEMBER is a protected member of its declaring class

— Function: **jmember-public-p** [**java**] *member*

MEMBER is a public member of its declaring class

— Function: **jmember-static-p** [**java**] *member*

MEMBER is a static member of its declaring class

— Function: **jmethod** [**java**] *class-ref method-name &rest parameter-class-refs*

Returns a reference to the Java method METHOD-NAME of CLASS-REF with the given PARAMETER-CLASS-REFS.

— Macro: **jmethod-let** [**java**]

not-documented

— Function: **jmethod-name** [**java**] *method*

Returns the name of METHOD as a Lisp string

— Function: **jmethod-params [java]** *method*

Returns a vector of parameter types (Java classes) for METHOD

— Function: **jmethod-return-type [java]** *method*

Returns the result type (Java class) of the METHOD

— Function: **jnew [java]** *constructor &rest args*

Invokes the Java constructor CONSTRUCTOR with the arguments ARGS.

— Function: **jnew-array [java]** *element-type &rest dimensions*

Creates a new Java array of type ELEMENT-TYPE, with the given DIMENSIONS.

— Function: **jnew-array-from-array [java]** *element-type array*

Returns a new Java array with base type ELEMENT-TYPE (a string or a class-ref) initialized from ARRAY.

— Function: **jnew-array-from-list [java]** *element-type list*

Returns a new Java array with base type ELEMENT-TYPE (a string or a class-ref) initialized from a Lisp list.

— Function: **jnew-runtime-class [java]** *class-name &rest args &key (superclass java.lang.Object) interfaces constructors methods fields (access-flags (quote (public))) annotations (class-loader (make-memory-class-loader))*

Creates and loads a Java class with methods calling Lisp closures as given in METHODS. CLASS-NAME and SUPER-NAME are strings, INTERFACES is a list of strings, CONSTRUCTORS, METHODS and FIELDS are lists of constructor, method and field definitions.

Constructor definitions - currently NOT supported - are lists of the form (argument-types function &optional super-invocation-arguments) where argument-types is a list of strings and function is a lisp function of (1+ (length argument-types)) arguments; the instance ('this') is passed in as the last argument. The optional super-invocation-arguments is a list of numbers between 1 and (length argument-types), where the number k stands for the kth argument to the just defined constructor. If present, the constructor of the superclass will be called with the appropriate arguments. E.g., if the constructor definition is (("java.lang.String" "int") #'(lambda (string i this) ...) (2 1)) then the constructor of the superclass with argument types (int, java.lang.String) will be called with the second and first arguments.

Method definitions are lists of the form

(METHOD-NAME RETURN-TYPE ARGUMENT-TYPES FUNCTION &key MODIFIERS ANNOTATIONS)

where METHOD-NAME is a string RETURN-TYPE denotes the type of the object returned by the method ARGUMENT-TYPES is a list of parameters to the method

The types are either strings naming fully qualified java classes or Lisp keywords referring to primitive types (:void, :int, etc.).

FUNCTION is a Lisp function of minimum arity (1+ (length argument-types)). The instance ('this') is passed as the first argument.

Field definitions are lists of the form (field-name type &key modifiers annotations).

— Function: **jnull-ref-p** [**java**] *object*
Returns a non-NIL value when the JAVA-OBJECT 'object' is 'null', or signals a TYPE-ERROR condition if the object isn't of the right type.

— Function: **jobject-class** [**java**] *obj*
Returns the Java class that OBJ belongs to

— Function: **jobject-lisp-value** [**java**] *java-object*
Attempts to coerce JAVA-OBJECT into a Lisp object.

— Function: **jproperty-value** [**java**] *object property*
setf-able access on the Java Beans notion of property named PROPETRY on OBJECT.

— Function: **jregister-handler** [**java**] *object event handler &key data count*
not-documented

— Function: **jresolve-method** [**java**] *method-name instance &rest args*
Finds the most specific Java method METHOD-NAME on INSTANCE applicable to arguments ARGS. Returns NIL if no suitable method is found. The algorithm used for resolution is the same used by JCALL when it is called with a string as the first parameter (METHOD-REF).

— Function: **jrun-exception-protected** [**java**] *closure*
Invokes the function CLOSURE and returns the result. Signals an error if stack or heap exhaustion occurs.

— Function: **jstatic** [**java**] *method class &rest args*
Invokes the static method METHOD on class CLASS with ARGS.

— Function: **jstatic-raw** [**java**] *method class &rest args*
Invokes the static method METHOD on class CLASS with ARGS. Does not attempt to coerce the arguments or result into a Lisp object.

— Function: **make-classloader** [**java**] *&optional parent*
not-documented

— Function: **make-immediate-object** [**java**] *object &optional type*
Attempts to coerce a given Lisp object into a java-object of the given type. If type is not provided, works as jobject-lisp-value. Currently, type may be :BOOLEAN, treating the object as a truth value, or :REF, which returns Java null if NIL is provided.
Deprecated. Please use JAVA:+NULL+, JAVA:+TRUE+, and JAVA:+FALSE+ for constructing wrapped primitive types, JAVA:JOBJECT-LISP-VALUE for converting a JAVA:JAVA-OBJECT to a Lisp value, or JAVA:JNULL-REF-P to distinguish a wrapped null JAVA-OBJECT from NIL.

— Function: **register-java-exception** [**java**] *exception-name condition-symbol*
Registers the Java Throwable named by the symbol EXCEPTION-NAME as the condition designated by CONDITION-SYMBOL. Returns T if successful, NIL if not.

— Function: **unregister-java-exception** [**java**] *exception-name*

Unregisters the Java Throwable EXCEPTION-NAME previously registered by REGISTER-JAVA-EXCEPTION.

3.4.2 The THREADS Dictionary

The extensions for handling multithreaded execution are collected in the `THREADS` package. Most of the abstractions in Doug Lea's excellent `java.util.concurrent` packages may be manipulated directly via the JSS contrib to great effect [Lea98]

— Function: **current-thread** [**threads**]

Returns a reference to invoking thread.

— Function: **destroy-thread** [**threads**]

not-documented

— Function: **get-mutex** [**threads**] *mutex*

Acquires a lock on the 'mutex'.

— Function: **interrupt-thread** [**threads**] *thread function &rest args*

Interrupts THREAD and forces it to apply FUNCTION to ARGS. When the function returns, the thread's original computation continues. If multiple interrupts are queued for a thread, they are all run, but the order is not guaranteed.

— Function: **mailbox-empty-p** [**threads**] *mailbox*

Returns non-NIL if the mailbox can be read from, NIL otherwise.

— Function: **mailbox-peek** [**threads**] *mailbox*

Returns two values. The second returns non-NIL when the mailbox is empty. The first is the next item to be read from the mailbox.

Note that due to multi-threading, the first value returned upon peek, may be different from the one returned upon next read in the calling thread.

— Function: **mailbox-read** [**threads**] *mailbox*

Blocks on the mailbox until an item is available for reading. When an item is available, it is returned.

— Function: **mailbox-send** [**threads**] *mailbox item*

Sends an item into the mailbox, notifying 1 waiter to wake up for retrieval of that object.

— Function: **make-mailbox** [**threads**] *&key ((queue g284829) NIL)*

not-documented

— Function: **make-mutex** [**threads**] *&key ((in-use g285092) NIL)*

not-documented

— Function: **make-thread** [**threads**] *function &key name*

not-documented

— Function: **make-thread-lock** [**threads**]

Returns an object to be used with the 'with-thread-lock' macro.

— Function: **mapcar-threads** [**threads**]

not-documented

— Function: **object-notify** [**threads**] *object*

Wakes up a single thread that is waiting on OBJECT's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. A thread waits on an object's monitor by calling one of the wait methods.

— Function: **object-notify-all** [**threads**] *object*

Wakes up all threads that are waiting on this OBJECT's monitor. A thread waits on an object's monitor by calling one of the wait methods.

— Function: **object-wait** [**threads**] *object &optional timeout*

Causes the current thread to block until object-notify or object-notify-all is called on OBJECT. Optionally unblock execution after TIMEOUT seconds. A TIMEOUT of zero means to wait indefinitely. A non-zero TIMEOUT of less than a nanosecond is interpolated as a nanosecond wait. See the documentation of java.lang.Object.wait() for further information.

— Function: **release-mutex** [**threads**] *mutex*

Releases a lock on the 'mutex'.

— Special Operator: **synchronized-on** [**threads**]

not-documented

— Class: **thread** [**threads**]

not-documented

— Function: **thread-alive-p** [**threads**] *thread*

Boolean predicate whether THREAD is alive.

— Function: **thread-join** [**threads**] *thread*

Waits for thread to finish.

— Function: **thread-name** [**threads**]

not-documented

— Function: **threadp** [**threads**]

not-documented

— Macro: **with-mutex** [**threads**]

Acquires a lock on 'mutex', executes the body and releases the lock.

— Macro: **with-thread-lock** [**threads**]

Acquires a lock on the 'lock', executes 'body' and releases the lock.

— Function: **yield** [**threads**]

A hint to the scheduler that the current thread is willing to yield its current use of a processor. The scheduler is free to ignore this hint.

See java.lang.Thread.yield().

3.4.3 The EXTENSIONS Dictionary

The symbols in the `extensions` package (often referenced by its shorter nickname `ext`) constitutes extensions to the ANSI standard that are potentially useful to the user. They include functions for manipulating network sockets, running external programs, registering object finalizers, constructing reference weakly held by the garbage collector and others.

See [Rho09] for a generic function interface to the native JVM contract for `java.util.List`.

— Macro: **%caddr** [**extensions**]
not-documented

— Macro: **%cadr** [**extensions**]
not-documented

— Macro: **%car** [**extensions**]
not-documented

— Macro: **%cdr** [**extensions**]
not-documented

— Variable: ***autoload-verbose*** [**extensions**]
not-documented

— Variable: ***batch-mode*** [**extensions**]
not-documented

— Variable: ***command-line-argument-list*** [**extensions**]
not-documented

— Variable: ***debug-condition*** [**extensions**]
not-documented

— Variable: ***debug-level*** [**extensions**]
not-documented

— Variable: ***disassembler*** [**extensions**]
not-documented

— Variable: ***ed-functions*** [**extensions**]
not-documented

— Variable: ***enable-inline-expansion*** [**extensions**]
not-documented

— Variable: ***inspector-hook*** [**extensions**]
not-documented

— Variable: ***lisp-home*** [**extensions**]
not-documented

— Variable: ***load-truename-fasl*** [**extensions**]
not-documented

— Variable: ***print-structure*** [**extensions**]
not-documented

— Variable: ***require-stack-frame*** [**extensions**]
not-documented

— Variable: ***saved-backtrace*** [**extensions**]
not-documented

— Variable: ***suppress-compiler-warnings*** [**extensions**]
not-documented

— Variable: ***warn-on-redefinition*** [**extensions**]
not-documented

— Function: **add-package-local-nickname** [**extensions**] *local-nickname actual-package &optional (package-designator *package*)*
not-documented

— Function: **adjoin-eql** [**extensions**] *item list*
not-documented

— Function: **arglist** [**extensions**] *extended-function-designator*
not-documented

— Function: **assq** [**extensions**]
not-documented

— Function: **assql** [**extensions**]
not-documented

— Function: **autoload** [**extensions**] *symbol-or-symbols &optional filename*
Setup the autoload for SYMBOL-OR-SYMBOLS optionally corresponding to FILENAME.

— Function: **autoload-macro** [**extensions**]
not-documented

— Function: **autoload-ref-p** [**extensions**] *symbol*
Boolean predicate for whether SYMBOL has generalized reference functions which need to be resolved.

— Function: **autoload-setf-expander** [**extensions**] *symbol-or-symbols filename*
Setup the autoload for SYMBOL-OR-SYMBOLS on the setf-expander from FILENAME.

— Function: **autoload-setf-function** [**extensions**] *symbol-or-symbols filename*
Setup the autoload for SYMBOL-OR-SYMBOLS on the setf-function from FILENAME.

— Function: **autoloadp** [**extensions**] *symbol*
Boolean predicate for whether SYMBOL stands for a function that currently needs to be autoloaded.

— Function: **cancel-finalization** [**extensions**] *object*

not-documented

— Function: **char-to-utf8** [**extensions**]

not-documented

— Function: **charpos** [**extensions**] *stream*

not-documented

— Function: **classp** [**extensions**]

not-documented

— Macro: **collect** [**extensions**]

Collect ((Name [Initial-Value] [Function])*) Form* Collect some values somehow. Each of the collections specifies a bunch of things which collected during the evaluation of the body of the form. The name of the collection is used to define a local macro, a la MACROLET. Within the body, this macro will evaluate each of its arguments and collect the result, returning the current value after the collection is done. The body is evaluated as a PROGN; to get the final values when you are done, just call the collection macro with no arguments.

Initial-Value is the value that the collection starts out with, which defaults to NIL. Function is the function which does the collection. It is a function which will accept two arguments: the value to be collected and the current collection. The result of the function is made the new value for the collection. As a totally magical special-case, the Function may be Collect, which tells us to build a list in forward order; this is the default. If an Initial-Value is supplied for Collect, the stuff will be rplacd'd onto the end. Note that Function may be anything that can appear in the functional position, including macros and lambdas.

— Function: **compile-system** [**extensions**] *&key quit (zip t) (cls-ext *compile-file-class-extension*) (abcl-ext *compile-file-type*) output-path*

not-documented

— Variable: **double-float-negative-infinity** [**extensions**]

not-documented

— Variable: **double-float-positive-infinity** [**extensions**]

not-documented

— Function: **dump-java-stack** [**extensions**]

not-documented

— Function: **exit** [**extensions**] *&key status*

not-documented

— Function: **featurep** [**extensions**] *form*

not-documented

— Function: **file-directory-p** [**extensions**] *pathspec &key (wild-error-p t)*
not-documented

— Function: **finalize** [**extensions**] *object function*
not-documented

— Function: **fixnump** [**extensions**]
not-documented

— Function: **gc** [**extensions**]
not-documented

— Function: **get-floating-point-modes** [**extensions**]
not-documented

— Function: **get-pid** [**extensions**]
Get the process identifier of this lisp process.
Used to be in SLIME but generally useful, so now back in ABCL proper.

— Function: **get-socket-stream** [**extensions**] *socket &key (element-type (quote character)) (external-format default)*
:ELEMENT-TYPE must be CHARACTER or (UNSIGNED-BYTE 8); the default is CHARACTER. EXTERNAL-FORMAT must be of the same format as specified for OPEN.

— Function: **get-time-zone** [**extensions**] *time-in-millis*
Returns as the first value the timezone difference in hours from the Greenwich meridian for TIME-IN-MILLIS via the Daylight Savings Time assumptions that were in place at the instant's occurance. Returns as the second value a boolean as to whether daylight savings time was in effect at the occurance.

— Function: **getenv** [**extensions**] *variable*
Return the value of the environment VARIABLE if it exists, otherwise return NIL.

— Function: **getenv-all** [**extensions**] *variable*
Returns all environment variables as an alist containing (name . value)

— Function: **init-gui** [**extensions**]
not-documented

— Function: **interrupt-lisp** [**extensions**]
not-documented

— Class: **jar-pathname** [**extensions**]
not-documented

— Function: **macroexpand-all** [**extensions**] *form &optional env*
not-documented

— Class: **mailbox** [**extensions**]
not-documented

— Function: **make-dialog-prompt-stream** [**extensions**]
not-documented

— Function: **make-server-socket** [**extensions**] *port*
Create a TCP server socket listening for clients on PORT.

— Function: **make-slime-input-stream** [**extensions**] *function output-stream*
not-documented

— Function: **make-slime-output-stream** [**extensions**] *function*
not-documented

— Function: **make-socket** [**extensions**] *host port*
Create a TCP socket for client communication to HOST on PORT.

— Function: **make-temp-directory** [**extensions**]
Create and return the pathname of a previously non-existent directory.

— Function: **make-temp-file** [**extensions**] *&key prefix suffix*
Create and return the pathname of a previously non-existent file.

— Function: **make-weak-reference** [**extensions**] *obj*
not-documented

— Function: **memq** [**extensions**] *item list*
not-documented

— Function: **memql** [**extensions**] *item list*
not-documented

— Variable: **most-negative-java-long** [**extensions**]
not-documented

— Variable: **most-positive-java-long** [**extensions**]
not-documented

— Class: **mutex** [**extensions**]
not-documented

— Function: **neq** [**extensions**] *obj1 obj2*
not-documented

— Class: **nil-vector** [**extensions**]
not-documented

— Function: **os-unix-p** [**extensions**]
Is the underlying operating system some Unix variant?

— Function: **os-windows-p** [**extensions**]
Is the underlying operating system Microsoft Windows?

— Function: **package-local-nicknames** [**extensions**] *package*
not-documented

— Function: **package-locally-nicknamed-by-list** [**extensions**] *package*
not-documented

— Function: **pathname-jar-p** [**extensions**]
not-documented

— Function: **pathname-url-p** [**extensions**] *pathname*
Predicate for whether PATHNAME references a URL.

— Function: **precompile** [**extensions**] *name &optional definition*
not-documented

— Function: **probe-directory** [**extensions**] *pathspec*
not-documented

— Function: **quit** [**extensions**] *&key status*
not-documented

— Function: **read-timeout** [**extensions**] *socket seconds*
Time in SECONDS to set local implementation of 'SO_RCVTIMEO' on SOCKET.

— Function: **remove-package-local-nickname** [**extensions**] *old-nickname &optional package-designator*
not-documented

— Function: **resolve** [**extensions**] *symbol*
Resolve the function named by SYMBOL via the autoloader mechanism. Returns either the function or NIL if no resolution was possible.

— Function: **run-shell-command** [**extensions**] *command &key directory (output *standard-output*)*
not-documented

— Function: **server-socket-close** [**extensions**] *socket*
Close the server SOCKET.

— Function: **set-floating-point-modes** [**extensions**] *&key traps*
not-documented

— Function: **show-restarts** [**extensions**] *restarts stream*
not-documented

— Function: **simple-string-fill** [**extensions**]
not-documented

— Function: **simple-string-search** [**extensions**]
not-documented

— Variable: **single-float-negative-infinity** [**extensions**]
not-documented

— Variable: **single-float-positive-infinity** [**extensions**]
not-documented

— Class: **slime-input-stream** [**extensions**]
not-documented

— Class: **slime-output-stream** [**extensions**]
not-documented

— Function: **socket-accept** [**extensions**] *socket*
Block until able to return a new socket for handling a incoming request to the specified server SOCKET.

— Function: **socket-close** [**extensions**] *socket*
Close the client SOCKET.

— Function: **socket-local-address** [**extensions**] *socket*
Returns the local address of the SOCKET as a dotted quad string.

— Function: **socket-local-port** [**extensions**] *socket*
Returns the local port number of the SOCKET.

— Function: **socket-peer-address** [**extensions**] *socket*
Returns the peer address of the SOCKET as a dotted quad string.

— Function: **socket-peer-port** [**extensions**] *socket*
Returns the peer port number of the given SOCKET.

— Function: **source** [**extensions**]
not-documented

— Function: **source-file-position** [**extensions**]
not-documented

— Function: **source-pathname** [**extensions**] *symbol*
Returns either the pathname corresponding to the file from which this symbol was compiled,or the keyword :TOP-LEVEL.

— Function: **special-variable-p** [**extensions**]
not-documented

— Function: **string-find** [**extensions**] *char string*
not-documented

— Function: **string-input-stream-current** [**extensions**] *stream*
not-documented

— Function: **string-position** [**extensions**]
not-documented

— Function: **style-warn** [**extensions**] *format-control &rest format-arguments*
not-documented

— Macro: **truly-the** [**extensions**]
not-documented

— Function: **uptime** [**extensions**]
not-documented

— Function: **uri-decode** [**extensions**]
not-documented

— Function: **uri-encode** [**extensions**]
not-documented

— Class: **url-pathname** [**extensions**]
not-documented

— Function: **url-pathname-authority** [**extensions**] *p*
not-documented

— Function: **url-pathname-fragment** [**extensions**] *p*
not-documented

— Function: **url-pathname-query** [**extensions**] *p*
not-documented

— Function: **url-pathname-scheme** [**extensions**] *p*
not-documented

— Class: **weak-reference** [**extensions**]
not-documented

— Function: **weak-reference-value** [**extensions**] *obj*
Returns two values, the first being the value of the weak ref,the second T if the reference is valid, or NIL if it hasbeen cleared.

— Function: **write-timeout** [**extensions**] *socket seconds*
No-op setting of write timeout to SECONDS on SOCKET.

Chapter 4

Beyond ANSI

Naturally, in striving to be a useful contemporary Common Lisp implementation, ABCL endeavors to include extensions beyond the ANSI specification which are either widely adopted or are especially useful in working with the hosting JVM. This chapter documents such extensions beyond ANSI conformation.

4.1 Compiler to Java Virtual Machine Bytecode

The `CL:COMPILE-FILE` interface emits a packed fasl[1] format whose `CL:PATHNAME` has the `TYPE` "abcl". Structurally, ABCL's fasls are operating system neutral byte archives packaged in the zip compression format which contain artifacts whose loading `CL:LOAD` understands. Internally, our fasls contain a piece of Lisp that `CL:LOAD` interprets as instructions to load the Java classes emitted by the ABCL Lisp compiler. The classes emitted by the ABCL compiler have a JVM class file version of "49.0".

4.1.1 Compiler Diagnostics

By default, the interface to the compiler does not signal warnings that result in its invocation, outputing diagnostics to the standard reporting stream. The generalized boolean `JVM:*RESIGNAL-COMPILER-WARNINGS*` provides the interface to enabling the compiler to signal all warnings.

4.1.2 Decompilation

Since ABCL compiles to JVM bytecode, the `CL:DISASSEMBLE` function provides introspection for the result of that compilation. By default the implementation attempts to find and use the `javap` command line tool shipped as part of the Java Development Kit to disassemble the results. Code for the use of additional JVM bytecode introspection tools is packaged as part of the ABCL-INTROSPECT contrib. After loading one of these tools via ASDF, the `SYS:CHOOSE-DISASSEMBLER` function can be used to select the tool used by `CL:DISASSEMBLE`. See 5.5.1 on 53 for further details.

4.2 Pathname

We implement a comprehensive extension to the `CL:PATHNAME` that allows for the description and retrieval of resources named in a URI [2] scheme that the JVM "understands". By definition, support is built-in into the JVM to access the "http" and "https" schemes but addi-

[1] The term "fasl" is short for "fast loader", which in Common Lisp implementations refers

[2] A URI is essentially a superset of what is commonly understood as a URL We sometimes use the term URL as shorthand in describing the URL Pathnames, even though the corresponding encoding is more akin to a URI as described in RFC3986 [BLFM05].

tional protocol handlers may be installed at runtime by having JVM symbols present in the `sun.net.protocol.dynamic` package. See [Mas00] for more details.

ABCL has created specializations of the ANSI `CL:PATHNAME` object to enable to use of URIs to address dynamically loaded resources for the JVM. The `EXT:URL-PATHNAME` specialization has a corresponding URI whose canonical representation is defined to be the `NAMESTRING` of the `CL:PATHNAME`. The `EXT:JAR-PATHNAME` extension further specializes the the `EXT:URL-PATHNAME` to provide access to components of zip archives.

```
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ext:   <http://abcl.org/cl/package/extensions/> .
@prefix cl:    <http://abcl.org/cl/package/common-lisp/> .

<ext:jar-pathname>    rdfs:subClassOf <ext:url-pathname> .
<ext:url-pathname>    rdfs:subClassOf <cl:pathname> .
<cl:logical-pathname> rdfs:subClassOf <cl:pathname> .
```

Both the `EXT:URL-PATHNAME` and `EXT:JAR-PATHNAME` objects may be broadly used anywhere a `CL:PATHNAME` is accepted with the general caveat that stream obtained via `CL:OPEN` on either subtype cannot be the target of write operations.

URL-PATHNAME

A `URL-PATHNAME` denotes a source of bytes addressable by its corresponding namestring interpreted as a URL.

A `EXT:URL-PATHNAME` always has a `HOST` component that is a property list. The values of the `HOST` property list are always character strings. The allowed keys have the following meanings:

:SCHEME Scheme of URI ("http", "ftp", "bundle", etc.)

:AUTHORITY Valid authority according to the URI scheme. For "http" this could be "example.org:8080".

:QUERY The query of the URI

:FRAGMENT The fragment portion of the URI

If the :SCHEME property is missing, it is assumed to be "file" denoting a reference to a file on the local filesystem

In order to encapsulate the implementation decisions for these meanings, the following functions provide a SETF-able API for reading and writing such values: `URL-PATHNAME-QUERY`, `URL-PATHNAME-FRAGMENT`, `URL-PATHNAME-AUTHORITY`, and `URL-PATHNAME-SCHEME`. The specific subtype of a Pathname may be determined with the predicates `PATHNAME-URL-P` and `PATHNAME-JAR-P`.

Any results of canonicalization procedures performed on a object of type `EXT:URL-PATHNAME` via local or network resolutions discarded between attempts (i.e. the implementation does not attempt to cache the results of current name resolution of the URI for underlying resource unless it is requested to be resolved.) Upon resolution, any canonicalization procedures followed in resolving the resource (e.g. following redirects) are discarded. Users may programatically initiate a new, local computation of the resolution of the resource by applying the `CL:TRUENAME` function to a `EXT:URL-PATHNAME` object. Depending on the reliability and properties of your local REST infrastructure, these results may not necessarily be idempotent over time[3]. A future implementation may attempt to expose an API to observer/customize the content negotiation initiated during retrieval of the representation of a given resource, which is currently handled at the application level.

The implementation of `EXT:URL-PATHNAME` allows the ABCL user to dynamically load code from the network. For example, Quicklisp ([Bea]) may be completely installed from the REPL to download and execute the Quicklisp setup code via:

[3] See [Eve11] for the design and implementation notes for the technical details

```
CL-USER> (load "https://beta.quicklisp.org/quicklisp.lisp")
```

Implementation of a JAR-PATHNAME

The implementation currently extends ANSI conformance by `DEVICE` possibly be a list of pathnames. objects. The first member of this list will be a `EXT:URL-PATHNAME` pointing either to a source of bytes on the local filesystem or as a remote source via an stream-oriented messaging protocol such as HTTPS. The rest of the list contains "traditional" `CL:PATHNAME` objects denoting successive relative archive paths.

In ABCL, the `DEVICE` can be either a string either denoting a drive letter or a UNC mount under DOS or a list of one or more elements. If `DEVICE` is a list, it denotes a `EXT:JAR-PATHNAME`. Passing any pathname with a `DEVICE` that is such a list to `CL:TRUENAME` will return a object whose Lisp type is `EXT:JAR-PATHNAME`, and whose `DEVICE` will contain references to objects normalized to the following gcontraints. The first entry in the `DEVICE` list of `EXT:JAR-PATHNAME` will be a `EXT:URL-PATHNAME` designating the root source of bytes for root archive compressed via the ZIP compression algorithm. If any, the remaining elements of the list contain simple `CL:PATHNAME` objects representing subsequent entries objects [4]

In order to implement useful behavior of merging with pathname defaults, the implementation will contain the `:unspecific` keyword in any truename that wasn't explicitly merging with a `EXT:JAR-PATHNAME`. We extend the semantics for the usual PATHNAME merge semantics when `*DEFAULT-PATHNAME-DEFAULTS*` contains a `EXT:JAR-PATHNAME` with the "do what I mean" algorithm described in 1.1 on page 7.

4.3 Package-Local Nicknames

ABCL allows giving packages local nicknames: they allow short and easy-to-use names to be used without fear of name conflict associated with normal nicknames.[5]

A local nickname is valid only when inside the package for which it has been specified. Different packages can use same local nickname for different global names, or different local nickname for same global name.

Symbol `:package-local-nicknames` in `*features*` denotes the support for this feature.

The options to `defpackage` are extended with a new option `:local-nicknames (local-nickname actual-package-name)*`.

The new package has the specified local nicknames for the corresponding actual packages.

Example:

```
(defpackage :bar (:intern "X"))
(defpackage :foo (:intern "X"))
(defpackage :quux (:use :cl)
  (:local-nicknames (:bar :foo) (:foo :bar)))
(find-symbol "X" :foo) ; => FOO::X
(find-symbol "X" :bar) ; => BAR::X
(let ((*package* (find-package :quux)))
  (find-symbol "X" :foo))                ; => BAR::X
(let ((*package* (find-package :quux)))
  (find-symbol "X" :bar))                ; => FOO::X
```

— Function: **package-local-nicknames** [**ext**] *package-designator*

Returns an ALIST of `(local-nickname . actual-package)` describing the nicknames local to the designated package.

[4] This allows for reference to arbitrarily nested ZIP archives, which is the case when the an ABCL fasl is included in in a jar archive.

[5] Package-local nicknames were originally developed in SBCL.

When in the designated package, calls to `find-package` with any of the local-nicknames will return the corresponding actual-package instead. This also affects all implied calls to `find-package`, including those performed by the reader.

When printing a package prefix for a symbol with a package local nickname, the local nickname is used instead of the real name in order to preserve print-read consistency.

— Function: **package-locally-nicknamed-by-list** [**ext**] *package-designator*

Returns a list of packages which have a local nickname for the designated package.

— Function: **add-package-local-nickname** [**ext**] *local-nickname actual-package &optional package-designator*

Adds `local-nickname` for `actual-package` in the designated package, defaulting to current package. `local-nickname` must be a string designator, and `actual-package` must be a package designator.

Returns the designated package.

Signals an error if `local-nickname` is already a package local nickname for a different package, or if `local-nickname` is one of "CL", "COMMON-LISP", or, "KEYWORD", or if `local-nickname` is a global name or nickname for the package to which the nickname would be added.

When in the designated package, calls to `find-package` with the `local-nickname` will return the package the designated `actual-package` instead. This also affects all implied calls to `find-package`, including those performed by the reader.

When printing a package prefix for a symbol with a package local nickname, local nickname is used instead of the real name in order to preserve print-read consistency.

— Function: **remove-package-local-nickname** [**ext**] *old-nickname &optional package-designator*

If the designated package had `old-nickname` as a local nickname for another package, it is removed. Returns true if the nickname existed and was removed, and `nil` otherwise.

4.4 Extensible Sequences

The SEQUENCE package fully implements Christopher Rhodes' proposal for extensible sequences. These user extensible sequences are used directly in `java-collections.lisp` provide these CLOS abstractions on the standard Java collection classes as defined by the `java.util.List` contract.

This extension is not automatically loaded by the implementation. It may be loaded via:

```
CL-USER> (require :java-collections)
```

if both extensible sequences and their application to Java collections is required, or

```
CL-USER> (require :extensible-sequences)
```

if only the extensible sequences API as specified in [Rho09] is required.

Note that `(require :java-collections)` must be issued before `java.util.List` or any subclass is used as a specializer in a CLOS method definition (see the section below).

See Rhodes2007 [Rho09] for the an overview of the abstractions of the `java.util.collection` package afforded by `JAVA-COLLECTIONS`.

4.5 Extensions to CLOS

4.5.1 Metaobject Protocol

ABCL implements the metaobject protocol for CLOS as specified in (A)MOP. The symbols are exported from the package `MOP`.

Contrary to the AMOP specification and following SBCL's lead, the metaclass `funcallable-standard-object` has `funcallable-standard-class` as metaclass instead of `standard-class`.

ABCL's fidelity to the AMOP specification is codified as part of Pascal Costanza's `closer-mop` ?? [Cos11].

4.5.2 Specializing on Java classes

There is an additional syntax for specializing the parameter of a generic function on a java class, viz. `(java:jclass CLASS-STRING)` where `CLASS-STRING` is a string naming a Java class in dotted package form.

For instance the following specialization would perhaps allow one to print more information about the contents of a `java.util.Collection` object

```
(defmethod print-object ((coll (java:jclass "java.util.Collection"))
                         stream)
  ;;; ...
)
```

If the class had been loaded via a classloader other than the original the class you wish to specialize on, one needs to specify the classloader as an optional third argument.

```
(defparameter *other-classloader*
  (jcall "getBaseLoader" cl-user::*classpath-manager*))

(defmethod print-object
   ((device-id (java:jclass "dto.nbi.service.hdm.alcatel.com.NBIDeviceID"
                            *other-classloader*))
    stream)
  ;;; ...
)
```

4.6 Extensions to the Reader

We implement a special hexadecimal escape sequence for specifying 32 bit characters to the Lisp reader[6], namely we allow a sequences of the form `#\U`*`xxxx`* to be processed by the reader as character whose code is specified by the hexadecimal digits *`xxxx`*. The hexadecimal sequence may be one to four digits long.

Note that that the reader escaped sequence is never output by the implementation. Instead, the implementation emits the bytes corresponding Unicode character is output for characters whose code is greater than 0x00ff.

4.7 Overloading of the CL:REQUIRE Mechanism

The `CL:REQUIRE` mechanism is overloaded by attaching these semantics to the execution of `REQUIRE` on the following symbols:

[6]This represents a compromise with contemporary in 2011 32bit hosting architectures for which we wish to make text processing efficient. Should the User require more control over UNICODE processing we recommend Edi Weisz' excellent work with |FLEXI-STREAMS which we fully support

`ASDF`
: Loads the ASDF version shipped with the implementation. After the evaluation of this symbols, symbols passed to `CL:REQUIRE` which are otherwise unresolved, are passed to ASDF for a chance for resolution. This means, for instance if `CL-PPCRE` can be located as a loadable ASDF system `(require :cl-ppcre)` is equivalent to `(asdf:load-system :cl-ppcre)`.

`ABCL-CONTRIB`
: Locates and pushes the top-level contents of "abcl-contrib.jar" into the ASDF central registry.

  `abcl-asdf`
  : Functions for loading JVM artifacts dynamically, hooking into ASDF 3 objects where possible. See 5.1 on page 51.

  `asdf-jar`
  : Package addressable JVM artifacts via `abcl-asdf` descriptions as a single binary artifact including recursive dependencies. See 5.2 on page 52.

  `jna` Allows dynamic access to specify Java Native Interface (JNI) facility providing C-linkage to shared objects by dynamically loading the 'jna.jar' artifact via Maven[7]

  `quicklisp-abcl`
  : Load Quicklisp, bootstrap a a local Quicklisp installation via the ASDF:IRI type introduced via ABCL-ASDF.

  `jfli`
  : A descendant of Rich Hickey's pre-Clojure work on the JVM.

  `jss` Introduces dynamic inspection of present symbols via the `SHARPSIGN-QUOTATION_MARK` macros as Java Syntax Sucks. See 5.3 on page **??** for more details.

  `abcl-introspect`
  : Provides a framework for introspecting runtime Java and Lisp object values. Include packaging for installing and using java decompilation tools for use with `CL:DISASSEMBLE`. See **??** on **??** for futher information.

  `abcl-build`
  : Provides a toolkit for building ABCL from source, as well as installing the necessary tools for such builds. See **??** on page **??**.

  `named-readtables`
  : Provides a namespace for readtables akin to the already-existing namespace of packages. See 5.7 on 55 for further information.

The user may extend the `CL:REQUIRE` mechanism by pushing function hooks into `SYSTEM:*MODULE-PROVIDER-FUNCTIO`
Each such hook function takes a single argument containing the symbol passed to `CL:REQUIRE` and returns a non-`NIL` value if it can successful resolve the symbol.

4.8 JSS extension of the Reader by SHARPSIGN-DOUBLE-QUOTE

The JSS contrib constitutes an additional, optional extension to the reader in the definition of the `SHARPSIGN-DOUBLE-QUOTE` ("#¨") reader macro. See section 5.3 on page 52 for more information.

[7] This loading can be inhibited if, at runtime, the Java class corresponding ":classname" clause of the system definition is present.

4.9 ASDF

asdf-3.3.4 (see [RBRK]) is packaged as core component of ABCL, but not initialized by default, as it relies on the CLOS subsystem which can take a bit of time to start [8]. The packaged ASDF may be loaded by the ANSI `REQUIRE` mechanism as follows:

```
CL-USER> (require :asdf)
```

4.10 Extension to CL:MAKE-ARRAY

With the `:nio` feature is present[9], the implementation adds two keyword arguments to `cl:make-array`, viz. `:nio-buffer` and `:nio-direct`.

With the `:nio-buffer` keyword, the user is able to pass instances of of `java.nio.ByteBuffer` and its subclasses for the storage of vectors and arrays specialized on the byte-vector types satisfying

```
(or
  (unsigned-byte 8)
  (unsigned-byte 16)
  (unsigned-byte 32))
```

As an example, the following would use the `:nio-buffer` as follows to create a 16 byte vector using the created byte-buffer for storage:

```
(let* ((length 16)
       (byte-buffer (java:jstatic "allocate" "java.nio.ByteBuffer" length)))
  (make-array length :element-type '(unsigned-byte 8) :nio-buffer byte-buffer))
```

This feature is available in CFFI[10] via `CFFI-SYS:MAKE-SHAREABLE-BYTE-VECTOR`[11]

`:nio-buffer NIO-BUFFER`
: Initializes the contents of the new vector or array with the contents of `NIO-BUFFER` which needs to be a reference to a `java-object` of class `java.nio.ByteBuffer`.

`:nio-direct NIO-DIRECT-P`
: When `NIO-DIRECT-P` is non-`nil`, constructs a java.nio.Buffer as a "direct" buffer. The buffers returned by this method typically have somewhat higher allocation and deallocation costs than non-direct buffers. The contents of direct buffers may reside outside of the normal garbage-collected heap, and so their impact upon the memory footprint of an application might not be obvious. It is therefore recommended that direct buffers be allocated primarily for large, long-lived buffers that are subject to the underlying system's native I/O operations. In general it is best to allocate direct buffers only when they yield a measureable gain in program performance.

[8] While this time is "merely" on the order of seconds for contemporary 2011 machines, for applications that need to initialize quickly, for example a web server, this time might be unnecessarily long

[9] Available starting in the Eighth Edition (aka abcl-1.7.0) and indicated by the presence of `:nio` in `cl:*features*`

[10] Available at runtime via Quicklisp

[11] Implemented in https://github.com/cffi/cffi/commit/47136ad9a97c2df98dbcd13a068e14489ced5b03

Chapter 5

Contrib

The ABCL contrib is packaged as a separate jar archive usually named `abcl-contrib.jar` or possibly something like `abcl-contrib-1.8.0.jar`. The contrib jar is not loaded by the implementation by default, and must be first initialized by the `REQUIRE` mechanism before using any specific contrib:

```
CL-USER> (require :abcl-contrib)
```

5.1 abcl-asdf

This contrib enables an additional syntax for ASDF system definition which dynamically loads JVM artifacts such as jar archives via encapsulation of the Maven build tool. The Maven Aether component can also be directly manipulated by the function associated with the `ABCL-ASDF:RESOLVE-DEPENDENCIES` symbol.

When loaded, abcl-asdf adds the following objects to ASDF: `JAR-FILE`, `JAR-DIRECTORY`, `CLASS-FILE-DIRECTORY` and `MVN`, exporting them (and others) as public symbols.

5.1.1 Referencing Maven Artifacts via ASDF

Maven artifacts may be referenced within ASDF system definitions, as the following example references the `log4j-1.4.9.jar` JVM artifact which provides a widely-used abstraction for handling logging systems:

```
;;;; -*- Mode: LISP -*-
(require :asdf)
(in-package :cl-user)

(asdf:defsystem :log4j
   :defsystem-depends-on (abcl-asdf)
   :components ((:mvn "log4j/log4j" :version "1.4.9")))
```

5.1.2 API

We define an API for ABCL-ASDF as consisting of the following ASDF classes:

`JAR-DIRECTORY`, `JAR-FILE`, and `CLASS-FILE-DIRECTORY` for JVM artifacts that have a currently valid pathname representation.

Both the `MVN` and `IRI` classes descend from `ASDF-COMPONENT`, but do not directly have a filesystem location.

For use outside of ASDF system definitions, we currently define one method, `ABCL-ASDF:RESOLVE-DEPENDENCIES` which locates, downloads, caches, and then loads into the currently executing JVM process all recursive dependencies annotated in the Maven pom.xml graph.

5.1.3 Directly Instructing Maven to Download JVM Artifacts

Bypassing ASDF, one can directly issue requests for the Maven artifacts to be downloaded

```
CL-USER> (abcl-asdf:resolve-dependencies "com.google.gwt"
                                         "gwt-user")
WARNING: Using LATEST for unspecified version.
"/Users/evenson/.m2/repository/com/google/gwt/gwt-user/2.4.0-rc1
/gwt-user-2.4.0-rc1.jar:/Users/evenson/.m2/repository/javax/vali
dation/validation-api/1.0.0.GA/validation-api-1.0.0.GA.jar:/User
s/evenson/.m2/repository/javax/validation/validation-api/1.0.0.G
A/validation-api-1.0.0.GA-sources.jar"
```

To actually load the dependency, use the `JAVA:ADD-TO-CLASSPATH` generic function:

```
CL-USER> (java:add-to-classpath
          (abcl-asdf:resolve-dependencies "com.google.gwt"
                                          "gwt-user"))
```

Notice that all recursive dependencies have been located and installed locally from the network as well.

More extensive documentations and examples can be found at http://abcl.org/svn/tags/1.8.0/contrib/abcl-asdf/README.markdown.

5.2 asdf-jar

The asdf-jar contrib provides a system for packaging ASDF systems into jar archives for ABCL. Given a running ABCL image with loadable ASDF systems the code in this package will recursively package all the required source and fasls in a jar archive.

The documentation for this contrib can be found at http://abcl.org/svn/tags/1.8.0/contrib/asdf-jar/README.markdown.

5.3 jss

To one used to the more universal syntax of Lisp pairs upon which the definition of read and compile time macros is quite natural [1], the Java syntax available to the Java programmer may be said to suck. To alleviate this situation, the JSS contrib introduces the `SHARPSIGN-DOUBLE-QUOTE` (`#"`) reader macro, which allows the the specification of the name of invoking function as the first element of the relevant s-expr which tends to be more congruent to how Lisp programmers seem to be wired to think.

While quite useful, we don't expect that the JSS contrib will be the last experiment in wrangling Java from Common Lisp.

5.3.1 JSS usage

Example:

```
CL-USER> (require :abcl-contrib)
==> ("ABCL-CONTRIB")
CL-USER> (require :jss)
```

[1] See Graham's "On Lisp" http://lib.store.yahoo.net/lib/paulgraham/onlisp.pdf.

```
==> ("JSS")
CL-USER) (#"getProperties" 'java.lang.System)
==> #<java.util.Properties {java.runtime.name=Java.... {2FA21ACF}>
CL-USER) (#"propertyNames" (#"getProperties" 'java.lang.System))
==> #<java.util.Hashtable$Enumerator java.util.Has.... {36B4361A}>
```

Some more information on jss can be found in its documentation at http://abcl.org/svn/tags/1.8.0/contrib/jss/README.markdown

5.4 jfli

The contrib contains a pure-Java version of JFLI, apparently a descendant of Rich Hickey's early experimentations with using Java from Common Lisp.

http://abcl.org/svn/tags/1.8.0/contrib/jfli/README.

5.5 abcl-introspect

??

ABCL-INTROSPECT offers more extensive functionality for inspecting the state of the implementation, most notably in integration with SLIME, where the backtrace mechanism is augmented to the point that local variables are inspectable.

A compiled function is an instance of a class - This class has multiple instances if it represents a closure, or a single instance if it represents a non-closed-over function.

The ABCL compiler stores constants that are used in function execution as private java fields. This includes symbols used to invoke function, locally-defined functions (such as via `LABEL` or `flet`) and string and other literal objects. ABCL-INTROSPECT implements a "do what I mean" API for introspecting these constants.

ABCL-INTROSPECT provides access to those internal values, and uses them in at least two ways. First, to annotate locally defined functions with the top-level function they are defined within, and second to search for callers of a give function [2] . This may yield some false positives, such as when a symbol that names a function is also used for some other purpose. It can also have false negatives, as when a function is inlined. Still, it's pretty useful. The second use to to find source locations for frames in the debugger. If the source location for a local function is asked for the location of its 'owner' is instead returns.

In order to record information about local functions, ABCL defines a function-plist, which is for the most part unused, but is used here with set of keys indicating where the local function was defined and in what manner, i.e. as normal local function, as a method function, or as an initarg function. There may be other places functions are stashed away (defstructs come to mind) and this file should be added to to take them into account as they are discovered.

ABCL-INTROSPECT does not depend on JSS, but provides a bit of jss-specific functionality if JSS *is* loaded.

5.5.1 Implementations for CL:DISASSEMBLE

The following ASDF systems packages various external tools that may be selected by the `SYS:CHOOSE-DISASSEMBLER` interface:

1. `objectweb`
2. `jad`
3. `javap`

[2] Since java functions are strings, local fields also have these strings. In the context of looking for callers of a function you can also give a string that names a java method. Same caveat re: false positives.

4. `fernweb`
5. `cfr`
6. `procyon`

To use one of these tools, first load the system via ASDF (and/or Quicklisp), then use the `SYS:CHOOSE-DISASSEMBLER` function to select the keyword that appears in `SYS:*DISASSEMBLERS*`.

```
CL-USER> (require :abcl-contrib)(asdf:load-system :objectweb)
CL-USER> sys:*disassemblers*
((:OBJECTWEB
  . ABCL-INTROSPECT/JVM/TOOLS/OBJECTWEB:DISASSEMBLE-CLASS-BYTES)
 (:SYSTEM-JAVAP . SYSTEM:DISASSEMBLE-CLASS-BYTES))
CL-USER> (sys:choose-disassembler :objectweb)
ABCL-INTROSPECT/JVM/TOOLS/OBJECTWEB:DISASSEMBLE-CLASS-BYTES
CL-USER> (disassemble 'cons)
; // class version 52.0 (52)
; // access flags 0x30
; final class org/armedbear/lisp/Primitives$pf_cons extends org/armedbear/lisp/Prim
{
;
;   // access flags 0x1A
;   private final static INNERCLASS org/armedbear/lisp/Primitives$pf_cons org/armed
;
;   // access flags 0x0
;   <init>()V
;     ALOAD 0
;     GETSTATIC org/armedbear/lisp/Symbol.CONS : Lorg/armedbear/lisp/Symbol;
;     LDC "object-1 object-2"
;     INVOKESPECIAL org/armedbear/lisp/Primitive.<init> (Lorg/armedbear/lisp/Symbol
;     RETURN
;     MAXSTACK = 3
;     MAXLOCALS = 1
;
;   // access flags 0x1
;   public execute(Lorg/armedbear/lisp/LispObject;Lorg/armedbear/lisp/LispObject;)L
;     NEW org/armedbear/lisp/Cons
;     DUP
;     ALOAD 1
;     ALOAD 2
;     INVOKESPECIAL org/armedbear/lisp/Cons.<init> (Lorg/armedbear/lisp/LispObject;
;     ARETURN
;     MAXSTACK = 4
;     MAXLOCALS = 3
; }
NIL
```

http://abcl.org/svn/tags/1.8.0/contrib/abcl-introspect/.

5.6 abcl-build

ABCL-BUILD constitutes a new implementation for the original Lisp-hosted ABCL build system API in the package `ABCL-BUILD` that uses the same build artifacts as all of the other current builds.

5.6.1 Utilities

ABCL-BUILD consolidates various utilities that are useful for system construction, namely

- The ability to introspect the invocation of given executable in the current implementation process PATH.
- Downloading and unpackaging selected JVM artifacts, namely the Ant and Maven build tools. The `ABCL-BUILD:WITH-ANT` and `ABCL-BUILD:WITH-MVN` macros abstracts this installation procedure conveniently away from the User.
- The beginnings of a generic framework to download arbitrary archives from the network.

`http://abcl.org/svn/tags/1.8.0/contrib/abcl-build/`.

5.7 named-readtables

`NAMED-READTABLES` is a library that provides a namespace for readtables akin to the already-existing namespace of packages.

Originally from `https://github.com/melisgl/named-readtables/`.

See `http://abcl.org/svn/tags/1.8.0/contrib/named-readtables/` for more information.

Chapter 6

History

ABCL was originally the extension language for the J editor, which was started in 1998 by Peter Graves. Sometime in 2003, a whole lot of code that had previously not been released publicly was suddenly committed that enabled ABCL to be plausibly termed an emergent ANSI Common Lisp implementation candidate.

From 2006 to 2008, Peter manned the development lists, incorporating patches as made sense. After a suitable search, Peter nominated Erik Hülsmann to take over the project.

In 2008, the implementation was transferred to the current maintainers, who have strived to improve its usability as a contemporary Common Lisp implementation.

On October 22, 2011, with the publication of this Manual explicitly stating the conformance of Armed Bear Common Lisp to ANSI, we released abcl-1.0.0. We released abcl-1.0.1 as a maintenance release on January 10, 2012.

In December 2012, we revised the implementation by adding (A)MOP with the release of abcl-1.1.0. We released abcl-1.1.1 as a maintenance release on February 14, 2013.

At the beginning of June 2013, we enhanced the stability of the implementation with the release of abcl-1.2.1.

In March 2014, we introduced the Fourth Edition of the implementation with abcl-1.3.0. At the end of April 2014, we released abcl-1.3.1 as a maintenance release.

In October 2016 we blessed the current SVN trunk `http://abcl.org/svn/trunk/` as 1.4.0, which includes the community contributions from Vihbu, Olof, Pipping, and Cyrus. We gingerly stepped into current century by establishing GIT bridges to the source repositories avaiable via the URIs `https://github.com/armedbear/abcl/` and `https://gitlab.common-lisp.net/abcl/abcl/` so that pull requests for enhancements to the implementation many be more easily facilitated.

In June 2017, we released ABCL 1.5.0 which dropped support for running upon Java 5.

Against the falling canvas of 2019 we released ABCL 1.6.0 which provided compatibility with Java 11 while skipping Java 9 and 10. In April 2020, we offered abcl-1.6.1 as a maintenance release for usage around ELS2020.

With the overhaul the implementation of arrays specialized on `(or (unsigned-byte 8) (unsigned-byte 16) (unsigned-byte 32))` to using `java.nio.Buffer` objects, we deemed the implementation worthy to bless with release as abcl-1.7.0 in June 2020. We released abcl-1.7.1 as a maintenance release in July 2020.

In accordance with the fashion of packaging Java applications as multiply compressed aritifacts, with abcl-1.8.0 in October 2020 we rendered usage of pathnames capable of accessing zip archives within archives workable across all supported platforms. This release supports the newly released openjdk15 as the most advanced platform baseline.

Appendix A

The MOP Dictionary

— Function: **%defgeneric** [**mop**] *function-name &rest all-keys*
not-documented

— Generic Function: **accessor-method-slot-definition** [**mop**]
not-documented

— Generic Function: **add-dependent** [**mop**]
not-documented

— Generic Function: **add-direct-method** [**mop**]
not-documented

— Generic Function: **add-direct-subclass** [**mop**]
not-documented

— Function: **canonicalize-direct-superclasses** [**mop**] *direct-superclasses*
not-documented

— Generic Function: **class-default-initargs** [**mop**]
not-documented

— Generic Function: **class-direct-default-initargs** [**mop**]
not-documented

— Generic Function: **class-direct-methods** [**mop**]
not-documented

— Generic Function: **class-direct-slots** [**mop**]
not-documented

— Generic Function: **class-direct-subclasses** [**mop**]
not-documented

— Generic Function: **class-direct-superclasses** [**mop**]
not-documented

— Function: **class-documentation** [**mop**]
not-documented

— Generic Function: **class-finalized-p** [**mop**]
not-documented

— Generic Function: **class-precedence-list** [**mop**]
not-documented

— Generic Function: **class-prototype** [**mop**]
not-documented

— Generic Function: **class-slots** [**mop**]
not-documented

— Generic Function: **compute-applicable-methods** [**common-lisp**]
not-documented

— Generic Function: **compute-applicable-methods-using-classes** [**mop**]
not-documented

— Generic Function: **compute-class-precedence-list** [**mop**]
not-documented

— Generic Function: **compute-default-initargs** [**mop**]
not-documented

— Generic Function: **compute-discriminating-function** [**mop**]
not-documented

— Generic Function: **compute-effective-method** [**mop**]
not-documented

— Generic Function: **compute-effective-slot-definition** [**mop**]
not-documented

— Generic Function: **compute-slots** [**mop**]
not-documented

— Class: **direct-slot-definition** [**mop**]
not-documented

— Generic Function: **direct-slot-definition-class** [**mop**]
not-documented

— Class: **effective-slot-definition** [**mop**]
not-documented

— Generic Function: **effective-slot-definition-class** [**mop**]
not-documented

— Function: **ensure-class** [**mop**] *name &rest all-keys &key &allow-other-keys*
not-documented

— Generic Function: **ensure-class-using-class** [**mop**]
not-documented

— Generic Function: **ensure-generic-function-using-class** [**mop**]
not-documented

— Class: **eql-specializer** [**mop**]
not-documented

— Function: **eql-specializer-object** [**mop**] *eql-specializer*
not-documented

— Function: **extract-lambda-list** [**mop**] *specialized-lambda-list*
not-documented

— Function: **extract-specializer-names** [**mop**] *specialized-lambda-list*
not-documented

— Generic Function: **finalize-inheritance** [**mop**]
not-documented

— Generic Function: **find-method-combination** [**mop**]
not-documented

— Class: **forward-referenced-class** [**system**]
not-documented

— Class: **funcallable-standard-class** [**mop**]
not-documented

— Function: **funcallable-standard-instance-access** [**mop**] *instance location*
not-documented

— Class: **funcallable-standard-object** [**mop**]
not-documented

— Generic Function: **generic-function-argument-precedence-order** [**mop**]
not-documented

— Generic Function: **generic-function-declarations** [**mop**]
not-documented

— Generic Function: **generic-function-lambda-list** [**mop**]
not-documented

— Generic Function: **generic-function-method-class** [**mop**]
not-documented

— Generic Function: **generic-function-method-combination** [**mop**]
not-documented

— Generic Function: **generic-function-methods** [**mop**]
not-documented

— Generic Function: **generic-function-name** [**mop**]
not-documented

— Function: **intern-eql-specializer** [**mop**] *object*
not-documented

— Generic Function: **make-method-lambda** [**mop**]
not-documented

— Generic Function: **map-dependents** [**mop**]
not-documented

— Class: **metaobject** [**mop**]
not-documented

— Generic Function: **method-function** [**mop**]
not-documented

— Generic Function: **method-generic-function** [**mop**]
not-documented

— Generic Function: **method-lambda-list** [**mop**]
not-documented

— Generic Function: **method-qualifiers** [**common-lisp**]
not-documented

— Generic Function: **method-specializers** [**mop**]
not-documented

— Generic Function: **reader-method-class** [**mop**]
not-documented

— Generic Function: **remove-dependent** [**mop**]
not-documented

— Generic Function: **remove-direct-method** [**mop**]
not-documented

— Generic Function: **remove-direct-subclass** [**mop**]
not-documented

— Function: **set-funcallable-instance-function** [**mop**] *funcallable-instance function*
not-documented

— Generic Function: **slot-boundp-using-class** [**mop**]
not-documented

— Class: **slot-definition** [**system**]
not-documented

— Generic Function: **slot-definition-allocation** [**mop**]
not-documented

— Generic Function: **slot-definition-documentation** [**mop**]
not-documented

— Generic Function: **slot-definition-initargs** [**mop**]
not-documented

— Generic Function: **slot-definition-initform** [**mop**]
not-documented

— Generic Function: **slot-definition-initfunction** [**mop**]
not-documented

— Generic Function: **slot-definition-location** [**mop**]
not-documented

— Generic Function: **slot-definition-name** [**mop**]
not-documented

— Generic Function: **slot-definition-readers** [**mop**]
not-documented

— Generic Function: **slot-definition-type** [**mop**]
not-documented

— Generic Function: **slot-definition-writers** [**mop**]
not-documented

— Generic Function: **slot-makunbound-using-class** [**mop**]
not-documented

— Generic Function: **slot-value-using-class** [**mop**]
not-documented

— Class: **specializer** [**mop**]
not-documented

— Generic Function: **specializer-direct-generic-functions** [**mop**]
not-documented

— Generic Function: **specializer-direct-methods** [**mop**]
not-documented

— Class: **standard-accessor-method** [**mop**]
not-documented

— Class: **standard-direct-slot-definition** [**mop**]
not-documented

— Class: **standard-effective-slot-definition** [**mop**]
not-documented

— Function: **standard-instance-access** [**system**] *instance location*
not-documented

— Class: **standard-method** [**common-lisp**]
not-documented

— Class: **standard-reader-method** [**mop**]
not-documented

— Class: **standard-slot-definition** [**mop**]
not-documented

— Class: **standard-writer-method** [**mop**]
not-documented

— Generic Function: **update-dependent** [**mop**]
not-documented

— Generic Function: **validate-superclass** [**mop**]
This generic function is called to determine whether the class superclass is suitable for use as a superclass of class.

— Generic Function: **writer-method-class** [**mop**]
not-documented

Appendix B

The SYSTEM Dictionary

The public interfaces in this package are subject to change with ABCL 1.9

— Function: **%allocate-funcallable-instance** [**system**] *class*
not-documented

— Function: **%class-default-initargs** [**system**]
not-documented

— Function: **%class-direct-default-initargs** [**system**]
not-documented

— Function: **%class-direct-methods** [**system**]
not-documented

— Function: **%class-direct-slots** [**system**]
not-documented

— Function: **%class-direct-subclasses** [**system**]
not-documented

— Function: **%class-direct-superclasses** [**system**]
not-documented

— Function: **%class-finalized-p** [**system**]
not-documented

— Function: **%class-layout** [**system**] *class*
not-documented

— Function: **%class-name** [**system**] *class*
not-documented

— Function: **%class-precedence-list** [**system**]
not-documented

— Function: **%class-slots** [**system**] *class*
not-documented

— Function: **%defun** [**system**] *name definition*
not-documented

— Function: **%documentation** [**system**] *object doc-type*
not-documented

— Function: **%float-bits** [**system**] *integer*
not-documented

— Function: **%in-package** [**system**]
not-documented

— Function: **%make-condition** [**system**]
not-documented

— Function: **%make-emf-cache** [**system**]
not-documented

— Function: **%make-instances-obsolete** [**system**] *class*
not-documented

— Function: **%make-integer-type** [**system**] *low high*
not-documented

— Function: **%make-list** [**system**]
not-documented

— Function: **%make-logical-pathname** [**system**] *namestring*
not-documented

— Function: **%make-slot-definition** [**system**] *slot-class*
Argument must be a subclass of standard-slot-definition

— Function: **%make-structure** [**system**] *name slot-values*
not-documented

— Function: **%member** [**system**]
not-documented

— Function: **%nstring-capitalize** [**system**]
not-documented

— Function: **%nstring-downcase** [**system**]
not-documented

— Function: **%nstring-upcase** [**system**]
not-documented

— Function: **%output-object** [**system**] *object stream*
not-documented

— Function: **%putf** [**system**] *plist indicator new-value*
not-documented

— Function: **%reinit-emf-cache** [**system**] *generic-function eql-specilizer-objects-list*
not-documented

— Function: **%set-class-default-initargs** [**system**]
not-documented

— Function: **%set-class-direct-default-initargs** [**system**]
not-documented

— Function: **%set-class-direct-methods** [**system**]
not-documented

— Function: **%set-class-direct-slots** [**system**]
not-documented

— Function: **%set-class-direct-subclasses** [**system**] *class direct-subclasses*
not-documented

— Function: **%set-class-direct-superclasses** [**system**]
not-documented

— Function: **%set-class-documentation** [**system**]
not-documented

— Function: **%set-class-finalized-p** [**system**]
not-documented

— Function: **%set-class-layout** [**system**] *class layout*
not-documented

— Function: **%set-class-name** [**system**]
not-documented

— Function: **%set-class-precedence-list** [**system**]
not-documented

— Function: **%set-class-slots** [**system**] *class slot-definitions*
not-documented

— Function: **%set-documentation** [**system**] *object doc-type documentation*
not-documented

— Function: **%set-fill-pointer** [**system**]
not-documented

— Function: **%set-find-class** [**system**]
not-documented

— Function: **%set-standard-instance-access** [**system**] *instance location new-value*
not-documented

— Function: **%set-std-instance-layout** [**system**]
not-documented

— Function: **%std-allocate-instance** [**system**] *class*
not-documented

— Function: **%stream-output-object** [**system**]
not-documented

— Function: **%stream-terpri** [**system**] *output-stream*
not-documented

— Function: **%stream-write-char** [**system**] *character output-stream*
not-documented

— Function: **%string-capitalize** [**system**]
not-documented

— Function: **%string-downcase** [**system**]
not-documented

— Function: **%string-equal** [**system**]
not-documented

— Function: **%string-greaterp** [**system**]
not-documented

— Function: **%string-lessp** [**system**]
not-documented

— Function: **%string-not-equal** [**system**]
not-documented

— Function: **%string-not-greaterp** [**system**]
not-documented

— Function: **%string-not-lessp** [**system**]
not-documented

— Function: **%string-upcase** [**system**]
not-documented

— Function: **%string/=** [**system**]
not-documented

— Function: **%string<** [**system**]
not-documented

— Function: **%string<=** [**system**]
not-documented

— Function: **%string>** [**system**]

not-documented

— Function: **%string>=** [**system**]

not-documented

— Function: **%type-error** [**system**] *datum expected-type*

not-documented

— Function: **%wild-pathname-p** [**system**] *pathname keyword*

Predicate for determing whether PATHNAME contains wild components. KEYWORD, if non-nil, should be one of :directory, :host, :device, :name, :type, or :version indicating that only the specified component should be checked for wildness.

— Variable: ***abcl-contrib*** [**system**]

Pathname of the abcl-contrib artifact.
Initialized via SYSTEM:FIND-CONTRIB.

— Variable: ***compile-file-class-extension*** [**system**]

not-documented

— Variable: ***compile-file-environment*** [**system**]

not-documented

— Variable: ***compile-file-type*** [**system**]

not-documented

— Variable: ***compile-file-zip*** [**system**]

not-documented

— Variable: ***compiler-diagnostic*** [**system**]

The stream to emit compiler diagnostic messages to, or nil to muffle output.

— Variable: ***compiler-error-context*** [**system**]

not-documented

— Variable: ***current-print-length*** [**system**]

not-documented

— Variable: ***current-print-level*** [**system**]

not-documented

— Variable: ***debug*** [**system**]

not-documented

— Variable: ***disassemblers*** [**system**]

Methods of invoking CL:DISASSEMBLE consisting of a pushable list of (name function), where function takes a object to disassemble, returns the results as a string.

The system is :jad using the venerable-but-still-works JAD.

— Variable: ***enable-autocompile*** [**system**]

not-documented

— Variable: ***explain*** [**system**]

not-documented

— Variable: ***fasl-loader*** [**system**]

not-documented

— Variable: ***fasl-version*** [**system**]

not-documented

— Variable: ***inline-declarations*** [**system**]

not-documented

— Variable: ***logical-pathname-translations*** [**system**]

not-documented

— Variable: ***noinform*** [**system**]

not-documented

— Variable: ***safety*** [**system**]

not-documented

— Variable: ***source*** [**system**]

not-documented

— Variable: ***source-position*** [**system**]

not-documented

— Variable: ***space*** [**system**]

not-documented

— Variable: ***speed*** [**system**]

not-documented

— Variable: ***traced-names*** [**system**]

not-documented

— Variable: +**cl-package**+ [**system**]

not-documented

— Variable: +**false-type**+ [**system**]
not-documented

— Variable: +**fixnum-type**+ [**system**]
not-documented

— Variable: +**integer-type**+ [**system**]
not-documented

— Variable: +**keyword-package**+ [**system**]
not-documented

— Variable: +**slot-unbound**+ [**system**]
not-documented

— Variable: +**true-type**+ [**system**]
not-documented

— Function: **aset** [**system**] *array subscripts new-element*
not-documented

— Function: **autocompile** [**system**] *function*
not-documented

— Function: **available-encodings** [**system**]
Returns all charset encodings suitable for passing to a stream constructor available at runtime.

— Macro: **aver** [**system**]
not-documented

— Function: **backtrace** [**system**]
Returns a Java backtrace of the invoking thread.

— Function: **built-in-function-p** [**system**]
not-documented

— Function: **cache-emf** [**system**] *generic-function args emf*
not-documented

— Function: **call-count** [**system**]
not-documented

— Variable: **call-registers-limit** [**system**]
not-documented

— Function: **canonicalize-logical-host** [**system**] *host*
not-documented

— Function: **check-declaration-type** [**system**] *name*

not-documented

— Function: **check-sequence-bounds** [**system**] *sequence start end*

not-documented

— Function: **choose-disassembler** [**system**] *&optional name*

Hook to choose invoked behavior of CL:DISASSEMBLE by using one of the methods registered in SYSTEM:*DISASSEMBLERS*.

Optionally, prefer the strategy named NAME if one exists.

— Function: **class-bytes** [**system**] *class*

not-documented

— Function: **coerce-to-condition** [**system**] *datum arguments default-type fun-name*

not-documented

— Function: **coerce-to-function** [**system**]

not-documented

— Function: **compile-file-if-needed** [**system**] *input-file &rest allargs &key force-compile &allow-other-keys*

not-documented

— Function: **compile-system** [**extensions**] *&key quit (zip t) (cls-ext *compile-file-class-extension*) (abcl-ext *compile-file-type*) output-path*

not-documented

— Function: **compiled-lisp-function-p** [**system**] *object*

not-documented

— Function: **compiler-defstruct** [**system**] *name &key conc-name default-constructor constructors copier include type named initial-offset predicate print-function print-object direct-slots slots inherited-accessors documentation*

not-documented

— Function: **compiler-error** [**system**] *format-control &rest format-arguments*

not-documented

— Function: **compiler-macroexpand** [**system**] *form &optional env*

not-documented

— Function: **compiler-style-warn** [**system**] *format-control &rest format-arguments*

not-documented

— Function: **compiler-subtypep** [**system**] *compiler-type typespec*

not-documented

— Function: **compiler-unsupported** [**system**] *format-control &rest format-arguments*
not-documented

— Function: **compiler-warn** [**system**] *format-control &rest format-arguments*
not-documented

— Function: **concatenate-fasls** [**system**] *inputs output*
not-documented

— Macro: **defconst** [**system**]
not-documented

— Macro: **define-source-transform** [**system**]
not-documented

— Macro: **defknown** [**system**]
not-documented

— Function: **delete-eq** [**system**] *item sequence*
not-documented

— Function: **delete-eql** [**system**] *item sequence*
not-documented

— Function: **describe-compiler-policy** [**system**]
not-documented

— Function: **disable-zip-cache** [**system**]
Disable all caching of ABCL FASLs and ZIPs.

— Function: **disassemble-class-bytes** [**system**] *java-object*
not-documented

— Function: **double-float-high-bits** [**system**] *float*
not-documented

— Function: **double-float-low-bits** [**system**] *float*
not-documented

— Function: **dump-form** [**system**] *form stream*
not-documented

— Function: **dump-uninterned-symbol-index** [**system**] *symbol*
not-documented

— Function: **empty-environment-p** [**system**] *environment*
not-documented

— Function: **ensure-input-stream** [**system**] *pathname*
Returns a java.io.InputStream for resource denoted by PATHNAME.

— Class: **environment** [**system**]
not-documented

— Function: **environment-add-function-definition** [**system**] *environment name lambda-expression*
not-documented

— Function: **environment-add-macro-definition** [**system**] *environment name expander*
not-documented

— Function: **environment-add-symbol-binding** [**system**] *environment symbol value*
not-documented

— Function: **environment-all-functions** [**system**] *environment*
not-documented

— Function: **environment-all-variables** [**system**] *environment*
not-documented

— Function: **environment-variables** [**system**] *environment*
not-documented

— Function: **expand-inline** [**system**] *form expansion*
not-documented

— Function: **expand-source-transform** [**system**] *form*
not-documented

— Function: **fdefinition-block-name** [**system**] *function-name*
not-documented

— Function: **find-contrib** [**system**]
Introspect runtime classpaths to return a pathname containing subdirectories containing ASDF definitions.

— Function: **find-system** [**system**]
Find the location of the Armed Bear system implementation
Used to determine relative pathname to find 'abcl-contrib.jar'.

— Function: **fixnum-constant-value** [**system**] *compiler-type*
not-documented

— Function: **fixnum-type-p** [**system**] *compiler-type*
not-documented

— Function: **float-infinity-p** [**system**]
not-documented

— Function: **float-nan-p** [**system**]
not-documented

— Function: **float-overflow-mode** [**system**] *&optional boolean*
not-documented

— Function: **float-string** [**system**]
not-documented

— Function: **float-underflow-mode** [**system**] *&optional boolean*
not-documented

— Class: **forward-referenced-class** [**system**]
not-documented

— Function: **frame-to-list** [**system**] *frame*
not-documented

— Function: **frame-to-string** [**system**] *frame*
Convert stack FRAME to a (potentially) readable string.

— Function: **fset** [**system**] *name function &optional source-position arglist documentation*
not-documented

— Function: **ftype-result-type** [**system**] *ftype*
not-documented

— Function: **function-plist** [**system**] *function*
not-documented

— Function: **function-result-type** [**system**] *name*
not-documented

— Function: **get-cached-emf** [**system**] *generic-function args*
not-documented

— Function: **get-function-info-value** [**system**] *name indicator*
not-documented

— Function: **gethash1** [**system**] *key hash-table*
not-documented

— Function: **grovel-java-definitions-in-file** [**system**] *file out*
not-documented

— Function: **hash-table-weakness** [**system**] *hash-table*
Return weakness property of HASH-TABLE, or NIL if it has none.

— Function: **hot-count** [**system**]
not-documented

— Function: **identity-hash-code** [**system**]
not-documented

— Function: **init-fasl** [**system**] *&key version*
not-documented

— Function: **inline-expansion** [**system**] *name*
not-documented

— Function: **inline-p** [**system**] *name*
not-documented

— Function: **inspected-parts** [**system**]
not-documented

— Function: **integer-constant-value** [**system**] *compiler-type*
not-documented

— Function: **integer-type-high** [**system**]
not-documented

— Function: **integer-type-low** [**system**]
not-documented

— Function: **integer-type-p** [**system**] *object*
not-documented

— Function: **interactive-eval** [**system**]
not-documented

— Function: **internal-compiler-error** [**system**] *format-control &rest format-arguments*
not-documented

— Class: **jar-stream** [**system**]
not-documented

— Function: **java-long-type-p** [**system**] *compiler-type*
not-documented

— Function: **java.class.path** [**system**]
Return a list of the directories as pathnames referenced in the JVM class-path.

— Function: **lambda-name** [**system**]
not-documented

— Function: **layout-class** [**system**] *layout*
not-documented

— Function: **layout-length** [**system**] *layout*
not-documented

— Function: **layout-slot-index** [**system**]
not-documented

— Function: **layout-slot-location** [**system**] *layout slot-name*
not-documented

— Function: **list-delete-eq** [**system**] *item list*
not-documented

— Function: **list-delete-eql** [**system**] *item list*
not-documented

— Function: **list-directory** [**system**] *directory &optional (resolve-symlinks nil)*
Lists the contents of DIRECTORY, optionally resolving symbolic links.

— Function: **load-compiled-function** [**system**] *source*
not-documented

— Function: **load-system-file** [**system**]
not-documented

— Function: **logical-host-p** [**system**] *canonical-host*
not-documented

— Function: **logical-pathname-p** [**system**] *object*
Returns true if OBJECT is of type logical-pathname; otherwise, returns false.

— Function: **lookup-known-symbol** [**system**] *symbol*
Returns the name of the field and its class designator which stores the Java object 'symbol'.

— Function: **macro-function-p** [**system**] *value*
not-documented

— Function: **make-closure** [**system**]
not-documented

— Function: **make-compiler-type** [**system**] *typespec*
not-documented

— Function: **make-double-float** [**system**] *bits*
not-documented

— Function: **make-environment** [**system**] *&optional parent-environment*
not-documented

— Function: **make-file-stream** [**system**] *pathname namestring element-type direction if-exists external-format*
not-documented

— Function: **make-fill-pointer-output-stream** [**system**]
not-documented

— Function: **make-integer-type** [**system**] *type*
not-documented

— Function: **make-keyword** [**system**] *symbol*
not-documented

— Function: **make-layout** [**system**] *class instance-slots class-slots*
not-documented

— Function: **make-macro** [**system**] *name expansion-function*
not-documented

— Function: **make-macro-expander** [**system**] *definition*
not-documented

— Function: **make-single-float** [**system**] *bits*
not-documented

— Function: **make-structure** [**system**]
not-documented

— Function: **make-symbol-macro** [**system**] *expansion*
not-documented

— Macro: **named-lambda** [**system**]
not-documented

— Function: **normalize-type** [**system**] *type*
not-documented

— Function: **note-name-defined** [**system**] *name*
not-documented

— Function: **notinline-p** [**system**] *name*
not-documented

— Function: **out-synonym-of** [**system**] *stream-designator*
not-documented

— Function: **output-object** [**system**] *object stream*
not-documented

— Function: **package-external-symbols** [**system**]
not-documented

— Function: **package-inherited-symbols** [**system**]
not-documented

— Function: **package-internal-symbols** [**system**]
not-documented

— Function: **package-symbols** [**system**]
not-documented

— Function: **parse-body** [**system**] *body &optional (doc-string-allowed t)*
not-documented

— Function: **precompile** [**extensions**] *name &optional definition*
not-documented

— Function: **process-alive-p** [**system**] *process*
not-documented

— Function: **process-error** [**system**] *process*
not-documented

— Function: **process-exit-code** [**system**] *instance*
The exit code of a process.

— Function: **process-input** [**system**] *process*
not-documented

— Function: **process-kill** [**system**] *process*
Kills the process.

— Function: **process-optimization-declarations** [**system**] *forms*
not-documented

— Function: **process-output** [**system**] *process*
not-documented

— Function: **process-p** [**system**] *object*
not-documented

— Function: **process-pid** [**system**] *process*
Return the process ID.

— Function: **process-wait** [**system**] *process*
Wait for process to quit running for some reason.

— Function: **proclaimed-ftype** [**system**] *name*
not-documented

— Function: **proclaimed-type** [**system**] *name*
not-documented

— Function: **psxhash** [**system**] *object*
not-documented

— Function: **put** [**system**]
not-documented

— Function: **puthash** [**system**] *key hash-table new-value &optional default*
not-documented

— Function: **read-8-bits** [**system**] *stream &optional eof-error-p eof-value*
not-documented

— Function: **read-vector-unsigned-byte-8** [**system**] *vector stream start end*
not-documented

— Function: **record-source-information** [**system**] *name &optional source-pathname source-position*
not-documented

— Function: **record-source-information-for-type** [**system**] *name type &optional source-pathname source-position*
Record source information on the SYS:SOURCE property for symbol with NAME
TYPE is either a symbol or list.
Source information for functions, methods, and generic functions are represented as lists of the following form:
(:generic-function function-name) (:function function-name) (:method method-name qualifiers specializers)
Where FUNCTION-NAME or METHOD-NAME can be a either be of the form 'symbol or '(setf symbol).
Source information for all other forms have a symbol for TYPE which is one of the following:
:class, :variable, :condition, :constant, :compiler-macro, :macro :package, :structure, :type, :setf-expander, :source-transform
These values follow SBCL'S implemenation in SLIME c.f. <https://github.com/slime/slime/blob/bad2acf

— Function: **remember** [**system**]
not-documented

— Function: **remove-zip-cache-entry** [**system**] *pathname*

not-documented

— Function: **require-type** [**system**] *arg type*

not-documented

— Function: **run-program** [**system**] *program args &key environment (wait t) clear-environment (input stream) (output stream) (error stream) if-input-does-not-exist (if-output-exists error) (if-error-exists error) directory*

Run PROGRAM with ARGS in with ENVIRONMENT variables.

Possibly WAIT for subprocess to exit.

Optionally CLEAR-ENVIRONMENT of the subprocess of any non specified values.

Creates a new process running the the PROGRAM.

ARGS are a list of strings to be passed to the program as arguments.

For no arguments, use nil which means that just the name of the program is passed as arg 0.

Returns a process structure containing the JAVA-OBJECT wrapped Process object, and the PROCESS-INPUT, PROCESS-OUTPUT, and PROCESS-ERROR streams.

c.f. http://download.oracle.com/javase/6/docs/api/java/lang/Process.html

Notes about Unix environments (as in the :environment):

* The ABCL implementation of run-program, like SBCL, Perl and many other programs, copies the Unix environment by default.

* Running Unix programs from a setuid process, or in any other situation where the Unix environment is under the control of someone else, is a mother lode of security problems. If you are contemplating doing this, read about it first. (The Perl community has a lot of good documentation about this and other security issues in script-like programs.

The &key arguments have the following meanings:

:environment An alist of STRINGs (name . value) describing new environment values that replace existing ones.

:clear-environment If non-NIL, the current environment is cleared before the values supplied by :environment are inserted.

:wait If non-NIL, which is the default, wait until the created process finishes. If NIL, continue running Lisp until the program finishes.

:input If T, I/O is inherited from the Java process. If NIL, /dev/null is used (nul on Windows). If a PATHNAME designator other than a stream is supplied, input will be read from that file. If set to :STREAM, a stream will be available via PROCESS-INPUT to read from. Defaults to :STREAM.

:if-input-does-not-exist If :input points to a non-existing file, this may be set to :ERROR in order to signal an error, :CREATE to create and read from an empty file, or NIL to immediately NIL instead of creating the process. Defaults to NIL.

:output If T, I/O is inherited from the Java process. If NIL, /dev/null is used (nul on Windows). If a PATHNAME designator other than a stream is supplied, output will be redirect to that file. If set to :STREAM, a stream will be available via PROCESS-OUTPUT to write to. Defaults to :STREAM.

:if-output-exists If :output points to a non-existing file, this may be set to :ERROR in order to signal an error, :SUPERSEDE to supersede the existing file, :APPEND to append to it instead, or NIL to immediately NIL instead of creating the process. Defaults to :ERROR.

:error Same as :output, but can also be :output, in which case the error stream is redirected to wherever the standard output stream goes. Defaults to :STREAM.

:if-error-exists Same as :if-output-exists, but for the :error target.

:directory If set will become the working directory for the new process, otherwise the working directory will be unchanged from the current Java process. Defaults to NIL.

— Function: **set-call-count** [**system**]
not-documented

— Function: **set-car** [**system**]
not-documented

— Function: **set-cdr** [**system**]
not-documented

— Function: **set-char** [**system**] *string index character*
not-documented

— Function: **set-function-info-value** [**system**] *name indicator value*
not-documented

— Function: **set-hot-count** [**system**]
not-documented

— Function: **set-schar** [**system**] *string index character*
not-documented

— Function: **set-std-slot-value** [**system**] *instance slot-name new-value*
not-documented

— Function: **setf-function-name-p** [**system**] *thing*
not-documented

— Function: **sha256** [**system**] *&rest paths-or-strings*
not-documented

— Function: **shrink-vector** [**system**] *vector new-size*
not-documented

— Function: **simple-format** [**system**] *destination control-string &rest format-arguments*
not-documented

— Function: **simple-search** [**system**] *sequence1 sequence2*
not-documented

— Function: **simple-typep** [**system**]
not-documented

— Function: **single-float-bits** [**system**] *float*
not-documented

— Class: **slot-definition** [**system**]
not-documented

— Function: **source-transform** [**system**] *name*
not-documented

— Function: **standard-instance-access** [**system**] *instance location*
not-documented

— Function: **standard-object-p** [**system**] *object*
not-documented

— Function: **std-instance-class** [**system**]
not-documented

— Function: **std-instance-layout** [**system**]
not-documented

— Function: **std-slot-boundp** [**system**] *instance slot-name*
not-documented

— Function: **std-slot-value** [**system**] *instance slot-name*
not-documented

— Function: **structure-length** [**system**] *instance*
not-documented

— Function: **structure-object-p** [**system**] *object*
not-documented

— Function: **structure-ref** [**system**] *instance index*
not-documented

— Function: **structure-set** [**system**] *instance index new-value*
not-documented

— Function: **subclassp** [**system**] *class*
not-documented

— Function: **svset** [**system**] *simple-vector index new-value*
not-documented

— Function: **swap-slots** [**system**] *instance-1 instance-2*
not-documented

— Function: **symbol-macro-p** [**system**] *value*
not-documented

— Function: **system-artifacts-are-jars-p** [**system**]
not-documented

— Function: **undefined-function-called** [**system**] *name arguments*
not-documented

— Function: **untraced-function** [**system**] *name*
not-documented

— Function: **unzip** [**system**] *pathname &optional directory => unzipped_pathnames*
not-documented

— Class: **url-stream** [**system**]
not-documented

— Function: **vector-delete-eq** [**system**] *item vector*
not-documented

— Function: **vector-delete-eql** [**system**] *item vector*
not-documented

— Function: **whitespacep** [**system**]
not-documented

— Function: **write-8-bits** [**system**] *byte stream*
not-documented

— Function: **write-vector-unsigned-byte-8** [**system**] *vector stream start end*
not-documented

— Function: **zip** [**system**] *pathname pathnames &optional topdir*
Creates a zip archive at PATHNAME whose entries enumerated via the list of PATHNAMES. If the optional TOPDIR argument is specified, the archive will preserve the hierarchy of PATHNAMES relative to TOPDIR. Without TOPDIR, there will be no sub-directories in the archive, i.e. it will be flat.

Appendix C

The JSS Dictionary

These public interfaces are provided by the JSS contrib.

— Variable: ***cl-user-compatibility*** [**jss**]

Whether backwards compatibility with JSS's use of CL-USER has been enabled.

— Variable: ***do-auto-imports*** [**jss**]

Whether to automatically introspect all Java classes on the classpath when JSS is loaded.

— Variable: ***muffle-warnings*** [**jss**]

Attempt to make JSS less chatting about how things are going.

— Function: **classfiles-import** [**jss**] *directory*

Load all Java classes recursively contained under DIRECTORY in the current process.

— Function: **ensure-compatibility** [**jss**]

Ensure backwards compatibility with JSS's use of CL-USER.

— Function: **find-java-class** [**jss**] *name*

not-documented

— Function: **get-java-field** [**jss**] *object field &optional (try-harder *running-in-osgi*)*

Get the value of the FIELD contained in OBJECT. If OBJECT is a symbol it names a dot qualified static FIELD.

— Function: **hashmap-to-hashtable** [**jss**] *hashmap &rest rest &key (keyfun (function identity)) (valfun (function identity)) (invert? NIL) table &allow-other-keys*

Converts the a HASHMAP reference to a java.util.HashMap object to a Lisp hashtable.

The REST paramter specifies arguments to the underlying MAKE-HASH-TABLE call.

KEYFUN and VALFUN specifies functions to be run on the keys and values of the HASHMAP right before they are placed in the hashtable.

If INVERT? is non-nil than reverse the keys and values in the resulting hashtable.

— Macro: **invoke-add-imports** [**jss**]

Push these imports onto the search path. If multiple, earlier in list take precedence

— Function: **invoke-restargs** [**jss**] *method object args &optional (raw? NIL)*

not-documented

— Function: **iterable-to-list** [**jss**] *iterable*

Return the items contained the java.lang.Iterable ITERABLE as a list.

— Function: **j2list** [**jss**] *thing*

Attempt to construct a Lisp list out of a Java THING.

THING may be a wide range of Java collection types, their common iterators or a Java array.

— Function: **japropos** [**jss**] *string*

Output the names of all Java class names loaded in the current process which match STRING..

— Function: **jar-import** [**jss**] *file*

Import all the Java classes contained in the pathname FILE into the JSS dynamic lookup cache.

— Function: **jarray-to-list** [**jss**] *jarray*

Convert the Java array named by JARRARY into a Lisp list.

— Function: **java-class-method-names** [**jss**] *class &optional stream*

Return a list of the public methods encapsulated by the JVM CLASS.

If STREAM non-nil, output a verbose description to the named output stream.

CLASS may either be a string naming a fully qualified JVM class in dot notation, or a symbol resolved against all class entries in the current classpath.

— Function: **jclass-all-interfaces** [**jss**] *class*

Return a list of interfaces the class implements

— Function: **jcmn** [**jss**] *class &optional stream*

Return a list of the public methods encapsulated by the JVM CLASS.

If STREAM non-nil, output a verbose description to the named output stream.

CLASS may either be a string naming a fully qualified JVM class in dot notation, or a symbol resolved against all class entries in the current classpath.

— Function: **jlist-to-list** [**jss**] *list*

Convert a LIST implementing java.util.List to a Lisp list.

— Function: **jmap** [**jss**] *function thing*

Call FUNCTION for every element in the THING. Returns NIL.

THING may be a wide range of Java collection types, their common iterators or a Java array.

In case the THING is a map-like object, FUNCTION will be called with two arguments, key and value.

— Function: **list-to-list** [**jss**] *list*

not-documented

— Function: **new** [**jss**] *class-name &rest args*

Invoke the Java constructor for CLASS-NAME with ARGS.

CLASS-NAME may either be a symbol or a string according to the usual JSS conventions.

— Function: **set-java-field** [**jss**] *object field value &optional (try-harder *running-in-osgi*)*

Set the FIELD of OBJECT to VALUE. If OBJECT is a symbol, it names a dot qualified Java class to look for a static FIELD. If OBJECT is an instance of java:java-object, the associated is used to look up the static FIELD.

— Function: **set-to-list** [**jss**] *set*

Convert the java.util.Set named in SET to a Lisp list.

— Function: **to-hashset** [**jss**] *list*

Convert LIST to the java.util.HashSet contract

— Function: **vector-to-list** [**jss**] *vector*

Return the elements of java.lang.Vector VECTOR as a list.

— Macro: **with-constant-signature** [**jss**]

Expand all references to FNAME-JNAME-PAIRS in BODY into static function calls promising that the same function bound in the FNAME-JNAME-PAIRS will be invoked with the same argument signature.

FNAME-JNAME-PAIRS is a list of (symbol function &optional raw) elements where symbol will be the symbol bound to the method named by the string function. If the optional parameter raw is non-nil, the result will be the raw JVM object, uncoerced by the usual conventions.

Use this macro if you are making a lot of calls and want to avoid the overhead of the dynamic dispatch.

Bibliography

[Bea] Zach Beane. Quicklisp. http://www.quicklisp.org/. Last accessed Jan 25, 2012.

[BLFM05] Tim Berners-Lee, Roy Fielding, and Larry Masinter. Rfc 3986: Uri generic syntax. http://www.ietf.org/rfc/rfc3986.txt, 2005. Last accessed Feb 5, 2012.

[Cos11] Costanza.Pascal. Closer to mop is a compatibility layer that rectifies many of the absent or incorrect clos mop features across a broad range of common lisp implementations. https://github.com/pcostanza/closer-mop, 2011. Last accessed Oct 2, 2016.

[Eve11] Mark Evenson. Unpublished draft of An Implementation and Analysis of Adding IRI to Common Lisp's Pathname. https://github.com/easye/abcl/blob/master/doc/design/pathnames/url-pathnames.mark, 2011. Last accessed Oct 2, 2016.

[Gro06] Mike Grogan. Scripting for the Java platform. Final Draft Specification JSR-223, Sun Microsystems, Inc., 2006. http://jcp.org/aboutJava/communityprocess/final/jsr223/index.html.

[Lea98] Doug Lea. Overview of package util.concurrent release 1.3.4. http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html, 1998. Last accessed Oct 2, 2016.

[Mas00] Brian Maso. A new era for Java protocol handlers. http://docslide.us/documents/java-protocol-handler.html, August 2000. Last accessed Oct 2, 2016.

[P+96] Kent Pitman et al. Common Lisp HyperSpec. http://www.lispworks.com/documentation/HyperSpec/Front/index.htm, 1996. Last accessed Feb 4, 2012.

[RBRK] François-René Rideau, Daniel Barlow, Christopher Rhodes, and Garry King. Asdf. http://common-lisp.net/project/asdf/. Last accessed Feb 5, 2012.

[Rho09] Christophe Rhodes. User-extensible sequences in Common Lisp. In *Proceedings of the 2007 International Lisp Conference*, pages 13:1–13:14. ACM, 2009. Also available at http://doc.gold.ac.uk/~mas01cr/papers/ilc2007/sequences-20070301.pdf.

[sli] SLIME: The Superior Lisp Interaction Mode for Emacs. http://common-lisp.net/project/slime/. Last accessed Feb 4, 2012.

Index