Armed Bear Common Lisp User Manual

Mark Evenson Erik Hülsmann Rudolf Schlatte Alessio Stalla
Ville Voutilainen

Version 1.3.0-rc-0
January 27, 2014

Contents

0.0.1 Preface to the Fourth Edition

ABCL 1.3 now implements an optimized implementation of the LispStack abstraction thanks to Dmitry Nadezhin.

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 Second Edition

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

Chapter 1

Introduction

Armed Bear Common Lisp (ABCL) is an implementation of Common Lisp that runs on the Java Virtual Machine. It compiles Common Lisp to Java 5 bytecode [1], providing 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 9).
- Java code can call Lisp functions and generic functions, either directly (Section 3.2.1, page 11) or via `JSR-223` (Section 3.3, page 13).
- `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.. [2]

ABCL is supported by the Lisp library manager QuickLisp[3] 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.
- 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.[4]

[1] The class file version is "49.0".

[2] 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.

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

[4] 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 device"). A future version of the implementation may return to conformance by using the `HOST` value to reflect the type explicitly.

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 HyperSpec [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` [5], 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 suitable abstraction between ANSI and Gray streams. [6]
- Incomplete documentation (missing docstrings from exported symbols and the draft status of this user manual).

1.2 License

ABCL is licensed under the terms of the GPL v2 of June 1991 with the "classpath-exception" (see the file `COPYING` in the source distribution [7] 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

- Philipp Marek `Thanks for the markup`
- Douglas Miles `Thanks for the whacky IKVM stuff and keeping the flame alive in the dark years.`
- Alan Ruttenberg `Thanks for JSS.`
- and of course *Peter Graves*

[5] Somewhat surprisingly allowed by ANSI

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

[7] See `http://abcl.org/svn/trunk/tags/1.3.0/COPYING`

Chapter 2

Running ABCL

ABCL is packaged as a single jar file usually named either `abcl.jar` or possibly something like `abcl-1.3.0.jar` if using a versioned package on the local filesystem 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/` directory and described later in this document, see Section 6.3 on page 46.

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 parameter values 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. It's 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 resides 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 12).

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) { // the symbol 'NIL' is explicitly named in the Java
                       // namespace at ``Symbol.NIL''
                       // but is always present in the
                       // local namespace in its unadorned form for
                       // the convenience of the User.
    return false;
  }
  return true;
}
```

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 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.

3.3.8 Implementation of Java classes in Lisp

See `JAVA:JNEW-RUNTIME-CLASS` on 4.1.2.

Chapter 4

Implementation Dependent Extensions

As outlined by the CLHS ANSI conformance guidelines, we document the extensions to the Armed Bear Lisp implementation made accessible to the user by virtue of being an exported symbol in the JAVA, THREADS, or EXTENSIONS packages.

4.1 JAVA

4.1.1 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.

4.1.2 Creating a synthetic Java Class at Runtime

See `JAVA:JNEW-RUNTIME-CLASS` on 4.1.2.

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

— Function: **jclass-superclass-p** [**java**] *class-1 class-2*
Returns T if CLASS-1 is a superclass or interface of CLASS-2

— 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: **dump-classpath** [**java**] *&optional classloader*
not-documented

— Function: **ensure-java-object** [**java**] *obj*
Ensures OBJ is wrapped in a JAVA-OBJECT, wrapping it if necessary.

— Function: **jmethod-return-type** [**java**] *method*
Returns the result type (Java class) of the METHOD

— Function: **jfield-name** [**java**] *field*
Returns the name of FIELD as a Lisp string

— 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.

— Function: **jinstance-of-p** [**java**] *obj class*
OBJ is an instance of CLASS (or one of its subclasses)

— 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.

— Macro: **define-java-class** [**java**]
not-documented

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

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

Returns the name of METHOD as a Lisp string

— Function: **get-default-classloader** [**java**]

not-documented

— 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: **get-current-classloader** [**java**]

not-documented

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

not-documented

— 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.

— Function: **jproperty-value** [**java**] *obj prop*

not-documented

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

Returns the type (Java class) of FIELD

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

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 and argument-types are strings or keywords for primitive types (:void, :int, etc.), and function is a Lisp function of minimum arity (1+ (length argument-types)); the instance ('this') is passed in as the first argument.

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

— Function: **jclass-constructors** [**java**] *class*
Returns a vector of constructors for CLASS

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

— Function: **jmethod-params** [**java**] *method*
Returns a vector of parameter types (Java classes) for METHOD

— Function: **jnew** [**java**] *constructor &rest args*
Invokes the Java constructor CONSTRUCTOR with the arguments ARGS.

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

— Function: **jclass-superclass** [**java**] *class*
Returns the superclass of CLASS, or NIL if it hasn't got one

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

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

— Generic Function: **add-to-classpath** [**java**]
not-documented

— Function: **unregister-java-exception** [**java**] *exception-name*
Unregisters the Java Throwable EXCEPTION-NAME previously registered by REGISTER-JAVA-EXCEPTION.

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

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

MEMBER is a public member of its declaring class

— Variable: **+null+** [**java**]

The JVM null object reference.

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

not-documented

— Class: **java-class** [**java**]

not-documented

— Macro: **jmethod-let** [**java**]

not-documented

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

Returns T if CLASS is an array class

— 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: **jarray-ref-raw** [**java**] *java-array &rest indices*

Dereference the Java array JAVA-ARRAY using the given INDICIES. Does not attempt to coerce the result into a Lisp object.

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

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

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

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

— Macro: **chain** [**java**]

not-documented

— 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.

— Class: **java-object** [**java**]

not-documented

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

Returns the vector of interfaces of CLASS

— Variable: **+true+** [**java**]

The JVM primitive value for boolean true.

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

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: **make-classloader** [**java**] *&optional parent*

not-documented

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

MEMBER is a protected member of its declaring class

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

— 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

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

— Function: **describe-java-object** [**java**]
not-documented

— 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: **jconstructor-params** [**java**] *constructor*
Returns a vector of parameter types (Java classes) for CONSTRUCTOR

— Function: **jmember-static-p** [**java**] *member*
MEMBER is a static member of its declaring class

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

— Function: **jarray-length** [**java**] *java-array*
not-documented

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

— Function: **jclass-field** [**java**] *class field-name*
Returns the field named FIELD-NAME of CLASS

— Generic Function: **jmake-proxy** [**java**]
not-documented

— 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.

— Variable: **+false+** [**java**]
The JVM primitive value for boolean false.

— Function: **jclass-interface-p** [**java**] *class*
Returns T if CLASS is an interface

4.2 THREADS

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.

— Function: **mailbox-empty-p** [**threads**] *mailbox*
Returns non-NIL if the mailbox can be read from, NIL otherwise.

— Function: **threadp** [**threads**]
not-documented

— Function: **destroy-thread** [**threads**]
not-documented

— Macro: **with-mutex** [**threads**]
not-documented

— Function: **thread-join** [**threads**] *thread*
Waits for thread to finish.

— Function: **release-mutex** [**threads**] *mutex*
Releases a lock on the 'mutex'.

— Function: **object-wait** [**threads**]
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: **object-notify-all** [**threads**]
not-documented

— Function: **make-mailbox** [**threads**] *&key ((queue g2304220) NIL)*
not-documented

— Function: **object-notify** [**threads**] *object*
not-documented

— Function: **get-mutex** [**threads**] *mutex*
Acquires a lock on the 'mutex'.

— 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 if the first is NIL.

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: **thread-alive-p** [**threads**] *thread*
Boolean predicate whether THREAD is alive.

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

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

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

not-documented

— 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: **make-mutex** [**threads**] *&key ((in-use g2304421) NIL)*

not-documented

— Class: **thread** [**threads**]

not-documented

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

not-documented

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

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

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

not-documented

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

not-documented

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

not-documented

4.3 EXTENSIONS

The symbols in the EXTENSIONS package (nicknamed "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`.

— Function: **compile-file-if-needed** [**extensions**] *input-file &rest allargs &key force-compile &allow-other-keys*
not-documented

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

— Function: **dump-java-stack** [**extensions**]
not-documented

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

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

— Function: **grovel-java-definitions** [**extensions**]
not-documented

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

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

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

— Function: **process-kill** [**extensions**] *process*
Kills the process.

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

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

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

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

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

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

— Special Operator: **truly-the** [**extensions**]
not-documented

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

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

— Function: **make-socket** [**extensions**] *host port*
not-documented

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

— Function: **process-input** [**extensions**]
not-documented

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

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

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

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

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

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

— Function: **simple-search** [**extensions**] *sequence1 sequence2*
not-documented

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

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

— Function: **file-directory-p** [**extensions**]
not-documented

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

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

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

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

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

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

— Function: **process-error** [**extensions**]
not-documented

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

— Function: **process-alive-p** [**extensions**] *process*
Return t if process is still alive, nil otherwise.

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

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

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

— Function: **char-to-utf8** [**extensions**]
not-documented

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

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

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

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

— Function: **socket-close** [**extensions**] *socket*
not-documented

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

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

— Function: **compile-system** [**extensions**] *&key quit (zip t) output-path*
not-documented

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

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

— Function: **socket-accept** [**extensions**] *socket*
not-documented

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

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

— Function: **autoloadp** [**extensions**] *symbol*
not-documented

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

— Function: **resolve** [**extensions**] *symbol*
not-documented

— Function: **cancel-finalization** [**extensions**] *object*
not-documented

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

— Function: **run-program** [**extensions**] *program args &key environment (wait t)*

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 the new environment. The default is to copy the environment of the current process.

:wait If non-NIL, which is the default, wait until the created process finishes. If NIL, continue running Lisp until the program finishes.

— Function: **fixnump** [**extensions**]

not-documented

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

not-documented

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

not-documented

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

not-documented

— Class: **jar-pathname** [**extensions**]

not-documented

NIL

— Function: **simple-string-search** [**extensions**]

not-documented

— Function: **assql** [**extensions**]

not-documented

— Function: **finalize** [**extensions**] *object function*

not-documented

— Function: **run-shell-command** [**extensions**] *command &key directory (output *standard-output*)*

not-documented

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

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

— Macro: **collect** [**extensions**]
not-documented

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

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

— Function: **charpos** [**extensions**] *stream*
not-documented

— Function: **make-temp-file** [**extensions**]
not-documented

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

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

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

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

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

— Function: **server-socket-close** [**extensions**] *socket*
not-documented

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

— Function: **get-floating-point-modes** [**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.

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

— Function: **featurep** [**extensions**] *form*
not-documented

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

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

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

— Function: **make-server-socket** [**extensions**] *port*
not-documented

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

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

— Function: **init-gui** [**extensions**]
Dummy function used to autoload this file

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

— Function: **process-exit-code** [**extensions**] *instance*
The exit code of a process.

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

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

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

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

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

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

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

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

— Function: **process-wait** [**extensions**] *process*
Wait for process to quit running for some reason.

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

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

— Function: **process-p** [**extensions**] *object*
not-documented

— Variable: ***gui-backend*** [**extensions**]
not-documented

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

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

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

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

— 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: **process-output** [**extensions**]
not-documented

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

— Class: **compiler-unsupported-feature-error** [**extensions**]
not-documented

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

— Function: **compiler-error** [**extensions**] *format-control &rest format-arguments*
not-documented

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

Chapter 5

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.

5.1 Compiler to Java 5 Bytecode

The `CL:COMPILE-FILE` interface emits a packed fasl format whose Pathname has the type "abcl". These fasls are operating system neutral byte archives packaged by the zip compression format which contain artifacts whose loading `CL:LOAD` understands.

5.2 Pathname

We implement an extension to the `CL:PATHNAME` that allows for the description and retrieval of resources named in a URI [1] scheme that the JVM "understands". By definition, support is built-in into the JVM to access the "http" and "https" schemes but additional 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 ext:   <http://abcl.not.org/cl-packages/extensions/> .
@prefix cl:    <http://abcl.not.org/cl-packages/common-lisp/> .

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

Both the `EXT:URL-PATHNAME` and `EXT:JAR-PATHNAME` objects may be used anywhere a `CL:PATHNAME` is accepted with the following caveats:

- A stream obtained via `CL:OPEN` on a `CL:URL-PATHNAME` cannot be the target of write operations.

[1] A URI is essentially a superset of what is commonly understood as a URL We sometime suse 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].

- Any results of canonicalization procedures performed on the underlying URI are discarded between resolutions (i.e. the implementation does not attempt to cache the results of current name resolution of the representing 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[2].

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 as the single form:

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

will load and execute the Quicklisp setup code.

The implementation currently breaks ANSI conformance by allowing the types able to be `CL:READ` for the `DEVICE` to return a possible `CONS` of `CL:PATHNAME` objects.

In order to "smooth over" the bit about types being `CL:READ` from `CL:PATHNAME` components, 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 5.

Implementation

The implementation of these extensions stores all the additional information in the `CL:PATHNAME` object itself in ways that while strictly speaking are conformant, nonetheless may trip up libraries that don't expect the following:

- `DEVICE` can be either a string denoting a drive letter under DOS or a list of exactly one or two elements. If `DEVICE` is a list, it denotes a `EXT:JAR-PATHNAME`, with the entries containing `CL:PATHNAME` objects which describe the outer and (possibly inner) locations of the jar archive [3].

- 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

- 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`.

[2] See [**?**] for the draft of the publication of the technical details

[3] The case of inner and outer `EXT:JAR-PATHNAME` **??** arises when zip archives themselves contain zip archives which is the case when the ABCL fasl is included in the abcl.jar zip archive.

5.3 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.[4]

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.
>
> 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.

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

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.

5.4 Extensible Sequences

See Rhodes2007 [Rho09] for the design.

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).

5.5 Extensions to CLOS

5.5.1 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`.

5.5.2 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)
  ;;; ...
)
```

5.6 Extensions to the Reader

We implement a special hexadecimal escape sequence for specifying 32 bit characters to the Lisp reader[5], namely we allow a sequences of the form `#\Uxxxx` 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 this sequence is never output by the implementation. Instead, the corresponding Unicode character is output for characters whose code is greater than 0x00ff.

5.7 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:

`ASDF` Loads the ASDF implementation shipped with the implementation. After ASDF has been loaded in this manner, 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 toplevel contents of "abcl-contrib.jar" into the ASDF central registry.

1. `abcl-asdf` Functions for loading JVM artifacts dynamically, hooking into ASDF 3 objects where possible.
2. `asdf-jar` Package addressable JVM artifacts via `abcl-asdf` descriptions as a single binary artifact including recursive dependencies.
3. `mvn` These systems name common JVM artifacts from the distributed pom.xml graph of Maven Aether:
   (a) `jna` Dynamically load 'jna.jar' version 4.0.0 from the network [6]
4. `quicklisp-abcl` Boot a local Quicklisp installation via the ASDF:IRI type introduced bia ABCL-ASDF.

   ```
   CL-USER> (asdf:load-system :quicklisp-abcl :force t)
   ```

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.

[5] This represents a compromise with contemporary in 2011 32bit hosting architecures 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

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

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

The JSS contrib consitutes an additional, optional extension to the reader in the definition of the `SHARPSIGN-DOUBLE-QUOTE` ("#¨") reader macro. See section 6.3 on page 46 for more information.

5.9 ASDF

asdf-3.1.0.49 (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 [7]. The packaged ASDF may be loaded by the ANSI `REQUIRE` mechanism as follows:

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

[7] 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

Chapter 6

Contrib

The ABCL contrib is packaged as a separate jar archive usually named `abcl-contrib.jar` or possibly something like `abcl-contrib-1.3.0.jar`. The contrib jar is not loaded by the implementation by default, and must be first intialized by the `REQUIRE` mechanism before using any specific contrib:

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

6.1 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.

6.1.1 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 -*-
(in-package :asdf)

(defsystem :log4j
  :components ((:mvn "log4j/log4j" :version "1.4.9")))
```

6.1.2 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.

6.1.3 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/1.3.0/contrib/abcl-asdf/README.markdown.

6.2 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/1.3.0/abcl-contrib/asdf-jar/README.markdown.

6.3 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.

6.3.1 JSS usage

Example:

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

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

Some more information on jss can be found in its documentation at http://abcl.org/svn/tags/1.3.0/contrib/jss/README.markdown

6.4 jfli

The contrib contains a pure-Java version of JFLI.
http://abcl.org/svn/tags/1.3.0/contrib/jfli/README.

6.5 asdf-install

The asdf-install contrib provides an implementation of ASDF-INSTALL. Superseded by Quicklisp (see Xach2011 [Bea]).

The `require` of the `asdf-install` symbol has the side effect of pushing the directory `~/.asdf-install-dir/systems/` into the value of the ASDF central registry in `asdf:*central-registry*`, providing a convenient mechanism for stashing ABCL specific system definitions for convenient access.
http://abcl.org/tags/1.3.0/contrib/asdf-install/README.

Chapter 7

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 publically 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 maintainence 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 maintainence release on Feburary 14, 2013.

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

In January 2014, we introduced the Fourth Edition of the implementation with abcl-1.3.0.

Appendix A

The MOP Dictionary

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

— Generic Function: **method-qualifiers** [**common-lisp**]
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: **slot-definition-type** [**mop**]
not-documented

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Appendix B

The SYSTEM Dictionary

The public interfaces in this package are subject to change with ABCL 1.4.

— Function: **logical-pathname-p** [**system**] *object*
Returns true if OBJECT is of type logical-pathname; otherwise, returns false.

— Function: **%slot-definition-readers** [**system**] *slot-definition*
not-documented

— Function: **compiler-macroexpand** [**system**] *form &optional env*
not-documented

— Function: **set-generic-function-initial-methods** [**system**]
not-documented

— Function: **set-slot-definition-readers** [**system**] *slot-definition readers*
not-documented

— Function: **set-slot-definition-initform** [**system**] *slot-definition initform*
not-documented

— Function: **generic-function-documentation** [**system**]
not-documented

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

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

— Function: **environment-add-macro-definition** [**system**] *environment name expander*
not-documented

— Function: **lambda-name** [**system**]
not-documented

— Function: **std-slot-value** [**system**] *instance slot-name*
not-documented

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

— Function: **make-slot-definition** [**system**] *&optional class*
Cannot be called with user-defined subclasses of standard-slot-definition.

— Function: **write-vector-unsigned-byte-8** [**system**] *vector stream start end*
not-documented

— Function: **notinline-p** [**system**] *name*
not-documented

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

— Function: **package-symbols** [**system**]
not-documented

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

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

— Function: **compile-file-if-needed** [**system**] *input-file &rest allargs &key force-compile &allow-other-keys*
not-documented

— Function: **puthash** [**system**] *key hash-table new-value &optional default*
not-documented

— Function: **structure-set** [**system**] *instance index new-value*
not-documented

— Function: **make-layout** [**system**] *class instance-slots class-slots*
not-documented

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

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

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

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

— Function: **process-kill** [**system**] *process*
not-documented

— Function: **layout-length** [**system**] *layout*
not-documented

— Class: **environment** [**system**]
not-documented

— Function: **%generic-function-method-class** [**system**]
not-documented

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

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

— Variable: ***compile-file-type*** [**system**]
not-documented

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

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

— Function: **gf-required-args** [**system**]
not-documented

— Function: **set-schar** [**system**] *string index character*
not-documented

— Function: **make-fill-pointer-output-stream** [**system**]
not-documented

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

— Function: **inline-expansion** [**system**] *name*
not-documented

— Variable: ***source-position*** [**system**]
not-documented

— Function: **symbol-macro-p** [**system**] *value*
not-documented

— Function: **double-float-high-bits** [**system**] *float*
not-documented

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

— Function: **fdefinition-block-name** [**system**] *function-name*
not-documented

— Function: **expand-inline** [**system**] *form expansion*
not-documented

— Function: **function-result-type** [**system**] *name*
not-documented

— Function: **environment-add-symbol-binding** [**system**] *environment symbol value*
not-documented

— Function: **swap-slots** [**system**] *instance-1 instance-2*
not-documented

— Function: **%generic-function-methods** [**system**]
not-documented

— Variable: ***speed*** [**system**]
not-documented

— Function: **identity-hash-code** [**system**]
not-documented

— Function: **dump-form** [**system**] *form stream*
not-documented

— Function: **fixnum-type-p** [**system**] *compiler-type*
not-documented

— Function: **make-keyword** [**system**] *symbol*
not-documented

— Function: **make-environment** [**system**] *&optional parent-environment*
not-documented

— Function: **compiled-lisp-function-p** [**system**] *object*
not-documented

— Function: **list-directory** [**system**] *directory &optional (resolve-symlinks t)*
Lists the contents of DIRECTORY, optionally resolving symbolic links.

— Function: **make-file-stream** [**system**] *pathname namestring element-type direction if-exists external-format*
not-documented

— Function: **compiler-subtypep** [**system**] *compiler-type typespec*
not-documented

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

— Function: **check-sequence-bounds** [**system**] *sequence start end*
not-documented

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

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

— Function: **set-car** [**system**]
not-documented

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

— Function: **set-generic-function-argument-precedence-order** [**system**]
not-documented

— Function: **set-slot-definition-documentation** [**system**] *slot-definition documentation*
not-documented

— Variable: ***safety*** [**system**]
not-documented

— Function: **empty-environment-p** [**system**] *environment*
not-documented
NIL

— Function: **set-generic-function-documentation** [**system**]
not-documented

— Function: **%slot-definition-location** [**system**] *slot-definition*
not-documented

— Function: **remove-zip-cache-entry** [**system**] *pathname*
not-documented

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

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

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

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

— Function: **ensure-input-stream** [**system**] *pathname*
Returns a java.io.InputStream for resource denoted by PATHNAME.

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

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

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

— Function: **process-input** [**system**]
not-documented

— Function: **set-slot-definition-initargs** [**system**] *slot-definition initargs*
not-documented

— Function: **make-structure** [**system**]
not-documented

— Function: **%slot-definition-allocation-class** [**system**] *slot-definition*
not-documented

— Variable: ***inline-declarations*** [**system**]
not-documented

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

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

— Function: **set-function-info-value** [**system**] *name indicator value*
not-documented

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

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

— Function: **sha256** [**system**] *&rest paths-or-strings*
not-documented

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

— Function: **std-slot-boundp** [**system**] *instance slot-name*
not-documented

— Function: **set-generic-function-method-class** [**system**]
not-documented

— Function: **%generic-function-method-combination** [**system**]
not-documented

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

— Function: **%slot-definition-allocation** [**system**] *slot-definition*
not-documented

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

— Function: **double-float-low-bits** [**system**] *float*
not-documented

— Function: **simple-search** [**system**] *sequence1 sequence2*
not-documented

— Function: **float-infinity-p** [**system**]
not-documented

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

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

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

— Class: **process** [**system**]
not-documented

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

— Function: **%type-error** [**system**] *datum expected-type*
not-documented

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

— Function: **%finalize-generic-function** [**system**] *generic-function*
not-documented

NIL

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

— Variable: ***compile-file-environment*** [**system**]
not-documented

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

— Function: **set-slot-definition-writers** [**system**] *slot-definition writers*
not-documented

— Function: **%init-eql-specializations** [**system**] *generic-function eql-specilizer-objects-list*
not-documented

— Function: **set-slot-definition-type** [**system**] *slot-definition type*
not-documented

— Function: **out-synonym-of** [**system**] *stream-designator*
not-documented

— Function: **note-name-defined** [**system**] *name*
not-documented

— Function: **integer-type-p** [**system**] *object*
not-documented

— Function: **structure-length** [**system**] *instance*
not-documented

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

— Function: **hash-table-weakness** [**system**] *hash-table*
Return weakness property of HASH-TABLE, or NIL if it has none.

— Function: **float-overflow-mode** [**system**] *&optional boolean*
not-documented

— Function: **process-error** [**system**]
not-documented

— Function: **process-alive-p** [**system**] *process*
Return t if process is still alive, nil otherwise.

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

— Function: **make-double-float** [**system**] *bits*
not-documented

— Function: **set-generic-function-methods** [**system**]
not-documented

— Function: **layout-slot-index** [**system**]
not-documented

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

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

— Function: **interactive-eval** [**system**]
not-documented

— Class: **jar-stream** [**system**]
not-documented

— Function: **%slot-definition-name** [**system**] *slot-definition*
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.

— Function: **%slot-definition-writers** [**system**] *slot-definition*
not-documented

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

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

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

— Function: **shrink-vector** [**system**] *vector new-size*
not-documented

— Function: **package-inherited-symbols** [**system**]
not-documented

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

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

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

— Function: **require-type** [**system**] *arg type*
not-documented

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

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

— Function: **setf-function-name-p** [**system**] *thing*
not-documented

— Variable: ***compiler-error-context*** [**system**]
not-documented

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

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

Appendix C

The JSS Dictionary

These public interfaces are provided by the JSS contrib.

— 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.

— Function: **find-java-class** [**jss**] *name*

not-documented

— Macro: **invoke-add-imports** [**jss**]

not-documented

NIL

— 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.

— Variable: ***do-auto-imports*** [**jss**]

Whether to automatically introspect all Java classes on the classpath when JSS is loaded.

— 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: **list-to-list** [**jss**] *list*

not-documented

— Function: **jarray-to-list** [**jss**] *jarray*

Convert the Java array named by JARRARY into a Lisp list.

— Function: **set-to-list** [**jss**] *set*

not-documented

— 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: **jclass-all-interfaces** [**jss**] *class*
Return a list of interfaces the class implements

— Macro: **with-constant-signature** [**jss**]
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: **jlist-to-list** [**jss**] *list*
Convert a LIST implementing java.util.List to a Lisp list.

— Function: **iterable-to-list** [**jss**] *iterable*
Return the items contained the java.lang.Iterable ITERABLE as a list.

— Variable: ***cl-user-compatibility*** [**jss**]
Whether backwards compatibility with JSS's use of CL-USER has been enabled.

— Function: **jcmn** [**jss**]
not-documented

— Function: **classfiles-import** [**jss**] *directory*
Load all Java classes recursively contained under DIRECTORY in the current process.

— Function: **japropos** [**jss**] *string*
Output the names of all Java class names loaded in the current process which match STRING..

— Function: **vector-to-list** [**jss**] *vector*
not-documented
NIL

— Function: **invoke-restargs** [**jss**] *method object args &optional (raw? NIL)*
not-documented

— Function: **jar-import** [**jss**] *file*
Import all the Java classes contained in the pathname FILE into the JSS dynamic lookup cache.

— Function: **ensure-compatibility** [**jss**]
Ensure backwards compatibility with JSS's use of CL-USER.

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.

[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.

[Mas00] Brian Maso. A new era for Java protocol handlers. http://java.sun.com/developer/onlineTraining/protocolhandlers/, August 2000. Last accessed Jan 25, 2012.

[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