STELLA Manual

Painless symbolic programming with delivery in Common-Lisp, C++ and Java

Version: 1.24

This manual describes STELLA 3.4 or later.

The STELLA Development Team
Hans Chalupsky
Robert M. MacGregor
Thomas A. Russ
{hans,tar}@isi.edu

USC Information Sciences Institute
Table of Contents

1 Introduction .............................................. 1
   1.1 Credits and History .................................. 2

2 Installation .............................................. 3
   2.1 System Requirements .................................. 3
   2.2 Unpacking the Sources ................................. 4
   2.3 Lisp Installation ...................................... 4
   2.4 C++ Installation ...................................... 4
   2.5 Java Installation ..................................... 5
   2.6 X/Emacs Setup ........................................ 6
   2.7 Removing Unneeded Files ............................... 6

3 The STELLA Language ................................. 7
   3.1 Language Overview .................................... 7
   3.2 Basic Data Types (tbw) ................................. 8
   3.3 Control Structure (tbc) ............................... 8
      3.3.1 Conditionals ...................................... 8
   3.4 Functions (tbw) ....................................... 11
   3.5 Classes (tbw) ......................................... 11
   3.6 Methods (tbw) ......................................... 11
   3.7 Macros (tbw) .......................................... 11
   3.8 Modules (tbw) ......................................... 11

4 Programming in STELLA ......................... 12
   4.1 Hello World in STELLA ................................. 12
      4.1.1 Hello World in Lisp ............................... 12
      4.1.2 Hello World in C++ ............................... 14
      4.1.3 Hello World in Java .............................. 16
   4.2 Incrementally Developing STELLA Code ................ 17
   4.3 Performance Hints .................................... 21
      4.3.1 Lisp Performance Hints .......................... 23

5 Library Classes (tbw) ............................... 25
6 Library Functions ......................... 26
   6.1 Basic Constants and Predicates ............ 26
   6.2 Numbers .................................... 27
   6.3 Characters .................................. 30
   6.4 Strings ...................................... 31
   6.5 CONS Lists and Trees ....................... 36
      6.5.1 CONS Lists as Sets ..................... 41
   6.6 Lists ....................................... 41
      6.6.1 Lists as Sets ......................... 44
   6.7 Property and Key-Value Lists ............... 46
   6.8 Vectors ..................................... 49
   6.9 Hash Tables .................................. 51
   6.10 Key Value Maps .............................. 56
   6.11 Hash Sets ................................... 57
   6.12 Iterators ................................... 59
   6.13 Symbols .................................... 60
   6.14 Context and Modules ....................... 62
   6.15 Input and Output ............................ 65
   6.16 Files ....................................... 66
   6.17 Dates and Times ............................. 68
   6.18 XML Support ................................ 71
   6.19 Miscellaneous ............................... 74

Function Index ............................... 115

Variable Index ............................... 116

Concept Index ............................... 117
1 Introduction

This document describes the STELLA programming language. STELLA stands for Strongly-TypEd, Lisp-like LAnguage. It is an object-oriented language that strongly supports symbolic programming tasks. We developed it, since none of the currently “healthy” languages such as C++ or Java adequately support symbolic programming. While Common-Lisp would probably still be today’s language of choice for many symbolic programming applications, its dwindling vendor support and user base make it more and more difficult to justify its use.

When we started the development of the PowerLoom knowledge representation system in 1995 we were faced with exactly this problem. PowerLoom had to be delivered in C++, but it was simply inconceivable to write such a large symbolic programming application directly in C++. The solution was to invent a new programming language we called STELLA and write PowerLoom in STELLA instead.

STELLA is a strongly typed, object-oriented, Lisp-like language specifically geared to support artificial intelligence applications. STELLA preserves those features of Common Lisp deemed essential for symbolic programming such as built-in support for dynamic data structures, heterogeneous collections, first-class symbols, powerful iteration constructs, name spaces, an object-oriented type system with a simple meta-object protocol, exception handling, language extensibility through macros and automatic memory management. Maybe the biggest difference between STELLA and Common Lisp is that STELLA is strongly typed. All externally visible interfaces such as slots, function parameters and return values, etc. have to be explicitly typed. Internal objects such as local variables, however, are mostly typed implicitly supported by type inference. This in conjunction with a powerful type coercion mechanism significantly reduces the number of explicit type information that needs to be supplied by the programmer compared to languages such as C++ or Java.

STELLA programs are first translated into a target language such as Common Lisp, C++ or Java, and then compiled with the native target language compiler to generate executable code. The language constructs of STELLA are restricted to those that can be translated fairly directly into native constructs of the intended target languages. This allows STELLA to be translated into efficient, conventional and readable Lisp, C++ and Java code. The resulting native code can be understood and to some extent even maintained by programmers who don’t know STELLA, and it can easily be interfaced with other programs not written in STELLA.

As of Fall 2000, we have programmed approximately 100,000 lines of STELLA code - about 50% for the STELLA kernel itself and the other 50% for the PowerLoom knowledge representation system and related systems. Our subjective experience has been that it is only slightly more difficult to write and debug a STELLA program than a Lisp program, and that the inconvenience of having to supply some type information is much outweighed by the benefits such as catching many errors during compile time instead of at run time.

The biggest benefit, however, seems to be that we can still leverage all the incremental code development benefits of Lisp, since we use the Common Lisp-based version of STELLA for prototyping. This allows us to incrementally define and redefine functions, methods and classes and to inspect, debug and fix incorrect code on the fly. Even the most sophisticated C++ or Java IDE’s don’t yet seem to support this fully incremental development style, i.e.,
a change in a class (every change in Java is a change to a class) still requires recompilation and restart of the application. But it is the restart that can be the most time consuming if one debugs a complex application that takes a significant time to reach a certain state!

Once a STELLA program has matured, it can be translated into C++ or Java to gain extra efficiency, to deliver it as a stand-alone application, or to link it with other programs.

1.1 Credits and History

Bob MacGregor invented STELLA in 1995 to implement the PowerLoom knowledge representation system. He wrote most of the first Lisp-based kernel system of STELLA and still occasionally writes extensions or provides fixes. Today he is primarily a STELLA user writing his own applications.

Hans Chalupsky completed the first full STELLA bootstrap (STELLA translating itself) in Spring 1996, and then went on to deal with all the changes necessary to handle the many C++ and Java idiosyncrasies that were discovered when the first versions of these translators came online. He is currently one of the principal maintainers of STELLA supporting the STELLA code analyzer and the Lisp and C++ translators.

Eric Melz wrote the first version of the C++ translator under very trying circumstances (i.e., at a stage where the STELLA language changed under him on a daily basis). He got the first C++ version of STELLA running in the Fall of 1996.

Tom Russ wrote the Java translator and got the first Java version of STELLA running in Spring 1999. He is currently one of the principal maintainers of STELLA supporting the STELLA code analyzer and the Lisp and Java translators. He is also still active writing occasional extensions such as the STELLA XML parser.
Chapter 2: Installation

2 Installation

2.1 System Requirements

To install and use STELLA you’ll approximately need the following amounts of disk space:

- 8 MB for the tar-red or zip-ped archive file
- 35 MB for the untarred sources, translations, compiled Java files and documentation
- 8 MB to compile a Lisp version
- 11 MB to compile the C++ version (without -g)
- 3 MB to compile the Java version (already included)

This means that you will need approximately 55 MB to work with one Lisp, one C++ and one Java version of STELLA in parallel. If you also want to experiment with the Lisp translation variant that uses structures instead of CLOS instances to implement STELLA objects, then you will need an extra 8 MB to compile that.

The full STELLA development tree is quite large, since for every STELLA source file there are three to four translated versions and as many compiled versions thereof. The actual STELLA libraries that you have to ship with an application, however, are quite small. For example, the Java jar file ‘stella.jar’ is only about 1.3 MB in size. The dynamic C++ library ‘libstella.so’ compiled on a Linux platform is about 4 MB. Additionally, if you don’t need all the different translations of STELLA, you can delete some of the versions to keep your development tree smaller (See Section 2.7 [Removing Unneeded Files], page 6).

To run the Lisp version of STELLA you need an ANSI Common-Lisp (or at least one that supports CLOS and logical pathnames). We have successfully tested STELLA with Allegro-CL 4.2, 4.3, 5.0, 6.0 and 7.0, Macintosh MCL 3.0, 4.0 and 5.1, OpenMCL 1.0, SBCL 0.9.4, CMUCL 19c, LispWorks 4.4.6, CLisp 2.37, Lucid CL 4.1 (plus the necessary ANSI extensions and Mark Kantrowitz’s logical pathnames implementation) and various others. Our main development platform is Allegro CL running under Sun Solaris and Linux, so, the closer your environment is to ours, the higher are the chances that everything will work right out of the box. Lisp development under Windows should also be no problem.

To run the C++ version of STELLA you need a C++ compiler such as g++ that supports templates and exception handling. We have successfully compiled and run STELLA with g++ 3.2 and later under Linux Redhat 8.0 & 9.0, SunOS and MacOS X, and with CygWin 5.0 and MinGW 5.0 under Windows 2000 and XP. Both CygWin and MinGW provide a GNU Unix environment, but MinGW can generate Windows executables that are fully standalone. We’ve also managed to compile STELLA under MS Visual C++, however, we never got the Boehm garbage collector to work. The GC claims to be very portable, so this should be solvable for somebody with good knowledge of MS Visual C++.

For the Java version you will need Java JDK 1.2 or later. To get reasonable performance, you should use J2SDK 1.4 or 1.5. We’ve run the Java version of STELLA on a variety of platforms without any problems.

Any one of the Lisp, C++ or Java implementations of STELLA can be used to develop your own STELLA code and translate it into all three languages, but the most convenient
development environment is the one based on Lisp. If you use the C++ or Java version, translating and using your own STELLA macros is possible but not yet very well supported.

2.2 Unpacking the Sources

Uncompress and untar the file `stella-X.Y.Z.tar.gz` (or unzip the file `stella-X.Y.Z.zip`) in the parent directory of where you want to install STELLA (`X.Y.Z` are place holders for the actual version numbers). This will create the STELLA tree in the directory `stella-X.Y.Z/`. All pathnames mentioned below will be relative to that directory which we will usually refer to as the "STELLA directory".

2.3 Lisp Installation

To install the Lisp version startup Lisp and load the file `load-stella.lisp` with:

```lisp
(CL:load "load-stella.lisp")
```

The first time around this will compile all Lisp-translated STELLA files before they are loaded. During subsequent sessions the compiled files will be loaded right away.

If you want to use the version that uses Lisp structs instead of CLOS objects to implement STELLA objects do the following:

```lisp
(CL:setq cl-user::*load-cl-struct-stella?* CL:t)
(CL:load "load-stella.lisp")
```

Alternatively, you can edit the initial value of the variable `*load-cl-struct-stella?*` in the file `load-stella.lisp`. Using structs instead of CLOS objects greatly improves slot access speed, however, it may cause problems with incremental re-definition of STELLA classes. It is therefore recommended to only use this for systems that are in or near the production stage.

Once all the files are loaded, you should see a message like this:

```
Initializing STELLA...
STELLA 3.4.0 loaded.
Type '(in-package "STELLA")' to execute STELLA commands.
USER(2):
```

To reduce startup time, you might want to create a Lisp image that has all of STELLA preloaded.

Now type

```lisp
(in-package "STELLA")
```

to enter the STELLA Lisp package where all the STELLA code resides.

**IMPORTANT:** All unqualified Lisp symbols in this document are assumed to be in the STELLA Lisp package. Moreover, the STELLA package does **NOT** inherit anything from the COMMON-LISP package (see the file `sources/stella/cl-lib/cl-setup.lisp` for the few exceptions), hence, you have to explicitly qualify every Lisp symbol you want to use with `CL:`. For example, to get the result of the previous evaluation you have to type `CL::*` instead of `*`. 
2.4 C++ Installation

To compile the C++ version of STELLA change to the native C++ directory and run make:

```bash
% cd native/cpp/stella
% make
```

This will compile all STELLA files, the garbage collector and generate a static or dynamic ‘libstella’ library file in the directory ‘native/cpp/lib’ which can later be linked with your own C++-translated STELLA (or other) code. To test whether the compilation was successful you can run STELLA from the top-level STELLA directory using the `stella` script (or `stella.bat` under a Windows command prompt):

```bash
% stella c++
```

```
Running C++ version of STELLA...
Welcome to STELLA 3.4.0
Running kernel startup code...
Initializing symbol tables...
Initializing quoted constants...
Initializing global variables...
Creating class objects...
Finalizing classes...
Creating method objects...
Finalizing methods...
Running non-phased startup code...
Starting up translators...
Bye!
```

This will simply run various STELLA startup code and exit. See Section 4.1.2 [Hello World in C++], page 14, to see how you can use the STELLA C++ executable to translate STELLA code. The `c++` argument tells the script to run the C++ version of STELLA. If the argument is omitted and the C++ version is installed, it will be run automatically. Otherwise, the Java version will be run.

2.5 Java Installation

Nothing needs to be done to install the Java version. Since Java class files are platform independent, they are already shipped with the STELLA distribution and can be found in the directory ‘native/java’ and its subdirectories. Additionally, they have been collected into the file ‘native/java/lib/stella.jar’. To try out the Java version of STELLA run the `stella` script in the STELLA directory:

```bash
% stella java
```

```
Running Java version of STELLA...
Welcome to STELLA 3.4.0
Running kernel startup code...
Initializing symbol tables...
Initializing quoted constants...
Initializing global variables...
```
Creating class objects...
Finalizing classes...
Creating method objects...
Finalizing methods...
Running non-phased startup code...
Starting up translators...
Bye!

Similar to the C++ executable, this will simply run various STELLA startup code and exit. See Section 4.1.3 [Hello World in Java], page 16, to see how you can use the STELLA Java executable to translate STELLA code.

2.6 X/Emacs Setup

STELLA development is very similar to Lisp development, and it is best done in an X/Emacs-based Lisp development environment such as the Allegro-CL Emacs interface plus Allegro Composer, or ILISP. If you do use X/Emacs with the Allegro CL interface, add the following to your `.emacs` or `.xemacs/init.el` file:

```
(setq auto-mode-alist
  (cons `("\.ste\$" . fi:common-lisp-mode) auto-mode-alist))
```

If you are using the Allegro CL interface, you might want to install the file `emacs/fi-stella.el`, since it sets up proper indentation for STELLA code and makes looking up STELLA definitions via the `C-c .` or `M-.` commands work better. Look at the file `emacs/fi-stella.el` for specific installation instructions.

2.7 Removing Unneeded Files

To save disk space you can remove files that you don’t need. For example, if you are not interested in the C++ version of STELLA, you can delete the directory `native/cpp`. Similarly, you can remove `native/java` to eliminate all Java-related files. You could do the same thing for the Lisp directory `native/lisp`, but (in our opinion) that would make it less convenient for you to develop new STELLA code. Finally, if you don’t need any of the STELLA sources, you can delete the directory `sources/stella`. If you don’t need local copies of the STELLA documentation, you can delete parts or all of the `sources/stella/doc` directory.
3 The STELLA Language

3.1 Language Overview

STELLA is a strongly typed, object-oriented, Lisp-like language. STELLA programs are first translated into either Common Lisp, C++, or Java, and then compiled with any conventional compiler for the chosen target language to generate executable code. Over 95% of the STELLA system is written in STELLA itself, the rest is written in target-language-specific native code.

The design of STELLA borrows from a variety of programming languages, most prominently from Common Lisp, and to a lesser degree from other object-oriented languages such as Eiffel, Sather, and Dylan. Since STELLA has to be translatable into C++ and Java, various restrictions of these languages also influenced its design.

In the following, we assume that the reader is familiar with basic Common Lisp concepts, and has at least some familiarity with C++ or Java. Let us start with a cursory overview of STELLA’s main features:

Syntax: STELLA uses a parenthesized, uniform expression syntax similar to Lisp. Most definitional constructs and control structures are similar to their Common Lisp analogues with variations to support types.

Type system: STELLA is strongly typed and supports efficient static compilation similar to C++. Types are required for the arguments and return values of functions and methods, for global variables, and for slot definitions. Local, lexically scoped variables can be typed implicitly by relying on type inference.

Object system: Types are organized into a single inheritance class hierarchy. Restricted multiple inheritance is allowed via mixin classes. Dynamic method dispatch is based on the runtime type of the first argument (similar to C++ and Java). Slots can be static (native) or dynamic. Dynamic slots can be defined at runtime and do not occupy any space until they are filled. Slots can have both initial and default values, and demons can be triggered by slot accesses. A meta-object protocol allows the control of object creation, initialization, termination, and destruction.

Control structure: Functions and methods are distinguished. They can have multiple (zero or more) return values and a variable number of arguments. Lisp-style macros are supported to facilitate syntax extensions. Expressions and statements are distinguished. Local variables are lexically scoped, but dynamically scoped variables (specials) are also supported. STELLA has an elegant, uniform, and efficient iteration mechanism plus a built-in protocol for iterators. An exception mechanism can be used for error handling and non-local exits.

Symbolic programming: Symbols are first-class objects, and extensive support for dynamic datatypes such as cons-trees, lists, sets, association lists, hash tables, extensible vectors, etc., is available. A backquote mechanism facilitates macro writing and code generation. Interpreted function call, method call, slot access, and object creation is supported, and a restricted evaluator is also available.
**Name spaces:** Functions, methods, variables, and classes occupy separate name spaces (i.e., the same name can be used for a function and a class). A hierarchical module system compartmentalizes symbol tables and supports large-scale programming.

**Memory management:** STELLA relies on automatic memory management via a garbage collector. For Lisp and Java the native garbage collector is used. For the C++ version of STELLA we use the Boehm-Weiser conservative garbage collector with good results. Various built-in support for explicit memory management is also available.

The Common Lisp features most prominently absent from STELLA are anonymous functions via lambda abstraction, lexical closures, multi-methods, full-fledged eval (a restricted evaluator is available), optional and keyword arguments, and a modifiable readtable. STELLA does also not allow dynamic re/definition of functions and classes, even though the Lisp-based development environment provides this facility (similar to Dylan). The main influences of C++ and Java onto STELLA are the strong typing, limited multiple inheritance, first-argument polymorphism, and the distinction between statements and expressions.

### 3.2 Basic Data Types (tbw)

To be written.

### 3.3 Control Structure (tbc)

To be completed.

#### 3.3.1 Conditionals

STELLA conditionals are very similar to those found in Common-Lisp. The main difference is that most STELLA conditionals are statements and therefore do not return a value. For this reason, a C++-style `choose` directive has been added to the language to allow value conditionalization based on a boolean expression.

**if** condition **then-statement** **else-statement** [Statement]

Evaluate the boolean expression `condition`. If the result is true execute `then-statement`, otherwise, execute `else-statement`. Note that unlike the Common-Lisp version of `if` the `else-statement` is not optional in STELLA. Example:

```
(if (> x y)
  (print "x is greater than y" EOL)
  (print "x is less than or equal to y" EOL))
```

**when** condition **statement** . . . [Statement]

Evaluate the boolean expression `condition`. Only if the result is true execute the `statement`s in the body. Example:

```
(when (symbol? x)
  (print "x is a symbol, ")
  (print "its name is " (symbol-name (cast x SYMBOL)) EOL))
```
unless condition statement...  
Evaluate the boolean expression condition. Only if the result is false execute the statement’s in the body. Therefore, (unless test ...) is equivalent to (when (not test) ...). Example:

(unless (symbol? x)
  (print "x is not a symbol, ")
  (print "hence, its name is unknown" EOL))

cond clause...  
cond is a conditional with an arbitrary number of conditions each represented by a clause. Each cond clause has to be of the following form:

(condition statement...)

The first clause whose condition evaluates to true will be selected and its statement’s will be executed. Each clause can have 0 or more statements. The special condition otherwise always evaluates to true and can be used for the catch-all case. Example:

(cond ((symbol? x)
  (print "x is a symbol" EOL))
  ((cons? x)
  (print "x is a cons" EOL))
  (otherwise
  (print "x is an object" EOL)))

choose condition true-expression false-expression  
Evaluate the boolean expression condition. If the result is true return the value of true-expression, otherwise, return the value of false-expression. STELLA computes the most specific common supertype of true-expression and false-expression and uses that as the type returned by the choose expression. If no such type exists, a translation error will be signaled. Example:

(setq face (choose happy? :smile :frown))

case expression clause...  
Each case clause has to be of one of the following forms:

(key statement...)

((key...) statement...)

case selects the first clause whose key (or one of the listed key’s) matches the result of expression and executes the clause’s statement’s. Each case key has to be a constant such as a number, character, string, symbol, keyword or surrogate. Keys are compared with eql? (or string-eql? for strings). All keys in a case statement have to be of the same type. The special key otherwise can be used to catch everything. It is a run-time error if no clause with a matching key exists. Therefore, a STELLA case without an otherwise clause corresponds to a Common Lisp ecase. An empty otherwise clause can always be specified via (otherwise NULL). Example:

(case car-make
  ("Yugo"
   (setq price :cheap))
  ("VW")
(setq price :medium))
("Ferrari" "Rolls Royce")
(setq price :expensive))
(otherwise
 (setq price :unknown)))

(typecase expression clause)

Each typecase clause has to be of one of the following forms:

(type statement)
((type...) statement...)

(typecase) selects the first clause whose type (or one of the listed type’s) equals or is a supertype of the run-time type of the result of expression and then executes the clause’s statement’s. Therefore, typecase can be used to implement a type dispatch for cases where the run-time type of an expression can be different from the static type known at translation time. Currently, the static type of expression is required to be a subtype of OBJECT.

Each type expression has to be a symbol describing a simple type (i.e., parametric or anchored types are not allowed). Similar to case, the special key otherwise can be used to catch everything. It is a run-time error if no clause with a matching type exists. Therefore, a STELLA typecase without an otherwise clause corresponds to a Common Lisp etypecase. An empty otherwise clause can always be specified via (otherwise NULL). typecase does allow the value of expression to be undefined, in which case the otherwise clause is selected. Example:

(typecase (first list)
 (CON5
 (print "\textit{it is a cons}"))
 ((\texttt{SYMBOL} KEYWORD)
 (print "\textit{it is a symbol}"))
 (STANDARD-OBJECT
 (print "\textit{it is a regular object}"))
 (otherwise NULL))

Note that in the example above it is important to list STANDARD-OBJECT after SYMBOL and CONS, since it subsumes the preceding types. Otherwise, it would always shadow the clauses with the more specific types.

The semantics of typecase is slightly extended for the case where expression is a local variable. In that case each reference to the variable within a typecase clause is automatically casted to the appropriate narrower type. For example, in the code snippet below method calls such as first or slot accesses such as symbol-name are translated correctly without needing to explicitly downcast x which is assumed to be of type OBJECT:

(typecase x
 (CON5
 (print "\textit{it is a cons with value} \texttt{\first x}"))
 ((\texttt{SYMBOL} KEYWORD)
 (print "\textit{it is a symbol with name} \texttt{\symbol-name x}"))
 (STANDARD-OBJECT
 (print "\textit{it is a standard object}")))
(print "it is a regular object")
(otherwise NULL))

Since the typecase expression has to be a subtype of OBJECT, a typecase cannot be used to test against literal types such as STRING or INTEGER. If such type names are encountered as keys in a typecase, they are automatically converted to their wrapped version, e.g., STRING-WRAPPER, INTEGER-WRAPPER, etc.

3.4 Functions (tbw)
To be written.

3.5 Classes (tbw)
To be written.

3.6 Methods (tbw)
To be written.

3.7 Macros (tbw)
To be written.

3.8 Modules (tbw)
To be written.
4 Programming in STELLA

4.1 Hello World in STELLA

Included with the STELLA distribution is a simple Hello World application that shows you how to organize your own STELLA code and build a working STELLA application. The sources for the Hello World system consist of the following files:

- sources/systems/hello-world-system.ste
- sources/hello-world/file-a.ste
- sources/hello-world/file-b.ste

STELLA organizes code modules with a simple system facility. Translation always operates on a complete system, so you always need to create a system definition for the STELLA files comprising your application (somewhat similar to what you would put in a Unix Makefile).

For the Hello World system the system definition already exists and resides in the file `sources/systems/hello-world-system.ste`. By default, STELLA looks in the directory `sources/systems` to find the definition of a particular system. 

`hello-world-system.ste` defines two things:

1. The `HELLO-WORLD` module which defines a namespace for all objects in the Hello World systems. STELLA modules are mapped onto corresponding native namespace constructs, i.e., Lisp packages, C++ namespaces or Java packages. The exact mapping for each language can be defined via the keyword options :lisp-package, :cpp-package and :java-package in the module definition, for example:

   ```steella
   (defmodule "HELLO-WORLD"
      :lisp-package "STELLA"
      :cpp-package "hello_world"
      :java-package "edu.isi.hello_world"
      :uses ("STELLA"))
   ```

   The :uses directive tells STELLA from what other modules this one inherits.

2. The actual system definitions defining what source files comprise the system, and what parent systems this one depends on, plus a variety of other options:

   ```steella
   (defsystem HELLO-WORLD
      :directory "hello-world"
      :required-systems ("stella")
      :cardinal-module "HELLO-WORLD"
      :production-settings (1 0 3 3)
      :development-settings (3 2 3 3)
      :files ("file-a"
              "file-b"))
   ```
4.1.1 Hello World in Lisp

To generate a Lisp translation of Hello World you can use either the Lisp, C++ or Java version of STELLA. Before you can translate you have to make sure the following native directories exist:

```
native/lisp/hello-world/
bin/acl7.0/hello-world/
```

The directory `native/lisp/hello-world/` will hold the Lisp translations of the corresponding STELLA source files. The directory `bin/acl7.0/hello-world/` will hold the compiled Lisp files if you are using Allegro CL 7.0. If you are using a different Lisp, one of the other binary directories as defined in the top-level file `translations.lisp` will be used. The directory `bin/lisp/hello-world/` will be used as a fall-back if your version of Lisp is not yet handled in `translations.lisp`.

If you create your own system, you will need to create those directories by hand (future versions of STELLA might do that automatically). For the Hello World system these directories already exist.

To generate a Lisp translation of Hello World using Lisp startup a Lisp version of STELLA (see Section 2.3 [Lisp Installation], page 4). The following idiom will then translate the system into Lisp and also Lisp-compile and load it. The first argument to `make-system` is the name of the system, and the second argument indicates into what language it should be translated:

```
STELLA(3): (make-system "hello-world" :common-lisp)
Processing '/tmp/stella-3.1.0/sources/hello-world/file-a.ste':
*** Pass 1, generating objects...
Processing '/tmp/stella-3.1.0/sources/hello-world/file-b.ste':
*** Pass 1, generating objects...
........................................
;;; Writing fasl file
;;; /tmp/stella-3.1.0/native/lisp/bin/acl7.0/hello-world/startup-system.fasl
;;; Fasl write complete
; Fast loading
; /tmp/stella-3.1.0/native/lisp/bin/acl7.0/hello-world/startup-system.fasl
CL:T
STELLA(4):
```

After the system is loaded you can call its `main` function:

```
STELLA(10): (main)
Hello World A
Hello World B
bye
()
STELLA(11):
```

Using `main` in the Lisp version will not always make sense, since you can call any function directly at the Lisp top level, but both C++ and Java always need a `main` function as a top-level entry point.
While this would be somewhat unusual, you could also generate the Lisp translation using the C++ or Java version of STELLA. The easiest way to do that is to run the stella script in the STELLA directory like this:

```
% ./stella -e '(make-system "hello-world" :common-lisp)'
```

Running C++ version of STELLA...
Welcome to STELLA 3.4.0
Processing ‘sources/hello-world/file-a.ste’:
*** Pass 1, generating objects...
Processing ‘sources/hello-world/file-b.ste’:
*** Pass 1, generating objects...

...............................................

Translating ‘sources/hello-world/file-a.ste’ to ‘Common Lisp’...
Writing ‘native/lisp/hello-world/file-a.lisp’...
Translating ‘sources/hello-world/startup-system.ste’ to ‘Common Lisp’...
Writing ‘native/lisp/hello-world/startup-system.lisp’...

The -e command line option is used to evaluate an evaluable STELLA command. Conveniently, make-system is such a command, so you can supply a make-system form to the C++ or Java version of STELLA just as you would do in Lisp. Note the extra quotes around the expression to protect the characters from interpretation by the Unix shell.

To compile and load the translated Lisp files into Lisp you then have to startup a Lisp version of STELLA and call make-system again which now will only compile and load the necessary files, since the translations have already been generated in the previous step.

4.1.2 Hello World in C++

To generate a C++ translation of Hello World you can use either the Lisp, C++ or Java version of STELLA. Before you can translate you have to make sure the following native directory exists:

```
native/cpp/hello-world/
```

The directory ‘native/cpp/hello-world/’ will hold the C++ translations of the corresponding STELLA source files. If you create your own system, you will need to create this directory by hand (future versions of STELLA might do that automatically). For the Hello World system the directory already exist.

To generate a C++ translation of Hello World using Lisp startup a Lisp version of STELLA (see Section 2.3 [Lisp Installation], page 4). The following idiom will then translate the system into C++. The first argument to make-system is the name of the system, and the second argument indicates into what language it should be translated:

```
STELLA(4): (make-system "hello-world" :cpp)
Processing ‘/tmp/stella-3.1.0/sources/hello-world/file-a.ste’:
*** Pass 1, generating objects...
Processing ‘/tmp/stella-3.1.0/sources/hello-world/file-b.ste’:
*** Pass 1, generating objects...
```

Writing ‘/tmp/stella-3.1.0/native/cpp/hello-world/file-b.hh’...
Writing ‘/tmp/stella-3.1.0/native/cpp/hello-world/file-b.cc’...
Translating ‘/tmp/stella-3.1.0/sources/hello-world/startup-system.ste’.
Writing ‘/tmp/stella-3.1.0/native/cpp/hello-world/startup-system.hh’...
Writing ‘/tmp/stella-3.1.0/native/cpp/hello-world/startup-system.cc’...

:VOID
STELLA(5):

Alternatively, you can generate the translation using the C++ or Java version of STELLA.
The easiest way to do that is to run the stella script in the STELLA directory like this:

```
% ./stella -e '(make-system "hello-world" :cpp)
```

Running C++ version of STELLA...
Welcome to STELLA 3.4.0
Processing ‘sources/hello-world/file-a.ste’:
*** Pass 1, generating objects...
Processing ‘sources/hello-world/file-b.ste’:
*** Pass 1, generating objects...

...............................................

Writing ‘native/cpp/hello-world/file-b.hh’...
Writing ‘native/cpp/hello-world/file-b.cc’...
Translating ‘sources/hello-world/startup-system.ste’.
Writing ‘native/cpp/hello-world/startup-system.hh’...
Writing ‘native/cpp/hello-world/startup-system.cc’...

The -e command line option is used to evaluate an evaluable STELLA command. Conveniently, make-system is such a command, so you can supply a make-system form to the C++ or Java version of STELLA just as you would do in Lisp. Note the extra quotes around the expression to protect the characters from interpretation by the Unix shell.

Different from Lisp, neither of the above idioms will compile and load the generated C++ code. Instead you have to use the Unix ‘make’ facility to compile and link the C++ sources. First change into the native ‘hello-world’ directory and then call make (important: the generated Makefiles currently require the GNU version of make):

```
% cd native/cpp/hello-world/
% make
```

g++ -w -g -O2 -DSTELLA_USE_GC -I../stella/cpp-lib/gc/include \
   -c -I. main.cc
g++ -w -g -O2 -DSTELLA_USE_GC -I../stella/cpp-lib/gc/include \
   -c -I. file-a.cc
g++ -w -g -O2 -DSTELLA_USE_GC -I../stella/cpp-lib/gc/include \
   -c -I. file-b.cc
g++ -w -g -O2 -DSTELLA_USE_GC -I../stella/cpp-lib/gc/include \
   -c -I. startup-system.cc

.................................
g++ -dynamic -L../stella/cpp-lib/gc -Xlinker -rpath -Xlinker \
   ‘./lib:/tmp/stella-3.1.0/native/cpp/lib’ \ 
   main.o -o hello-world \ 
   -L../lib -lhello-world -L../lib -lstella -lgc -lm

The first time around this will also compile the C++ version of STELLA and the C++ garbage collector and create a STELLA library file. Future builds of the Hello World and
other systems will use the STELLA library file directly. To run the Hello World system simply run the `hello-world` executable that was built in the previous step:

```
% ./hello-world
Hello World A
Hello World B
bye
```

### 4.1.3 Hello World in Java

To generate a Java translation of Hello World you can use either the Lisp, C++ or Java version of STELLA. Before you can translate you have to make sure the following native directory exists:

```
native/java/edu/isi/hello-world/
```

The directory `native/java/edu/isi/hello-world/` will hold the Java translations of the corresponding STELLA source files. If you create your own system, you will need to create this directory by hand (future versions of STELLA might do that automatically). For the Hello World system the directory already exist.

Note that following Java convention we use the package `edu.isi.hello_world` to hold the Hello World system. This was specified via the `:java-package` option in the definition of the `HELLO-WORLD` module. Also note that we use `hello_world` instead of `hello-world` as the package name, since a dash cannot legally appear as part of a Java identifier.

To generate a Java translation of Hello World using Lisp startup a Lisp version of STELLA (see Section 2.3 [Lisp Installation], page 4). The following idiom will then translate the system into Java. The first argument to `make-system` is the name of the system, and the second argument indicates into what language it should be translated:

```
STELLA(5): (make-system "hello-world" :java)
Processing '/tmp/stella-3.1.0/sources/hello-world/file-a.ste':
*** Pass 1, generating objects...
..............................................
Writing '/tmp/stella-3.1.0/native/java/hello_world/Startup_Hello_... :VOID
STELLA(6):
```

Alternatively, you can generate the translation using the C++ or Java version of STELLA. The easiest way to do that is to run the `stella` script in the STELLA directory like this:

```
% ./stella -e '(make-system "hello-world" :java)'
Running C++ version of STELLA...
Welcome to STELLA 3.4.0
Processing 'sources/hello-world/file-a.ste':
*** Pass 1, generating objects...
Processing 'sources/hello-world/file-b.ste':
*** Pass 1, generating objects...
..............................................
Writing 'native/java/edu/isi/hello_world/HelloWorld.java'...
Writing 'native/java/edu/isi/hello_world/StartupFileA.java'...
Writing 'native/java/edu/isi/hello_world/StartupFileB.java'...
```
Chapter 4: Programming in STELLA

Writing ‘native/java/edu/isi/hello_world/StartupHelloWorldSystem.java’...

The -e command line option is used to evaluate an evaluable STELLA command. Conveniently, make-system is such a command, so you can supply a make-system form to the C++ or Java version of STELLA just as you would do in Lisp. Note the extra quotes around the expression to protect the characters from interpretation by the Unix shell.

Different from Lisp, neither of the above idioms will compile and load the generated C++ code. Instead you have to use the Java compiler to compile and Java to run the compiled Java sources. First change into the top-level native Java directory 'native/java' and then compile and run the Hello World system like this:

    % cd native/java/
    % javac edu/isi/hello_world/*.java
    % java edu.isi.hello_world.HelloWorld

Hello World A
Hello World B
bye

It is not necessary to Java-compile STELLA first, since STELLA already ships with a Java compilation of the STELLA system.

4.2 Incrementally Developing STELLA Code

The preferred method of STELLA code development is to use a Lisp-based version of STELLA for all the prototyping and testing, since that allows you to exploit most (or all) of the rapid-prototyping advantages of Lisp. Once a system has reached a certain point of stability, it can be translated into C++ or Java for delivery or to interface it with other C++ or Java code.

In the following, we assume an X/Emacs-based Lisp development environment such as the Allegro CL Emacs interface, where Lisp is run in an Emacs subprocess, and Lisp source can be compiled and evaluated directly from the source buffers. By "Lisp buffer" we mean the listener buffer in which Lisp is actually running, and by "source buffer" we mean a buffer that is used to edit a file that contains STELLA source.

Included in the distribution is the Hello World system comprised of the files

    sources/systems/hello-world-system.ste
    sources/hello-world/file-a.ste
    sources/hello-world/file-b.ste

To get started, simply add your code to either ‘file-a.ste’ or ‘file-b.ste’, since all the necessary definitions and directories for these files are already set up properly. See section ??? on how to setup your own system.

Make sure the Hello World system is loaded into Lisp by doing the following:

    (make-system "hello-world" :common-lisp)

This will make sure that the system definition is loaded and the necessary module definition is evaluated.

Now suppose you add the following function to ‘file-a.ste’:
(defun (factorial INTEGER) ((n INTEGER))
  (if (eql? n 0)
    (return 1)
    (return (* n (factorial (1- n))))))

There are various options for translating and evaluating this definition. For example, you can simply remake the complete system similar to what you would do for a C++ or Java program:

(make-system "hello-world" :common-lisp)

This will retranslate the modified files, recompile them and reload them into your Lisp image.

Instead of retranslating and recompiling everything, you can incrementally evaluate the definition of `factorial` from your Emacs-to-Lisp interface. Simply put your cursor somewhere inside the definition in the source buffer and evaluate it by typing `M-C-x`. This translates the STELLA code into Lisp and compiles (or evaluates) the resulting Lisp code. Now you can actually try it out in the Lisp buffer, for example:

STELLA(4): (factorial 6)
720

Finally, instead of evaluating the definition in the source buffer, you can also enter it directly at the Lisp prompt with the same effect.

The way this works is that the Lisp symbol `stella::defun` is actually bound to a Lisp macro that calls all the necessary translation machinery to convert the STELLA `defun` into Lisp code. Look at the file `sources/stella/cl-lib/stella-to-cl.ste` for the complete set of such macros. This might be a bit confusing, since there are now three different bindings (or meanings) of `defun`:

1. The STELLA operator `defun` used to define STELLA functions.
2. The Lisp macro `stella::defun` that resides in the STELLA Lisp package and is only available for convenience in Lisp versions of STELLA.
3. The Lisp macro `CL:defun` which is the standard Common Lisp macro used to define Lisp functions.

We'll try to explicitly qualify which meaning is used wherever there might be some doubt which one is meant. In general, every unqualified symbol mentioned below is either part of the STELLA language or resides in the STELLA Lisp package.

Since a newly-written STELLA function might have errors, it is prudent to first only translate it without actually executing the result of the translation. In the source buffer you can do that by macro-expanding the `defun`. For example, if you use the Allegro CL interface you would position the cursor on the opening parenthesis of the `defun` and then type `M-M`. Any errors discovered by the STELLA translator are reported in the Lisp buffer window. The expansion will be a `CL:progn` that contains the translated definition as the first element plus various startup-time (initialization) code following it.

In the Lisp buffer you can achieve a similar effect with the `lptrans` macro. For example, executing

(lptrans
  (defun (factorial INTEGER) ((n INTEGER)))

```
(if (eql? n 0)
  (return 1)
  (return (* n (factorial (1- n)))))

in the Lisp buffer first Lisp-translates the definition, and then prints the translation. To see the C++ translation you can use cpptrans, calling jptrans will generate the Java translation.

You can also use lptrans/cpptrans/jptrans to translate code fragments that are not top-level definitions such as defun and its friends. For example:

STELLA(8): (lptrans
  (foreach element in (list 1 2 3)
    do (print element EOL)))

(CL:LET* ((ELEMENT NULL)
  (ITER-003 (%THE-CONS-LIST (LIST (WRAP-INTEGER 1) (WRAP-INTEGER 2)
    (WRAP-INTEGER 3)))))
  (CL:LOOP WHILE (CL:NOT (CL:EQ ITER-003 NIL)) DO
    (CL:PROGN (SETQ ELEMENT (%%VALUE ITER-003))
      (SETQ ITER-003 (%%REST ITER-003)))
    (%%PRINT-STREAM (%NATIVE-STREAM STANDARD-OUTPUT)
      ELEMENT EOL)))
)

STELLA(9): (cpptrans
  (foreach element in (list 1 2 3)
    do (print element EOL)))

{ Object* element = NULL;
  Cons* iter004 = list(3, wrapInteger(1), wrapInteger(2),
    wrapInteger(3))-> theConsList;

  while (!(iter004 == NIL)) {
    element = iter004->value;
    iter004 = iter004->rest;
    cout << element << endl;
  }
}
:VOID

STELLA(10): (jptrans
  (foreach element in (list 1 2 3)
    do (print element EOL)))

{ Stella_Object element = null;
  Cons iter005 = Stella.list
    (Stella_Object.cons
      (IntegerWrapper.wrapInteger(1),
        Stella_Object.cons
          (IntegerWrapper.wrapInteger(2),
            Stella_Object.cons
              (IntegerWrapper.wrapInteger(3)))
      ))
  for (int i = 0; i < 3; i++) {
    element = iter005->value;
    iter005 = iter005->rest;
    cout << element << endl;
  }
}
:VOID
(IntegerWrapper.wrapInteger(3),
   Stella.NIL))).theConsList;

while (!(iter005 == Stella.NIL)) {
    {
        element = iter005.value;
        iter005 = iter005.rest;
    }
    java.lang.System.out.println(element);
}
}
:VOID

The use of \texttt{lptrans} is really necessary here, since there is no Lisp macro \texttt{foreach} that knows how to translate STELLA \texttt{foreach} loops (those Lisp macros only exist for top-level definition commands such as \texttt{defun}). In order to translate such code fragments without error messages, they need to be self-contained, i.e., all referenced variables have to be either bound by a surrounding \texttt{let}, or they must be globally defined variables. Otherwise, the STELLA translator will generate various "undefined variable" error messages.

You can use the STELLA Lisp macro \texttt{eval} (i.e., \texttt{stella::eval} not \texttt{CL:eval}) to actually execute such a code fragment. For example:

\texttt{STELLA(11): (eval (foreach element in (list 1 2 3)
   do (print element EOL)))}

|L|1
|L|2
|L|3

This translates the loop and executes the result, which prints the wrapped numbers (hence, the \texttt{|L|} prefix) to standard output. The \texttt{()} at the end is the resulting Lisp value returned by the loop (in Lisp everything returns a value, even though for STELLA \texttt{foreach} is a statement, not an expression).

Make it a habit to wrap \texttt{eval} around any STELLA code you incrementally evaluate in the Lips buffer. This makes sure that all the arguments to a function, etc., are translated into the appropriate STELLA objects. For example, evaluating

\texttt{(eval (list :a :b :c))}

in the Lisp buffer generates a STELLA list that points to the STELLA keywords :a, :b and :c. If you don’t use \texttt{eval}, for example,

\texttt{(list :a :b :c)}

a STELLA list containing the Lisp keywords ‘:a’, ‘:b’ and ‘:c’ will be created. Lisp keywords are a completely different data structure than STELLA keywords, and any STELLA code expecting a STELLA keyword but finding a Lisp keyword will break, since Lisp keywords are not a legal STELLA data structure. Unfortunately, such cases can be very confusing, since Lisp and STELLA keywords look/print exactly alike.
**eval** is also necessary to access STELLA symbols and surrogates in the Lisp buffer. For example, to access a STELLA symbol, you can use **quote** (again, this is the STELLA **quote** not **CL:quote**):

```
(eval (quote foo))
```

This returns the STELLA symbol **foo**. We explicitly used **quote** here, since code typed at the Lisp prompt is first passed through the Lisp reader before the STELLA translator sees it, and the default Lisp reader interprets the ' character differently than the STELLA reader. Within a STELLA file you can use the syntax 'foo, since it will be read directly by the STELLA reader that knows how to interpret it correctly.

**lptrans**, **cpptrans** and **jptrans** are evaluable STELLA commands that can also be evaluated by the C++ and Java version of STELLA. For example, to generate a Java translation of a little STELLA code fragment you could run the **stella** script in the STELLA directory like this (the output below has been additionally indented by hand for clarity):

```
% ./stella -e '(jptrans\n    (foreach element in (list 1 2 3)\n      do (print element EOL)))'
```

Running C++ version of STELLA...
Welcome to STELLA 3.4.0
{ Stella_Object element = null;
  Cons iter001 = Stella.list
    (Stella_Object.cons
      (IntegerWrapper.wrapInteger(1),
       Stella_Object.cons
         (IntegerWrapper.wrapInteger(2),
          Stella_Object.cons
            (IntegerWrapper.wrapInteger(3),
             Stella.NIL))).theConsList;

  while (!(iter001 == Stella.NIL)) {
    {
      element = iter001.value;
      iter001 = iter001.rest;
    }
    java.lang.System.out.println(element);
  }
```

### 4.3 Performance Hints

Here are a few things to watch out for once you get serious about the performance of your translated STELLA programs:

**Safety checks:** The STELLA variable **safety** controls whether certain safety code is added to your translated STELLA program. For Lisp translations it also controls whether cast’s will be translated into run-time type checks or not. There is no run-time type checking performed in C++. In Java native casts will always perform runtime type tests.
The default *safety* level is 3 which enables the translation of all safety clauses with level 3 or lower. A safety level of 1 or lower disables the generation of calls to the cast function in Lisp. cast performs run-time type checks which are somewhat expensive. However, you should not disable run-time type checking in Lisp until you have fully debugged your program. Once you are confident that your program works correctly, you can set *safety* to 0 before you translate it. That way you will avoid the generation and execution of any safety code at all. All of the core STELLA system was translated with *safety* set to 1.

Quoted cons trees: Access to quoted constants that are not symbols is somewhat slow, since it currently uses hashing to find them in a table. Hence, access to quoted constants such as (quote (foo bar fum)) should be avoided in inner loops. Access to quoted symbols such as (quote foo) is fast and does not cause any performance problems. The use of quote for constant cons trees is rare in STELLA (and somewhat deprecated), which is the reason why this mechanism is not all that well supported. Future versions of STELLA might re-implement the handling of constants and alleviate this performance problem.

Equality tests: The standard equality test in STELLA is eql?, which the translator will translate into the most efficient equality test for the particular types of operands (eql? is somewhat similar to the Lisp function CL:eql with the exception of comparing strings). If the translator can determine that at least one of the operands is a subtype of STANDARD-OBJECT, it will translate the test into a fast pointer comparison with the Lisp function CL:eq or the C++/Java == operator. However, if both operands are of type OBJECT, they might be wrapped literals such as wrapped integers or strings. In that case the equality test translates into a call to the function eql? which in turn uses method calls to handle comparison of different types of wrapped literals (two wrapped literals are equal if their wrapped content is equal). This is of course a lot less efficient than a simple pointer comparison. It also means that if you can restrict the type of a variable that will be tested with eql? to STANDARD-OBJECT, you probably should do so for performance reasons.

Type tests: Run-time type tests as used implicitly within a typecase or explicitly with functions such as cons? have to use a call to the method primary-type. Hence, in performance-critical portions of your code you should try to keep the number of such tests as small as possible.

Wrapping and unwrapping literals: The STELLA translator automatically wraps (or objectifies) literals such as numbers or strings when they are stored in a variable or slot of type OBJECT. Similarly, it unwraps wrapped literals automatically to operate on the literal directly. This is very convenient, since it relieves the programmer from having to perform these conversions by hand and makes the code less cluttered. For example, consider the following code fragment:

```lisp
(let ((l (cons "foo" nil))
   (x (concatenate "bar" (first l))))
 (print x EOL)))
```

Here is its C++ translation:

```cpp
{ Cons* l = cons(wrapString("foo"), NIL);
  char* x = stringConcatenate
    ("bar", ((StringWrapper*)(l->value))->wrapperValue, 0);

  std::cout << x << std::endl;
```
Notice how the string literal \"foo\" is first wrapped so it can be inserted into the CONS list and then automatically unwrapped in the call to concatenate. While this is very convenient, it does cause a certain overhead that should be avoided in performance critical loops, etc. In such situations, it often helps to use auxiliary variables of the appropriate literal type to avoid unnecessary wrap/unwrap operations.

**Lisp-style property lists:** Lisp programs often use property lists for fast retrieval of information that is linked to symbols. To support the easy translation of existing Lisp programs that use this paradigm into STELLA, a similar mechanism implemented by the functions `symbol-value`, `symbol-plist`, and `symbol-property` is available that preserves the performance benefits of this storage scheme (see the file `sources/stella/symbols.ste`). However, property lists do not fit the object-oriented programming paradigm supported by STELLA, and, hence, are frowned upon.

**Compiler optimization:** The optimization settings used with the native Lisp or C++ compiler can greatly influence performance results. In particular, using high optimization settings with the Lisp compiler can greatly improve slot access time on STELLA objects.

### 4.3.1 Lisp Performance Hints

The standard Lisp implementation for STELLA objects are CLOS objects, since CLOS provides the most natural Lisp implementation for the STELLA object system. However, there is a price to pay, since in Lisp slot access on CLOS objects is a lot slower than slot access on structs. For example, in Allegro CL 4.3, the access to the `value` slot of a STELLA CONS cell takes about 4 times longer on a CLOS object implementation of CONS than on a struct implementation. Unfortunately, the struct implementation itself takes about 3 times longer than calling `CL:car` on a Lisp cons, which is why we are actually using Lisp conses as the Lisp implementation for STELLA CONSes. Note, that in the C++ and Java translation these slot-access performance problems are nonexistent.

In order to get the maximum performance out of the Lisp version of STELLA, you can tell the translator to use structs as the implementation for STELLA objects. It does so by using `CL:defstruct` instead of `CL:defclass` and dispatches methods directly on the structure object.

To use the struct translation scheme evaluate

```lisp
(set-stella-feature :use-common-lisp-structs)
```

before you translate a STELLA system. This will generate translated files with a .slisp extension. Make sure that after you translated all the files you are interested in, you disable the above feature with

```lisp
(unset-stella-feature :use-common-lisp-structs)
```

Otherwise, subsequent incremental translations in that Lisp image might fail, since different translation schemes cannot be mixed. If you already are using the struct version of STELLA, all systems will be translated in struct mode by default.

To use the struct translation of your system you have to use the struct version of STELLA. To do so do the following:
(CL:setq cl-user::*load-cl-struct-stella?* CL:t)
(CL:load "load-stella.lisp")

Alternatively, you can edit the initial value of the variable *load-cl-struct-stella?* in the file ‘load-stella.lisp’ (see also Section 2.3 [Lisp Installation], page 4).

The reasons why the struct translation scheme is not enabled by default are the following:

- Incremental redefinition of STELLA classes does not redefine any objects created with the old definition, and, hence, slot accessors might simply break or retrieve the value of a different slot when applied to such an old object. The programmer therefore has to be very careful when redefining a STELLA class while in struct mode. This means, that you should view the usage of the struct-translation scheme for Lisp as a kind of delivery option, similar to translating into C++. Part of the reason why slot access on CLOS object is expensive is the indirection machinery that allows redefinition of classes and their associated instances. This is great for code development, but the flexibility and expense is usually not needed or warranted for delivered code.

- The performance trade-offs between CLOS and struct versions might be different in different versions of Lisp. For example, in older version of Allegro CL slot access on structs was fast, but method dispatch was significantly slower than for CLOS objects which eliminated some/all of the performance gains.
5 Library Classes (tbw)

To be written.
6 Library Functions

6.1 Basic Constants and Predicates

true
Not yet implemented.

false
Not yet implemented.

null? (x) : BOOLEAN [Method on OBJECT]
Return true if x is undefined (handled specially by all translators).

null? (x) : BOOLEAN [Method on SECOND-CLASS-OBJECT]
Return true if x is undefined (handled specially by all translators).

null? (x) : BOOLEAN [Method on NATIVE-VECTOR]
Return true if x is undefined (handled specially by all translators).

null? (x) : BOOLEAN [Method on STRING]
Return true if x is undefined (handled specially by all translators).

null? (x) : BOOLEAN [Method on MUTABLE-STRING]
Return true if x is undefined (handled specially by all translators).

null? (x) : BOOLEAN [Method on CHARACTER]
Return true if x is undefined (handled specially by all translators).

null? (x) : BOOLEAN [Method on CODE]
Return true if x is undefined (handled specially by all translators).

null? (x) : BOOLEAN [Method on INTEGER]
Return true if x is undefined (handled specially by all translators).

null? (x) : BOOLEAN [Method on FLOAT]
Return true if x is undefined (handled specially by all translators).

defined? (x) : BOOLEAN [Method on OBJECT]
Return true if x is defined (handled specially by all translators).

defined? (x) : BOOLEAN [Method on SECOND-CLASS-OBJECT]
Return true if x is defined (handled specially by all translators).

defined? (x) : BOOLEAN [Method on NATIVE-VECTOR]
Return true if x is defined (handled specially by all translators).

defined? (x) : BOOLEAN [Method on STRING]
Return true if x is defined (handled specially by all translators).

defined? (x) : BOOLEAN [Method on MUTABLE-STRING]
Return true if x is defined (handled specially by all translators).
**Chapter 6: Library Functions**

### Method on Character

**defined?** (x) : BOOLEAN

[Method on CHARACTER]
Return true if x is defined (handled specially by all translators).

### Method on Code

**defined?** (x) : BOOLEAN

[Method on CODE]
Return true if x is defined (handled specially by all translators).

### Method on Integer

**defined?** (x) : BOOLEAN

[Method on INTEGER]
Return true if x is defined (handled specially by all translators).

### Method on Float

**defined?** (x) : BOOLEAN

[Method on FLOAT]
Return true if x is defined (handled specially by all translators).

### Function

**eq?** ((x UNKNOWN) (y UNKNOWN)) : BOOLEAN

[Function]
Return true if x and y are literally the same object (or simple number). Analogue to the Common Lisp EQL and C++ and Java’s ==.

**eql?** ((x OBJECT) (y OBJECT)) : BOOLEAN

[Function]
Return true if x and y are eq? or equivalent literals such as strings that also might be wrapped in non-identical wrappers. For the case where x or y are plain literals such as strings or integers, the STELLA translator substitutes the equality test appropriate for the particular target language and does not actually call this function. For cases where x or y are known to be of type STANDARD-OBJECT, the STELLA translator substitutes the faster eq? test inline.

**equal?** ((x OBJECT) (y OBJECT)) : BOOLEAN

[Function]
Return true if x and y are eql? or considered equal by a user-defined object-equal? method. This implements a fully extensible equality test similar to Java’s equals method. Note that writers of custom object-equal? methods must also implement a corresponding equal-hash-code method.

**object-equal?** (x (y OBJECT)) : BOOLEAN

[Method on OBJECT]
Return true if x and y are eq?.

**object-equal?** (x (y OBJECT)) : BOOLEAN

[Method on WRAPPER]
Return true if x and y are literal wrappers whose literals are considered eql?.

### 6.2 Numbers

**pi** : FLOAT

[Constant]
A float approximation of the mathematical constant pi.

**+** (&rest (arguments NUMBER)) : NUMBER

[Function]
Return the sum of all arguments.

**-** ((x NUMBER) &rest (arguments NUMBER)) : NUMBER

[Function]
If only x was supplied return the result of 0 - x. Otherwise, return the result of (...((x - arg1) - arg2) - ... - argN).

**\*** (&rest (arguments NUMBER)) : NUMBER

[Function]
Return the product of all arguments.
/ ((x NUMBER) &rest (arguments NUMBER)) : NUMBER
    [Function]
    If only x was supplied return the result of 1 / x. Otherwise, return the result of (...((x / arg1) / arg2) / ... / argN).

1+ ((expression OBJECT)) : OBJECT
    [Macro]
    Add 1 to expression and return the result.

1- ((expression OBJECT)) : OBJECT
    [Macro]
    Subtract 1 from expression and return the result.

++ ((place OBJECT) &body (increment CONS)) : OBJECT
    [Macro]
    Increment the value of place and return the result. place can be either a variable name or a slot reference. Increment by the optional increment (which can be a float) or 1 otherwise.

-- ((place OBJECT) &body (decrement CONS)) : OBJECT
    [Macro]
    Decrement the value of place and return the result. place can be either a variable name or a slot reference. Decrement by the optional decrement (which can be a float) or 1 otherwise.

= ((x NUMBER) (y NUMBER)) : BOOLEAN
    [Function]
    Return true if x and y are numbers of exactly the same magnitude.

< ((x NUMBER) (y NUMBER)) : BOOLEAN
    [Function]
    Return true if x is less than y.

<= ((x NUMBER) (y NUMBER)) : BOOLEAN
    [Function]
    Return true if x is less than or equal to y.

>= ((x NUMBER) (y NUMBER)) : BOOLEAN
    [Function]
    Return true if x is greater than or equal to y.

> ((x NUMBER) (y NUMBER)) : BOOLEAN
    [Function]
    Return true if x is greater than y.

zero?
    [???]
    Not yet implemented.

plus?
    [???]
    Not yet implemented.

even?
    [???]
    Not yet implemented.

odd?
    [???]
    Not yet implemented.

div
    [???]
    Not yet implemented.

rem
    [???]
    Not yet implemented.
Chapter 6: Library Functions

**mod**
Not yet implemented.

**gcd** ((x LONG-INTEGER) (y LONG-INTEGER)) : LONG-INTEGER
Return the greatest common divisor of x and y.

**ceiling** ((n NUMBER)) : INTEGER
Return the smallest integer >= n.

**floor** ((n NUMBER)) : INTEGER
Return the biggest integer <= n.

**round** ((n NUMBER)) : INTEGER
Round n to the closest integer and return the result.

**abs** (x) : INTEGER
Return the absolute value of x.

**abs** (x) : FLOAT
Return the absolute value of x.

**min**
Not yet implemented.

**max**
Not yet implemented.

**sqrt** ((n FLOAT)) : FLOAT
Return the square root of n.

**exp** ((n FLOAT)) : FLOAT
Return the e to the power n.

**expt** ((x FLOAT) (y FLOAT)) : FLOAT
Return x ^ y.

**log** ((n FLOAT)) : FLOAT
Return the natural logarithm (base e) of n.

**log10** ((n FLOAT)) : FLOAT
Return the logarithm (base 10) of n.

**sin** ((n FLOAT)) : FLOAT
Return the sine of n radians.

**cos** ((n FLOAT)) : FLOAT
Return the cosine of n radians.

**tan** ((n FLOAT)) : FLOAT
Return the tangent of n radians.

**asin** ((n FLOAT)) : FLOAT
Return the arcsine of n in radians.
acos (\((n \text{ float})\)) : \text{float}
Return the arccosine of \(n\) in radians.

atan (\((n \text{ float})\)) : \text{float}
Return the arc tangent of \(n\) in radians.

atan2 (\((x \text{ float}) (y \text{ float})\)) : \text{float}
Return the arc tangent of \(x / y\) in radians.

random (\((n \text{ integer})\)) : \text{integer}
Generate a random integer in the interval [0..n-1].

integer-to-string (\((i \text{ long-integer})\)) : \text{string}
Convert \(i\) to its string representation and return the result. This is more efficient than using a string stream.

string-to-integer (\((\text{string string})\)) : \text{long-integer}
Convert a \text{string} representation of an integer into an integer.

float-to-string (\((f \text{ float})\)) : \text{string}
Convert \(f\) to its string representation and return the result. This is more efficient than using a string stream.

string-to-float (\((\text{string string})\)) : \text{float}
Convert a \text{string} representation of a float into a float.

format-float (\((f \text{ float}) (nDecimals \text{ integer})\)) : \text{string}
Print \(f\) in fixed-point format with \(n\)Decimals behind the decimal point and return the result as a string.

wrap-integer (\((\text{value integer})\)) : \text{integer-wrapper}
Return a literal object whose value is the \text{INTEGER} value.

unwrap-integer (\((\text{wrapper integer-wrapper})\)) : \text{integer}
Unwrap \text{wrapper} and return the result. Return NULL if \text{wrapper} is NULL.

wrap-float (\((\text{value float})\)) : \text{float-wrapper}
Return a literal object whose value is the \text{FLOAT} value.

unwrap-float (\((\text{wrapper float-wrapper})\)) : \text{float}
Unwrap \text{wrapper} and return the result. Return NULL if \text{wrapper} is NULL.

6.3 Characters

character-code (\((ch \text{ character})\)) : \text{integer}
Return the 8-bit ASCII code of \(ch\) as an integer.

code-character (\((code \text{ integer})\)) : \text{character}
Return the character encoded by \(code\) (0 <= \(code\) <= 255).

digit-character? (\((ch \text{ character})\)) : \text{boolean}
Return TRUE if \(ch\) represents a digit.
letter-character? ((ch CHARACTER)) : BOOLEAN
Return TRUE if ch represents a letter.

upper-case-character? ((ch CHARACTER)) : BOOLEAN
Return TRUE if ch represents an upper-case character.

lower-case-character? ((ch CHARACTER)) : BOOLEAN
Return TRUE if ch represents a lower-case character.

white-space-character? ((ch CHARACTER)) : BOOLEAN
Return TRUE if ch is a white space character.

color-downcase ((ch CHARACTER)) : CHARACTER
If ch is lowercase, return its uppercase version, otherwise, return ch unmodified.

color-upcase ((ch CHARACTER)) : CHARACTER
If ch is uppercase, return its lowercase version, otherwise, return ch unmodified. If only
the first character of a sequence of characters is to be capitalized, color-capitalize should be used instead.

color-capitalize ((ch CHARACTER)) : CHARACTER
Return the capitalized character for ch. This is generally the same as the uppercase
color, except for obscure non-English characters in Java. It should be used if only
the first character of a sequence of characters is to be capitalized.

color-to-string ((c CHARACTER)) : STRING
Convert c into a one-element string and return the result.

wrap-character ((value CHARACTER)) : CHARACTER-WRAPPER
Return a literal object whose value is the CHARACTER value.

unwrap-character ((wrapper CHARACTER-WRAPPER)) : CHARACTER
Unwrap wrapper and return the result. Return NULL if wrapper is NULL.

6.4 Strings

string-eql? ((x STRING) (y STRING)) : BOOLEAN
Return true if x and y are equal strings or are both undefined. This test is substituted
automatically by the STELLA translator if eql? is applied to strings.

string-equal? ((x STRING) (y STRING)) : BOOLEAN
Return true if x and y are equal strings ignoring character case or are both undefined.

empty? (x) : BOOLEAN
Return true if x is the empty string ""

non-empty? (x) : BOOLEAN
Return true if x is not the empty string ""
Chapter 6: Library Functions

string-compare ((x STRING) (y STRING) (case-sensitive? BOOLEAN)) : [Function]
    INTEGER
    Compare x and y lexicographically, and return -1, 0, or 1, depending on whether x is less than, equal, or greater than y. If case-sensitive? is true, then case does matter for the comparison.

string< ((x STRING) (y STRING)) : BOOLEAN [Function]
    Return true if x is lexicographically < y, considering case.

string<= ((x STRING) (y STRING)) : BOOLEAN [Function]
    Return true if x is lexicographically <= y, considering case.

string>= ((x STRING) (y STRING)) : BOOLEAN [Function]
    Return true if x is lexicographically >= y, considering case.

string> ((x STRING) (y STRING)) : BOOLEAN [Function]
    Return true if x is lexicographically > y, considering case.

string-less? ((x STRING) (y STRING)) : BOOLEAN [Function]
    Return true if x is lexicographically < y, ignoring case.

string-less-equal? ((x STRING) (y STRING)) : BOOLEAN [Function]
    Return true if x is lexicographically <= y, ignoring case.

string-greater-equal? ((x STRING) (y STRING)) : BOOLEAN [Function]
    Return true if x is lexicographically >= y, ignoring case.

string-greater? ((x STRING) (y STRING)) : BOOLEAN [Function]
    Return true if x is lexicographically > y, ignoring case.

all-upper-case-string? ((s STRING)) : BOOLEAN [Function]
    Return TRUE if all letters in s are upper case.

all-lower-case-string? ((s STRING)) : BOOLEAN [Function]
    Return TRUE if all letters in s are lower case.

make-string ((size INTEGER) (initchar CHARACTER)) : STRING [Function]
    Return a new string filled with size initchars.

make-mutable-string ((size INTEGER) (initchar CHARACTER)) : [Function]
    MUTABLE-STRING
    Return a new mutable string filled with size initchars.

make-raw-mutable-string ((size INTEGER)) : MUTABLE-STRING [Function]
    Return a new uninitialized mutable string of size.

first (self) : CHARACTER [Method on STRING]
    Return the first character of self.

first (self) : CHARACTER [Method on MUTABLE-STRING]
    Return the first character of self (settable via setf).
second (self) : CHARACTER
Return the second character of self.

second (self) : CHARACTER
Return the second character of self (settable via self).

third (self) : CHARACTER
Return the third character of self.

third (self) : CHARACTER
Return the third character of self (settable via self).

fourth (self) : CHARACTER
Return the fourth character of self.

fourth (self) : CHARACTER
Return the fourth character of self (settable via self).

fifth (self) : CHARACTER
Return the fifth character of self.

fifth (self) : CHARACTER
Return the fifth character of self (settable via self).

nth (self (position integer)) : CHARACTER
Return the character in self at position.

nth (self (position integer)) : CHARACTER
Return the character in self at position.

rest (self) : STRING
Not documented.

length (self) : INTEGER
Return the length of the string self.

length (self) : INTEGER
Return the length of the string self.

member? (self (char character)) : BOOLEAN
Not documented.

position (string (character character) (start integer)) :
integer
Return the position of character within string (counting from zero); or return NULL
if character does not occur within string. If start was supplied as non-NULL, only
consider the substring starting at start, however, the returned position will always be
relative to the entire string.
last-position (string (character CHARACTER) (end integer)) : integer
Return the last position of character within string (counting from zero); or return NULL if character does not occur within string. If end was supplied as non-NULL, only consider the substring ending at end, however, the returned position will always be relative to the entire string.

string-search ((string string) (substring string) (start integer)) : integer
Return start position of the left-most occurrence of substring in string, beginning from start. Return NULL if it is not a substring.

copy (string) : string
Return a copy of string.

string-upcase ((string string)) : string
Return an upper-case copy of string.

string-downcase ((string string)) : string
Return a lower-case copy of string.

string-capitalize ((string string)) : string
Return a capitalized version of string.

concatenate (string1 (string2 string) &rest (otherStrings string)) : string
Return a new string representing the concatenation of string1, string2, and otherStrings. The two mandatory parameters allow us to optimize the common binary case by not relying on the somewhat less efficient variable arguments mechanism.

subsequence (string (start integer) (end integer)) : string
Return a substring of string beginning at position start and ending up to but not including position end, counting from zero. An end value of NULL stands for the rest of the string.

remove (string (char CHARACTER)) : string
Remove all occurrences of char from string.

substitute (self (new-char CHARACTER) (old-char CHARACTER)) : string
Substitute all occurrences of old-char with new-char in the string self.

substitute (self (new-char CHARACTER) (old-char CHARACTER)) : mutable-string
Substitute all occurrences of old-char with new-char in the string self.

replace-substrings ((string string) (new string) (old string)) : string
Replace all occurrences of old in string with new.
**instantiate-string-template** ((template string) &rest (vars&values string)) : string
For each occurrence of a <var> string from vars&values in template replace it with its corresponding <value> string. Replacement is done in sequence which means (part of) a value might be replaced further with a later <var> and <value>.

**insert-string** ((source string) (start integer) (end integer) (target mutable-string) (target-index integer) (case-conversion keyword)) : integer
Inserts characters from source beginning at start and ending at end into target starting at target-index. If end is null, then the entire length of the string is used. The copy of characters is affected by the case-conversion keyword which should be one of :UPCASE :DOWNCASE :CAPITALIZE :PRESERVE.
The final value of target-index is returned.

**wrap-string** (((value string)) : STRING-WRAPPER
Return a literal object whose value is the STRING value.

**wrap-mutable-string** (((value mutable-string)) :
Return a literal object whose value is the MUTABLE-STRING value.

**unwrap-string** (((wrapper string-wrapper)) : STRING
Unwrap wrapper and return the result. Return NULL if wrapper is NULL.

**unwrap-mutable-string** (((wrapper mutable-string-wrapper)) :
Unwrap wrapper and return the result. Return NULL if wrapper is NULL.

**string-to-mutable-string** (((s string)) : MUTABLE-STRING
Copy s into a mutable string with the same content. In Lisp and C++ this simply copies s.

**mutable-string-to-string** (((s mutable-string)) : string
Convert s into a regular string with the same content. In Lisp and C++ this is a no-op.

**integer-to-string** (((i long-integer)) : string
Convert i to its string representation and return the result. This is more efficient than using a string stream.

**string-to-integer** (((string string)) : LONG-INTEGER
Convert a string representation of an integer into an integer.

**float-to-string** (((f float)) : string
Convert f to its string representation and return the result. This is more efficient than using a string stream.

**string-to-float** (((string string)) : float
Convert a string representation of a float into a float.
### Chapter 6: Library Functions

#### Function: `format-float` ((f float) (nDecimals integer)) : STRING
Print `f` in fixed-point format with `nDecimals` behind the decimal point and return the result as a string.

#### Function: `character-to-string` ((c character)) : STRING
Convert `c` into a one-element string and return the result.

#### Function: `stringify` ((expression object)) : STRING
Print `expression` onto a string and return the result. Printing is done with `*printReadably?*` set to true and with `*printPretty?*` set to false.

#### Function: `stringify-in-module` ((tree object) (module module)) : STRING
Stringify a parse `tree` relative to `module`, or `*module*` if no module is specified.

#### Function: `unstringify` ((string string)) : object
Read a STELLA expression from `string` and return the result. This is identical to `read-s-expression-from-string`.

#### Function: `unstringify-in-module` ((string string) (module module)) : object
Unstringify relative to `module`, or `*MODULE*` if no module is specified.

### 6.5 CONS Lists and Trees

#### Class: `CONS` : STANDARD-OBJECT
Not documented.

- **any-value** : OBJECT
  Not documented.

- **value** : (LIKE (ANY-VALUE SELF))
  Not documented.

- **rest** : (CONS OF (LIKE (ANY-VALUE SELF)))
  Not documented.

#### Variable: `nil` : CONS
Not documented.

- **empty?** (self) : BOOLEAN
  Return `true` iff `self` equals `nil`.

- **non-empty?** (self) : BOOLEAN
  Return `true` iff `self` is not equal to `nil`.

- **nil?** ((x object)) : BOOLEAN
  Return `true` iff `x` equals `nil`.

- **equal-cons-trees?** ((tree1 object) (tree2 object)) : BOOLEAN
  Return `true` iff the cons trees `tree1` and `tree2` are structurally equivalent. Uses an `eql?` test.
Chapter 6: Library Functions

**object-equal?** \((tree1 (tree2 object))\) : BOOLEAN

Return **true** iff the cons trees \(tree1\) and \(tree2\) are structurally equivalent. Uses \(\text{equal?}\) to test equality of subtrees.

**equal-hash-code** \((self)\) : INTEGER

Return an **equal?** hash code for \(self\). Note that this is \(O(N)\) in the number of elements of \(self\).

**cons** \(((value \text{object}) (rest \text{cons}))\) : CONS

Return a cons record that points to \(value\) and \(rest\).

**first** \((self)\) : \(\text{LIKE (ANY-VALUE SELF)}\)

Return the first element of \(self\). The first element of \(self\) can be set with **setf**. Note that \((\text{first NIL}) = \text{null}\).

**second** \((self)\) : \(\text{LIKE (ANY-VALUE SELF)}\)

Return the second element of \(self\). The second element of \(self\) can be set with **setf**. Note that \((\text{second NIL}) = \text{null}\).

**third** \((self)\) : \(\text{LIKE (ANY-VALUE SELF)}\)

Return the third element of \(self\). The third element of \(self\) can be set with **setf**. Note that \((\text{third NIL}) = \text{null}\).

**fourth** \((self)\) : \(\text{LIKE (ANY-VALUE SELF)}\)

Return the fourth element of \(self\). The fourth element of \(self\) can be set with **setf**. Note that \((\text{fourth NIL}) = \text{null}\).

**fifth** \((self)\) : \(\text{LIKE (ANY-VALUE SELF)}\)

Return the fifth element of \(self\). The fifth element of \(self\) can be set with **setf**. Note, that \((\text{fifth NIL}) = \text{null}\).

**nth** \((self \text{position integer})\) : \(\text{LIKE (ANY-VALUE SELF)}\)

Return the element of \(self\) at position. The nth element of \(self\) can be set with **setf**. Note, that \((\text{nth NIL <pos>} = \text{null}\).

**nth-rest** \((self \text{position integer})\) : \(\text{LIKE SELF}\)

Apply \(rest\) position times to \(self\).

**last** \((self)\) : \(\text{LIKE (ANY-VALUE SELF)}\)

Return the last element of \(self\).

**but-last** \((self)\) : \(\text{ITERATOR OF (LIKE (ANY-VALUE SELF))}\)

Generate all but the last element of the cons list \(self\).

**last-cons** \((self \text{cons})\) : \(\text{CONS OF (LIKE (ANY-VALUE SELF))}\)

Return the last cons of \(self\).

**length** \((self)\) : INTEGER

Return the length of the CONS list \(self\).

**member?** \((self \text{object object})\) : BOOLEAN

Return **true** iff \(object\) is a member of the cons list \(self\) (uses an \(\text{eq1?}\) test).
memb? (self (object object)) : BOOLEAN

Return true iff object is a member of the cons list self (uses an eq? test).

position (self (object object) (start integer)) : INTEGER

Return the position of object within the cons-list self (counting from zero); or return null if object does not occur within self (uses an eql? test). If start was supplied as non-null, only consider the sublist starting at start, however, the returned position will always be relative to the entire list.

last-position (self (object object) (end integer)) : INTEGER

Return the position of object within the cons-list self (counting from zero); or return null if object does not occur within self (uses an eql? test). If start was supplied as non-null, only consider the sublist ending at end, however, the returned position will always be relative to the entire list.

reverse (self) : (LIKE SELF)

Destructively reverse the members of the cons list self.

remove (self (value object)) : (LIKE SELF)

Destructively remove all entries in the cons list self that match value. Unless the remaining list is null, insures that the cons that heads the list is unchanged.

remove-duplicates (self) : (LIKE SELF)

Destructively remove duplicates from self and return the result. Removes all but the first occurrence of items in the list. Preserves the original order of the remaining members. Runs in linear time.

remove-if (self (test? function-code)) : (LIKE SELF)

Destructively removes all members of the cons list self for which test? evaluates to true. test? takes a single argument of type OBJECT and returns true or false. Returns a cons list. In case the first element is removed, the return result should be assigned to a variable.

substitute (self (inValue object) (outValue object)) : CONS

Destructively replace each appearance of outValue by inValue in the cons list self.

concatenate (list1 (list2 cons) &rest (otherLists cons)) : CONS

Return a cons list consisting of the concatenation of list1, list2, and otherLists. The operation is destructive wrt all but the last list argument which is left intact. The two mandatory parameters allow us to optimize the common binary case by not relying on the somewhat less efficient variable arguments mechanism.

append ((consList1 cons) (consList2 cons)) : CONS

Return a cons list representing the concatenation of consList1 and consList2. The concatenation is NOT destructive.

prepend (self (list1 cons)) : CONS

Return a cons list consisting of the concatenation of list1 and self. A copy of list1 is prepended to self. This operation results in structure sharing of self; to avoid this, self should not be pointed to by anything other than the tail of the prepended copy.
pushq ((variable SYMBOL) (value OBJECT)) : OBJECT
   Push value onto the cons list variable.

pushq-new ((variable SYMBOL) (value OBJECT)) : OBJECT
   Push value onto the cons list variable, unless value is already a member of the list.

popq ((variable SYMBOL)) : OBJECT
   Pops a value from the cons list variable.

cons-list (&rest (values OBJECT)) : CONS
   Return a cons list containing values, in order.

list* (&rest (values OBJECT)) : CONS
   Return a list of conses that make up the list values, terminated by the last value rather than by nil. Assumes that at least one value is passed in.

copy-cons-list ((self CONS)) : (LIKE SELF)
   Return a copy of the cons list self.

copy-cons-tree ((self OBJECT)) : (LIKE SELF)
   Return a copy of the cons tree self.

substitute-cons-tree ((tree OBJECT) (newValue OBJECT) (oldValue OBJECT)) : OBJECT
   Destructively replace each appearance of oldValue by newValue in the cons tree tree. Return the tree. Uses an eql? test.

search-cons-tree? ((tree OBJECT) (value OBJECT)) : BOOLEAN
   Return true iff the value value is embedded within the cons tree tree. Uses an eql? test.

tree-size ((self OBJECT)) : INTEGER
   Not documented.

safe-tree-size ((tree CONS)) : INTEGER STRING
   Not documented.

cons-tree-nth ((tree CONS) &rest (index INTEGER)) : OBJECT
   Access an arbitrary element of tree identified by a path specified as a list of index values. The first index specifies the index-th element of tree, the second index the index-th subelement of that element, etc. Example:
   (cons-tree-nth (quote (a (b (c d e) f) g)) 1 1 2) => e

cons-tree-nth-rest ((tree CONS) &rest (index INTEGER)) : CONS
   Access an arbitrary sublist of tree identified by a path specified as a list of index values. The first index specifies the index-th element of tree, the second index the index-th subelement of that element, ..., the last index specifies the nth-rest of the previous element (different from cons-tree-nth). Example:
   (cons-tree-nth-rest (quote (a (b (c d e) f) g)) 1 1 1) => (d e)
Chapter 6: Library Functions

**match-cons-tree** ((tree object) (pattern object) (bindings key-value-list)) : key-value-list

Match pattern against tree and return a list of variable bindings if they match, or NULL otherwise. bindings can be NULL or an initial list of bindings to consider. Pattern variables use KIF syntax, i.e., they need to start with a ? character. A single question mark is interpreted as the anonymous variable. Example:

\[
\text{(match-cons-tree (quote (a (b (a d) e) (a d) f g)) (quote (a (?x ?y ?) ?y ? g)) NULL)} \\
\Rightarrow \text{|kv|([<?Y,(A D)> <?X,B>])}
\]

Variables can’t be quoted but quoting can effectively be achieved by inserting to-be-quoted variables bound to themselves into bindings.

**cons-tree-match?** ((tree object) (pattern object)) : boolean

Predicate version of match-cons-tree (which see).

**consify** (self) : (cons of (like (any-value self)))

Return self.

**allocate-iterator** (self) : (cons-iterator of (like (any-value self)))

Not documented.

**next?** (self) : boolean

Not documented.

**sort** (self (predicate function-code)) : (cons of (like (any-value self)))

Perform a stable, destructive sort of self according to predicate, and return the result. If predicate has a < semantics, the result will be in ascending order. It is not guaranteed that self will point to the beginning of the sorted result. If predicate is null, a suitable < predicate is chosen depending on the first element of self, and it is assumed that all elements of self have the same type (supported element types are GENERALIZED-SYMBOL, STRING, INTEGER, and FLOAT).

**sort-tuples** (self (n integer) (predicate function-code)) : (cons of (like (any-value self)))

Just like sort but assumes each element of self is a tuple (a cons) whose n-th element (0-based) will be used for comparison.

**map-null-to-nil** ((self cons)) : (like self)

Return nil iff self is null or self otherwise.

**printpretty?** : boolean

If true conses will be pretty printed.

**printreadably?** : boolean

If true conses will be printed as readable Stella code.
*printprettycode?* : BOOLEAN [Special Variable]

When true pretty-print Stella and translated code. Since (Lisp) pretty-printing is somewhat slow, turning this off speeds up file translation, but it also makes translated output very unreadable.

6.5.1 CONS Lists as Sets

subset? (self (otherList cons)) : BOOLEAN [Method on CONS]

Return true if every element of self also occurs in otherList. Uses an eql? test and a simple quadratic-time algorithm. Note that this does not check whether self and otherList actually are sets.

equivalent-sets? (self (otherList cons)) : BOOLEAN [Method on CONS]

Return true if every element of self occurs in otherList and vice versa. Uses an eql? test and a simple quadratic-time algorithm. Note that this does not check whether self and otherList actually are sets.

union (self (otherList cons)) : CONS [Method on CONS]

Return the set union of self and otherList. Uses an eql? test and a simple quadratic-time algorithm. Note that the result is only guaranteed to be a set if both self and otherList are sets.

intersection (self (otherList cons)) : CONS [Method on CONS]

Return the set intersection of self and otherList. Uses an eql? test and a simple quadratic-time algorithm. Note that the result is only guaranteed to be a set if both self and otherList are sets.

difference (self (otherList cons)) : CONS [Method on CONS]

Return the set difference of self and otherList (i.e., all elements that are in self but not in otherSet). Uses an eql? test and a simple quadratic-time algorithm. Note that the result is only guaranteed to be a set if both self and otherList are sets.

subtract (self (otherList cons)) : CONS [Method on CONS]

Return the set difference of self and otherList by destructively removing elements from self that also occur in otherList. Uses an eql? test and a simple quadratic-time algorithm. Note that the result is only guaranteed to be a set if self is a set.

6.6 Lists

LIST : SEQUENCE [Class]

Not documented.

any-value : OBJECT [Class Parameter of ]

Not documented.

the-cons-list : (CONS OF (LIKE (ANY-VALUE SELF))) [Slot of ]

Not documented.
Chapter 6: Library Functions

nil-list : LIST
   Not documented. [Variable]

defined-list? ((self LIST)) : BOOLEAN
   Return TRUE unless self is NULL or the NIL-LIST. [Function]

null-list? ((self LIST)) : BOOLEAN
   Return TRUE iff self is NULL or the NIL-LIST. [Function]

empty? (self) : BOOLEAN
   Return TRUE if the list self has no members. [Method on LIST]

non-empty? (self) : BOOLEAN
   Return TRUE if the list self has at least one member. [Method on LIST]

object-equal? (x (y OBJECT)) : BOOLEAN
   Return TRUE iff the lists x and y are structurally equivalent. Uses equal? to test
   equality of elements. [Method on LIST]

equal-hash-code (self) : INTEGER
   Return an equal? hash code for self. Note that this is O(N) in the number of elements
   of self. [Method on LIST]

list (&rest (values OBJECT)) : LIST
   Return a list containing values, in order. [Function]

first (self) : (LIKE (ANY-VALUE SELF))
   Return the first item in the list self, or NULL if empty. [Method on LIST]

second (self) : (LIKE (ANY-VALUE SELF))
   Return the second item in the list self, or NULL if empty. [Method on LIST]

third (self) : (LIKE (ANY-VALUE SELF))
   Return the third item in the list self, or NULL if empty. [Method on LIST]

fourth (self) : (LIKE (ANY-VALUE SELF))
   Return the fourth item in the list self, or NULL if empty. [Method on LIST]

fifth (self) : (LIKE (ANY-VALUE SELF))
   Return the fifth item in the list self, or NULL if empty. [Method on LIST]

nth (self (position INTEGER)) : (LIKE (ANY-VALUE SELF))
   Return the nth item in the list self, or NULL if empty. [Method on LIST]

rest (self) : (CONS OF (LIKE (ANY-VALUE SELF)))
   Return a cons list of all but the first item in the list self. [Method on LIST]

last (self) : (LIKE (ANY-VALUE SELF))
   Return the last element of self. [Method on LIST]

but-last (self) : (ITERATOR OF (LIKE (ANY-VALUE SELF)))
   Generate all but the last element of the list self. [Method on LIST]
**length** *(self)* : INTEGER
Not documented.

**member?** *(self (object object))* : BOOLEAN
[Method on LIST]
Return TRUE iff object is a member of the list self (uses an eql? test).

**memb?** *(self (object (like (any-value self))))* : BOOLEAN
[Method on LIST]
Return TRUE iff object is a member of the cons list self (uses an eq? test).

**position** *(self (object object) (start integer))* : INTEGER
[Method on LIST]
Return the position of object within the list self (counting from zero); or return NULL if object does not occur within self (uses an eql? test). If start was supplied as non-NULL, only consider the sublist starting at start, however, the returned position will always be relative to the entire list.

**last-position** *(self (object object) (end integer))* : INTEGER
[Method on LIST]
Return the position of object within the list self (counting from zero); or return NULL if object does not occur within self (uses an eql? test). If end was supplied as non-NULL, only consider the sublist ending at end, however, the returned position will always be relative to the entire list.

**insert** *(self (value (like (any-value self))))* :
[Method on LIST]
Add value to the front of the list self.

**push** *(self (value (like (any-value self))))* :
[Method on LIST]
Add value to the front of the list self.

**insert-new** *(self (value (like (any-value self))))* :
[Method on LIST]
Add value to the front of the list self unless its already a member.

**insert-last** *(self (value (like (any-value self))))* :
[Method on LIST]
Insert value as the last entry in the list self.

**reverse** *(self)* : (like self)
[Method on LIST]
Reverse the members of self (in place).

**remove** *(self (value (like (any-value self))))* : (like self)
[Method on LIST]
Destructively remove all entries in self that match value.

**remove-duplicates** *(self)* : (like self)
[Method on LIST]
Destructively remove duplicates from self and return the result. Preserves the original order of the remaining members.

**remove-deleted-members** *(self)* : (like self)
[Method on LIST]
Not documented.

**remove-if** *(self (test? function-code))* : (like self)
[Method on LIST]
Destructively remove all members of the list self for which test? evaluates to TRUE. test takes a single argument of type OBJECT and returns TRUE or FALSE. Returns self.
pop (self) : (LIKE (ANY-VALUE SELF))  [Method on LIST]
    Remove and return the first element in the list self. Return NULL if the list is empty.

substitute (self (inValue OBJECT) (outValue OBJECT)) : (LIKE self)  [Method on LIST]
    Destructively replace each appearance of outValue by inValue in the list self.

concatenate (list1 (list2 LIST) &rest (otherLists LIST)) : LIST  [Method on LIST]
    Copy list2 and all otherLists onto the end of list1. The operation is destructive wrt list1, but leaves all other lists intact. The two mandatory parameters allow us to optimize the common binary case by not relying on the somewhat less efficient variable arguments mechanism.

prepend (self (list2 LIST)) : (LIKE self)  [Method on LIST]
    Copy list2 onto the front of the list self. The operation is destructive wrt self, but leaves list2 intact.

copy (self) : (LIST OF (LIKE (ANY-VALUE SELF)))  [Method on LIST]
    Return a copy of the list self. The conses in the copy are freshly allocated.

clear (self) :  [Method on LIST]
    Make self an empty list.

consify (self) : (CONS OF (LIKE (ANY-VALUE SELF)))  [Method on LIST]
    Return a list of elements in self.

allocate-iterator (self) : (LIST-ITERATOR OF (LIKE (ANY-VALUE self)))  [Method on LIST]
    Not documented.

next? (self) : BOOLEAN  [Method on LIST-ITERATOR]
    Not documented.

sort (self (predicate FUNCTION-CODE)) : (LIST OF (LIKE (ANY-VALUE self)))  [Method on LIST]
    Perform a stable, destructive sort of self according to predicate, and return the result. If predicate has a < semantics, the result will be in ascending order. If predicate is NULL, a suitable < predicate is chosen depending on the first element of self, and it is assumed that all elements of self have the same type (supported element types are GENERALIZED-SYMBOL, STRING, INTEGER, and FLOAT).

map-null-to-nil-list ((self LIST)) : LIST  [Function]
    Return NIL-LIST iff self is NULL or self otherwise.

6.6.1 Lists as Sets

Similar to CONS lists LIST’s can also be treated as sets and support the set manipulations below. Note that LIST constructors do not check for proper set-hood and may have surprising results if a list contains duplicate elements.
subset? (self (otherList list)) : BOOLEAN
   Return true if every element of self also occurs in otherList. Uses an eql? test and a simple quadratic-time algorithm. Note that this does not check whether self and otherList actually are sets.

equivalent-sets? (self (otherList list)) : BOOLEAN
   Return true if every element of self occurs in otherList and vice versa. Uses an eql? test and a simple quadratic-time algorithm. Note that this does not check whether self and otherList actually are sets.

union (self (otherList list)) : LIST
   Return the set union of self and otherList. Uses an eql? test and a simple quadratic-time algorithm. Note that the result is only guaranteed to be a set if both self and otherList are sets.

intersection (self (otherList list)) : LIST
   Return the set intersection of self and otherList. Uses an eql? test and a simple quadratic-time algorithm. Note that the result is only guaranteed to be a set if both self and otherList are sets.

difference (self (otherList list)) : LIST
   Return the set difference of self and otherList (i.e., all elements that are in self but not in otherList). Uses an eql? test and a simple quadratic-time algorithm. Note that the result is only guaranteed to be a set if both self and otherList are sets.

subtract (self (otherList list)) : LIST
   Return the set difference of self and otherList by destructively removing elements from self that also occur in otherList. Uses an eql? test and a simple quadratic-time algorithm. Note that the result is only guaranteed to be a set if self is a set.

SET is a subclass of LIST that overrides certain LIST operations to prevent duplicate elements. The following additional or modified operations are supported:

SET : LIST, SET-MIXIN
   Not documented.

any-value : OBJECT
   Not documented.

insert (self (value (LIKE (ANY-VALUE self)))) : [Method on set]
   Add value to the set self unless it is already a member.

push (self (value (LIKE (ANY-VALUE self)))) : [Method on set]
   Add value to the front of set self unless it is already a member.

insert-last (self (value (LIKE (ANY-VALUE self)))) : [Method on set]
   Add value to the end of set self unless it is already a member.

substitute (self (new object) (old object)) : (LIKE self) [Method on set]
   Destructively replace old with new in the set self unless new is already a member.
**concatenate** (set1 (set2 list) &rest (otherSets list)) : set  [Method on set]
Union set2 and all otherSets onto the end of set1. The operation is destructive wrt set1, but leaves all other sets intact. The two mandatory parameters allow us to optimize the common binary case by not relying on the somewhat less efficient variable arguments mechanism.

**object-equal?** (x (y object)) : BOOLEAN  [Method on set]
Return TRUE iff x and y are SET’s with equivalent members. Uses equal? to test equality of elements. This is more general than equivalent-sets?, since that only uses an eql? test.

**equal-hash-code** (self) : INTEGER  [Method on set]
Return an equal? hash code for self. Note that this is O(N) in the number of elements of self.

**set** (&rest (values object)) : set  [Function]
Return a set containing values, in order.

### 6.7 Property and Key-Value Lists

**property-list** : dictionary  [Class]
Not documented.

- **any-key** : object  [Class Parameter of ]
  Not documented.

- **any-value** : object  [Class Parameter of ]
  Not documented.

- **the-plist** : cons  [Slot of ]
  Not documented.

**empty?** (self) : BOOLEAN  [Method on property-list]
Not documented.

**non-empty?** (self) : BOOLEAN  [Method on property-list]
Not documented.

**object-equal?** (x (y object)) : BOOLEAN  [Method on property-list]
Return TRUE if x and y represent the same set of key/value pairs.

**equal-hash-code** (self) : INTEGER  [Method on property-list]
Return an equal? hash code for self. Note that this is O(N) in the number of entries of self.

**length** (self) : INTEGER  [Method on property-list]
Not documented.

**lookup** (self (key (like (any-key self)))) (like (any-value self))  [Method on property-list]
Not documented.
**insert-at** *(self (key (like (any-key self)))) (value (like (any-value self))))* : [Method on PROPERTY-LIST]

Insert the entry `<key`, value`> into the property list `self`. If a previous entry existed with key `key`, that entry is replaced.

**remove-at** *(self (key (like (any-key self))))* : [Method on PROPERTY-LIST]

OBJECT

Remove the entry that matches the key `key`. Return the value of the matching entry, or NULL if there is no matching entry. Assumes that at most one entry matches `key`.

**copy** *(self)* : [Method on PROPERTY-LIST]

COPY

Return a copy of the list `self`. The conses in the copy are freshly allocated.

**clear** *(self)* : [Method on PROPERTY-LIST]

COPY

Make `self` an empty property list.

**allocate-iterator** *(self)* : [Method on PROPERTY-LIST]

OF

Not documented.

**next?** *(self)* : BOOLEAN [Method on PROPERTY-LIST-ITERATOR]

Not documented.

**KV-CONS** : STANDARD-OBJECT

Not documented.

**any-key** : OBJECT

Not documented.

**any-value** : OBJECT

Not documented.

**key** : OBJECT

Not documented.

**value** : OBJECT

Not documented.

**rest** : KV-CONS

Not documented.

**kv-cons** ((key OBJECT) (value OBJECT) (rest KV-CONS)) : KV-CONS [Function]

Create, fill-in, and return a new KV-CONS.

**copy-kv-cons-list** ((kvconslist KV-CONS)) : KV-CONS [Function]

Return a copy of the cons list `consList`.

**KEY-VALUE-LIST** : DICTIONARY

Not documented.

**any-key** : OBJECT

Not documented.
any-value : OBJECT
Not documented.

the-kv-list : KV-CONS
Not documented.

empty? (self) : BOOLEAN
Not documented.

non-empty? (self) : BOOLEAN
Not documented.

object-equal? (x (y OBJECT)) : BOOLEAN
Return TRUE if x and y represent the same set of key/value pairs.

equal-hash-code (self) : INTEGER
Return an equal? hash code for self. Note that this is O(N) in the number of entries of self.

length (self) : INTEGER
Not documented.

lookup (self (key (LIKE (ANY-KEY SELF)))) (LIKE (ANY-VALUE SELF))
Not documented.

reverse (self) : (LIKE SELF)
Destructively reverse the members of the list self.

insert-at (self (key (LIKE (ANY-KEY SELF)))) (value (LIKE (ANY-VALUE SELF)))
Insert the entry `<key`, value> into the association self. If a previous entry existed with key key, that entry is replaced.

remove-at (self (key (LIKE (ANY-KEY SELF))))
Remove the entry that matches the key key. Return the value of the matching entry, or NULL if there is no matching entry. Assumes that at most one entry matches key.

insert-entry (self (key (LIKE (ANY-KEY SELF)))) (value (LIKE (ANY-VALUE SELF)))
Insert an entry `<key`, value> to self unless an identical entry already exists. This can generate duplicate entries for key.

remove-entry (self (key (LIKE (ANY-KEY SELF)))) (value (LIKE (ANY-VALUE SELF)))
Remove the entry that matches `<key`, value>. Assumes that more than one entry can match key.
push (self (value kv-cons)):
Make value be the new first element of self. Note that the rest slot of value should be null, since it will be overwritten. This might duplicate an existing entry. If a previous entry existed with the same key as value, that entry is retained, but shadowed by this new entry.

kv-push (self (key (like (any-key self)))) (value (like (any-value self))):
Add a new entry <'key', value> to the front of the association self. This might duplicate an existing entry. If a previous entry existed with key key, that entry is retained, but shadowed by this new entry.

pop (self):
Remove and return the value of the first element of the kv-list self. It does NOT return the KV CONS object. Return null if the list is empty.

copy (self):
Return a copy of the kv-list self. The kv-conses in the copy are freshly allocated.

clear (self):
Make self an empty dictionary.

consify (self):
Return a list of key-value pairs in self.

allocate-iterator (self):
Not documented.

next? (self):
Not documented.

6.8 Vectors

VECTOR : SEQUENCE
Not documented.

any-value : OBJECT
Not documented.

array-size : INTEGER
Not documented.

the-array : (native-vector of (like (any-value self)))
Not documented.

EXTENSIBLE-VECTOR : VECTOR
Not documented.

empty? (self):
Return true if self has length 0.
non-empty? (self) : BOOLEAN
    Return true if self has length > 0.

object-equal? (x (y object)) : BOOLEAN
    Return TRUE iff the vectors x and y are structurally equivalent. Uses equal? to test
    equality of elements.

equal-hash-code (self) : INTEGER
    Return an equal? hash code for self.

vector (&rest (values object)) : VECTOR
    Return a vector containing values, in order.

first (self) : (LIKE (ANY-VALUE SELF))
    Not documented.

second (self) : (LIKE (ANY-VALUE SELF))
    Not documented.

third (self) : (LIKE (ANY-VALUE SELF))
    Not documented.

fourth (self) : (LIKE (ANY-VALUE SELF))
    Not documented.

fifth (self) : (LIKE (ANY-VALUE SELF))
    Not documented.

nth (self (position INTEGER)) : (LIKE (ANY-VALUE SELF))
    Not documented.

last (self) : (LIKE (ANY-VALUE SELF))
    Return the last item in the vector self.

but-last (self) : (ITERATOR OF (LIKE (ANY-VALUE SELF)))
    Generate all but the last element of the vector self.

length (self) : INTEGER
    Not documented.

member? (self (object object)) : BOOLEAN
    Not documented.

position (self (object object) (start INTEGER)) : INTEGER
    Return the position of object within the vector self (counting from zero); or return
    null if object does not occur within self (uses an eql? test). If start was supplied as
    non-‘null’, only consider the portion starting at start, however, the returned position
    will always be relative to the entire vector.

last-position (self (object object) (end INTEGER)) : INTEGER
    Return the position of object within the vector self (counting from zero); or return
    null if object does not occur within self (uses an eql? test). If end was supplied as
    non-‘null’, only consider the portion ending at index end, however, the returned
    position will always be relative to the entire vector.
**insert-at**  
\( \text{self} (\text{offset \ integer}) \)  
\( (\text{value \ like \ (any-value \ self))}) : \)  
Not documented.

**copy**  
\( \text{self} \) : \( \text{VECTOR \ of \ (like \ (any-value \ self))} \)  
Return a copy of the vector self.

**clear**  
\( \text{self} \) :  
Not documented.

**resize-vector**  
\( \text{(self \ vector \ size \ integer)} \) :  
Change the size of self to size. If size is smaller than the current size of self the vector will be truncated. Otherwise, the internal array of self will be grown to size and unused elements will be initialized to NULL.

**consify**  
\( \text{self} \) : \( \text{(cons \ of \ (like \ (any-value \ self))}) \)  
Return a list of elements in self.

**insert-at**  
\( \text{self} (\text{offset \ integer}) \)  
\( (\text{value \ like \ (any-value \ self))}) : \)  
Not documented.

**insert**  
\( \text{self} (\text{value \ like \ (any-value \ self)}) \) :  
Append value to the END of the sequence self. Resize the array if necessary.

**remove**  
\( \text{self} (\text{value \ like \ (any-value \ self)}) : \)  
VECTOR-SEQUENCE  
Remove value from the sequence self, and left shift the values after it to close the gap.

**length**  
\( \text{self} \) : \( \text{INTEGER} \)  
Not documented.

### 6.9 Hash Tables

**hash-table** : \( \text{ABSTRACT-HASH-TABLE} \)  
Not documented.

**any-key** : \( \text{OBJECT} \)  
Not documented.

**any-value** : \( \text{OBJECT} \)  
Not documented.

**string-hash-table** : \( \text{ABSTRACT-HASH-TABLE} \)  
Not documented.

**any-key** : \( \text{STRING} \)  
Not documented.

**any-value** : \( \text{OBJECT} \)  
Not documented.
STRING-TO-INTEGER-HASH-TABLE : ABSTRACT-HASH-TABLE
Not documented.

any-key : STRING
Not documented.

any-value : INTEGER
Not documented.

INTEGER-HASH-TABLE : ABSTRACT-HASH-TABLE
Not documented.

any-key : INTEGER
Not documented.

any-value : OBJECT
Not documented.

FLOAT-HASH-TABLE : ABSTRACT-HASH-TABLE
Not documented.

any-key : FLOAT
Not documented.

any-value : OBJECT
Not documented.

lookup (self (key (like (any-key self)))) : (like (any-value self))
Not documented.

insert-at (self (key (like (any-key self)))) (value (like (any-value self)))) :
Not documented.

remove-at (self (key (like (any-key self)))) :
Not documented.

lookup (self (key STRING)) : (like (any-value self))
Not documented.

insert-at (self (key STRING) (value OBJECT)) :
Not documented.

remove-at (self (key STRING)) :
Not documented.

lookup (self (key STRING)) : INTEGER
Not documented.
**insert-at** (self (key string) (value integer)) : [Method on STRING-TO-INTEGER-HASH-TABLE]
Not documented.

**lookup** (self (key integer)) : (like (any-value self)) [Method on INTEGER-HASH-TABLE]
Not documented.

**insert-at** (self (key integer) (value object)) : [Method on INTEGER-HASH-TABLE]
Not documented.

**insert-at** (self (key float) (value object)) : [Method on FLOAT-HASH-TABLE]
Not documented.

STELLA provides its own implementation of hash tables for cases where language-native implementations are not available, or where additional features are needed.

**STELLA-HASH-TABLE** : ABSTRACT-HASH-TABLE [Class]
Not documented.

**any-key** : object [Class Parameter of ]
Not documented.

**any-value** : object [Class Parameter of ]
Not documented.

**the-table** : (array size () of kv-cons) [Slot of ]
Not documented.

**size** : integer [Slot of ]
Not documented.

**initial-size** : integer [Slot of ]
If supplied, the initial hash table will be sized to hold at least that many elements.

**free-elements** : integer [Slot of ]
Not documented.

**equal-test?** : boolean [Slot of ]
If true use equal? as the equality test and equal-hash-code as the hash function, otherwise, use eql? and hash-code (the default).

**lookup** (self (key (like (any-key self)))) : (like [Method on STELLA-HASH-TABLE]
(any-value self))
Lookup the entry identified by key in self and return its value, or NULL if no such entry exists. Uses an eql? test by default or equal? if equal-test? of self is TRUE.

**insert-at** (self (key (like (any-key self)))) (value (like (any-value self))) : [Method on STELLA-HASH-TABLE]
Set the value of the entry identified by key in self to value or add a new entry if no entry with key exists yet. Uses an eql? test by default or equal? if equal-test? of self is TRUE.
**remove-at** ((self (key (like (any-key self))))) :  
[Method on STELLA-HASH-TABLE]  
Remove the entry identified by key from self. Uses an eql? test by default or equal? if equal-test? of self is TRUE.

**length** (self) : INTEGER  
[Method on STELLA-HASH-TABLE]  
Return the number of entries in self.

**empty?** (self) : BOOLEAN  
[Method on STELLA-HASH-TABLE]  
Return TRUE if self has zero entries.

**non-empty?** (self) : BOOLEAN  
[Method on STELLA-HASH-TABLE]  
Return TRUE if self has at least 1 entry.

**copy** (self) : (like self)  
[Method on STELLA-HASH-TABLE]  
Return a copy of the hash table self. The bucket table and buckets are freshly allocated, however, the keys and values of entries are not copied themselves (similar to what we do for lists, etc.).

**clear** (self) :  
[Method on STELLA-HASH-TABLE]  
Remove all entries from self. This will result in a re-initialization of the table upon the first insertion into self.

**consify** (self) : (cons of cons)  
[Method on STELLA-HASH-TABLE]  
Collect all entries of self into a cons list of (<key> <value>) pairs and return the result.

**object-equal?** ((x (y object))) : BOOLEAN  
[Method on STELLA-HASH-TABLE]  
Return TRUE if x and y represent the same set of key/value pairs.

**equal-hash-code** (self) : INTEGER  
[Method on STELLA-HASH-TABLE]  
Return an equal? hash code for self. Note that this is O(N) in the number of entries of self.

**allocate-iterator** (self) :  
[Method on STELLA-HASH-TABLE]  
(STELLA-HASH-TABLE-ITERATOR OF (like (any-key self)) (like (any-value self)))  
Allocate an iterator for self.

Hashing objects into STELLA hash tables is accomplished via **hash-code** and **equal-hash-code** methods. These methods are implemented for all built-in STELLA types but are user extensible for cases where special functionality on user-defined objects is needed. Defining new **hash-code** methods should only be necessary if new wrapper types are defined, since for all types descending from **STANDARD-OBJECT** the built-in method should be adequate.

**object-hash-code** ((self object)) : INTEGER  
[Function]  
Return a hash code for self (can be negative). Two objects that are eq? are guaranteed to generate the same hash code. Two objects that are not eq? do not necessarily generate different hash codes. Similar to **hash-code** but always hashes on the address of self even if it is a wrapper.
**hash-code** *(self)* : integer  
**[Method on object]**  
Return a hash code for *self* (can be negative). Two objects that are **eq1?** are guaranteed to generate the same hash code. Two objects that are not **eq1?** do not necessarily generate different hash codes.

**hash-code** *(self)* : integer  
**[Method on standard-object]**  
Not documented.

**hash-code** *(self)* : integer  
**[Method on string-wrapper]**  
Not documented.

**hash-code** *(self)* : integer  
**[Method on integer-wrapper]**  
Not documented.

**hash-code** *(self)* : integer  
**[Method on float-wrapper]**  
Not documented.

**hash-code** *(self)* : integer  
**[Method on character-wrapper]**  
Not documented.

**hash-code** *(self)* : integer  
**[Method on boolean-wrapper]**  
Not documented.

**hash-code** *(self)* : integer  
**[Method on string]**  
Not documented.

**hash-code** *(self)* : integer  
**[Method on integer]**  
Not documented.

**hash-code** *(self)* : integer  
**[Method on float]**  
Not documented.

**hash-code** *(self)* : integer  
**[Method on character]**  
Not documented.

**equal-hash-code** *(self)* : integer  
**[Method on object]**  
Return a hash code for *self* (can be negative). Two objects that are **equal?** are guaranteed to generate the same hash code (provided, that writers of **object-equal?** methods also implemented the appropriate **equal-hash-code** method). Two objects that are not **equal?** do not necessarily generate different hash codes.

The following low-level utilities are available to implement specialized hashing schemes or for defining new versions of **equal-hash-code**.

**hashmod** *(code integer) (size integer)* : integer  
**[Function]**  
Map the hash code *code* onto a bucket index for a hash table of size (i.e., onto the interval [0..size-1]). This is just like **rem** for positive hash codes but also works for negative hash codes by mapping those onto a positive number first. Note, that the sign conversion mapping is not equivalent to calling the **abs** function (it simply masks the sign bit for speed) and therefore really only makes sense for hash codes.
**Chapter 6: Library Functions**

**rotate-hash-code** ((arg INTEGER)) : INTEGER  [Function]
Rotate arg to the right by 1 position. This means shift arg to the right by one and feed in args bit zero from the left. In Lisp the result will stay in positive FIXNUM range. In C++ and Java this might return a negative value which might be equal to NULL-INTEGER. Important: to make this inlinable, it must be called with an atom (i.e., constant or variable) as its argument. This function is primarily useful for hashing sequences of items where the hash code should take the sequential order of elements into account (e.g., lists).

### 6.10 Key Value Maps

**KEY-VALUE-MAP** is a full-featured dictionary class that supports **eq1?** or extensible **equal?** equality tests, O(1) access operations even for large numbers of entries by using a hash table, light-weight **KV-CONS** representation for small tables and iteration even if the dictionary is represented by a hash table (note that in STELLA we cannot iterate over regular **HASH-TABLE**'s, since native Lisp hash tables do not allow us to implement a hash table iterator). Since large **KEY-VALUE-MAP**'s are implemented via **STELLA-HASH-TABLE**'s, we can support iteration.

**KEY-VALUE-MAP : dictionary**  [Class]
Full-featured dictionary class that supports **eq1?** or **equal?** equality tests, O(1) access operations even for large numbers of entries by using a hash table, light-weight **KV-CONS** representation for small tables and iteration even if the dictionary is represented by a hash table.

**the-map : object**  [Slot of ]
Not documented.

**equal-test? : boolean**  [Slot of ]
If true use **equal?** as the equality test (and **equal-hash-code** as the hash function), otherwise, use **eq1?** (and **hash-code**) (the default).

**initial-size : integer**  [Slot of ]
If supplied, the initial table will be sized to hold at least that many elements.

**crossover-point : integer**  [Slot of ]
Not documented.

**lookup** (self (key (like (any-key self)))) (like (any-value self))  [Method on **KEY-VALUE-MAP**]
Lookup the entry identified by key in self and return its value, or NULL if no such entry exists. Uses an **eq1?** test by default or **equal?** if **equal-test?** of self is TRUE.

**insert-at** (self (key (like (any-key self)))) (value (like (any-value self)))  [Method on **KEY-VALUE-MAP**]
Set the value of the entry identified by key in self to value or add a new entry if no entry with key exists yet. Uses an **eq1?** test by default or **equal?** if **equal-test?** of self is TRUE.
**Chapter 6: Library Functions**

**remove-at (self (key (like (any-key self))))** : [Method on KEY-VALUE-MAP]
Remove the entry identified by key from self. Uses an eql? test by default or equal? if equal-test? of self is TRUE.

**length (self) : INTEGER** [Method on KEY-VALUE-MAP]
Return the number of entries in self.

**empty? (self) : BOOLEAN** [Method on KEY-VALUE-MAP]
Return TRUE if self has zero entries.

**non-empty? (self) : BOOLEAN** [Method on KEY-VALUE-MAP]
Return TRUE if self has at least 1 entry.

**copy (self) : (like self)** [Method on KEY-VALUE-MAP]
Return a copy of the map self. All entries are freshly allocated, however, the keys and values of entries are not copied themselves (similar to what we do for lists, etc.).

**clear (self)** :
Reset self to have zero entries.

**allocate-iterator (self) : (dictionary-iterator of (like (any-key self)) (like (any-value self)))** [Method on KEY-VALUE-MAP]
Allocate an iterator for self. The only modifying operations allowed during iteration are removal of the current element or changing its value. All other removal or insertion operations might lead to corruption or undefined results.

**consify (self) : cons** [Method on KEY-VALUE-MAP]
Collect all entries of self into a cons list of (<key> <value>) pairs and return the result.

**object-equal? (x y object)** : BOOLEAN [Method on KEY-VALUE-MAP]
Return TRUE if x and y represent the same set of key/value pairs.

**equal-hash-code (self)** : INTEGER [Method on KEY-VALUE-MAP]
Return an equal? hash code for self. Note that this is O(N) in the number of entries of self.

### 6.11 Hash Sets

**HASH-SET** is a full-featured set class that supports eql? or extensible equal? equality tests, O(1) insert and member? operations, O(N) intersection etc. operations even for large numbers of entries by using a STELLA hash table, light-weight KV-CONS representation for small sets and iteration even if the set is represented by a hash table. The only minor drawback right now is that we waste one value slot per entry, since we piggy-back off KEY-VALUE-MAP’s, however, that wastes at most 25% space.

**HASH-SET : KEY-VALUE-MAP, SET-MIXIN** [Class]
Full-featured set class that supports eql? or equal? equality tests, O(1) insert and member? operations & O(N) intersection etc. operations even for large numbers of entries by using a hash table, light-weight KV-CONS representation for small sets
and iteration even if the set is represented by a hash table. The only minor drawback right now is that this wastes a value slot per entry, since we piggy-back off KEY-VALUE-MAP’s, however, that wastes at most 25% space.

**hash-set** (\krest (\values object)) : HASH-SET

Return an **eql?** HASH-SET containing \values.

**member?** (\self (\object object)) : BOOLEAN

Return **TRUE** iff \object is a member of the set \self. Uses an **eql?** test by default or **equal?** if **equal-test?** of \self is **TRUE**.

**insert** (\self (\value (\like (any-value \self))))

Add \value to the set \self unless it is already a member. Uses an **eql?** test by default or **equal?** if **equal-test?** of \self is **TRUE**.

**remove** (\self (\value (\like (any-value \self)))) : \self

Destructively remove \value from the set \self if it is a member and return \self. Uses an **eql?** test by default or **equal?** if **equal-test?** of \self is **TRUE**.

**remove-if** (\self (\test? function-code)) : \self

Destructively remove all elements of the set \self for which \test? evaluates to **TRUE**. \test? takes a single argument of type **OBJECT** and returns **TRUE** or **FALSE**. Returns \self.

**pop** (\self)

Remove and return an arbitrary element of the set \self. Return **NULL** if the set is empty. Performance note: for large sets implemented via hash tables it takes \(O(N)\) to empty out the set with repeated calls to **pop**, since the emptier the table gets, the longer it takes to find an element. Therefore, it is usually better to use iteration with embedded removals for such cases.

**substitute** (\self (\new object) (\old object))

Destructively replace \old with \new in the set \self unless \new is already a member. Uses an **eql?** test by default or **equal?** if **equal-test?** of \self is **TRUE**.

**copy** (\self)

Return a copy of the set \self. All entries are freshly allocated, however, the values are not copied themselves (similar to what we do for lists, etc.).

**consify** (\self)

Collect all entries of \self into a cons list and return the result.

**subset?** (\self (\otherSet hash-set)) : BOOLEAN

Return **true** if every element of \self also occurs in \otherSet. Uses an **eql?** test by default or **equal?** if **equal-test?** of \self is **TRUE**.

**equivalent-sets?** (\self (\otherSet hash-set)) : BOOLEAN

Return **true** if every element of \self occurs in \otherSet and vice versa. Uses an **eql?** test by default or **equal?** if **equal-test?** of \self is **TRUE**.
intersection (self (otherSet hash-set)) : hash-set
Return the set intersection of self and otherSet as a new set. Uses an eql? test by default or equal? if equal-test? of self is TRUE.

union (self (otherSet hash-set)) : hash-set
Return the set union of self and otherSet as a new set. Uses an eql? test by default or equal? if equal-test? of self is TRUE.

difference (self (otherSet hash-set)) : hash-set
Return the set difference of self and otherSet as a new set (i.e., all elements that are in self but not in otherSet). Uses an eql? test by default or equal? if equal-test? of self is TRUE.

subtract (self (otherSet hash-set)) : hash-set
Return the set difference of self and otherSet by destructively removing elements from self that also occur in otherSet. Uses an eql? test by default or equal? if equal-test? of self is TRUE.

object-equal? (x (y object)) : BOOLEAN
Return TRUE iff sets x and y are HASH-SET’s with equivalent members. Uses an eql? test by default or equal? if equal-test? of self is TRUE. This is equivalent to calling equivalent-sets?.

equal-hash-code (self) : INTEGER
Return an equal? hash code for self. Note that this is O(N) in the number of elements of self.

6.12 Iterators

empty? (self) : BOOLEAN
Return TRUE if the sequence represented by self has no elements. Side-effect free.

member? (self (value object)) : BOOLEAN
Iterate over values of self, returning TRUE if one of them is eql to ‘value.

length (self) : INTEGER
Iterate over self, and count how many items there are. Bad idea if self iterates over an infinite collection, since in that case it will run forever.’

pop (self) (like (any-value self))
Return the first item of the sequence represented by self, or NULL if it is empty. Destructively uses up the first iteration element.

advance (self (n integer)) : (like self)
Return self after skipping over the first n elements in the (remainder of the) iteration.

concatenate (iterator1 (iterator2 iterator) &rest (otherIterators iterator)) : all-purpose-iterator
Return an iterator that first generates all values of iterator1, then those of iterator2, and then those of all otherIterators. The generated values can be filtered by supplying a filter function to the resulting iterator.
consify (self) : (CONS OF (LIKE (ANY-VALUE SELF)))  [Method on ITERATOR]
Return a list of elements generated by self.

next? (self) : BOOLEAN  [Method on ALL-PURPOSE-ITERATOR]
Apply the stored next? function to self.

6.13 Symbols

lookup-symbol ((name STRING)) : SYMBOL  [Function]
Return the first symbol with name visible from the current module.

intern-symbol ((name STRING)) : SYMBOL  [Function]
Return a newly-created or existing symbol with name name.

unintern-symbol ((self SYMBOL)) :
Remove self from its home module and the symbol table.

lookup-symbol-in-module ((name STRING) (module MODULE)
  (local? BOOLEAN)) : SYMBOL
Return the first symbol with name visible from module. If local? only consider symbols directly interned in module. If module is null, use *MODULE* instead.

intern-symbol-in-module ((name STRING) (module MODULE)
  (local? BOOLEAN)) : SYMBOL
Look for a symbol named name in module (if local? do not consider inherited modules). If none exists, intern it locally in module. Return the existing or newly-created symbol.

intern-derived-symbol ((baseSymbol GENERALIZED-SYMBOL)
  (newName STRING)) : SYMBOL
Return a newly-created or existing symbol with name newName which is interned in the same module as baseSymbol.

visible-symbol? ((self SYMBOL)) : BOOLEAN  [Function]
Return true if self is visible from the current module.

lookup-visible-symbols-in-module ((name STRING)
  (module MODULE) (enforceShadowing? BOOLEAN)) : (CONS OF SYMBOL)
Return the list of symbols with name visible from module. More specific symbols (relative to the module precedence order defined by visible-modules) come earlier in the list. If module is null, start from *MODULE* instead. If enforceShadowing? is true, do not return any symbols that are shadowed due to some :SHADOW declaration.

import-symbol ((symbol SYMBOL) (module MODULE)) : SYMBOL  [Function]
Import symbol into module and return the imported symbol. Signal an error if a different symbol with the same name already exists locally in module. Any symbol with the same name visible in module by inheritance will be shadowed by the newly imported symbol.
safe-import-symbol (((symbol symbol) (module module)) : symbol)  
Safe version of import-symbol (which see). Only imports symbol if no symbol with 
that name is currently interned or visible in module. Returns symbol if it was im-
ported or the conflicting symbol in module otherwise.

lookup-surrogate (((name string)) : surrogate)  
Return the first surrogate with name visible from the current module.

intern-surrogate (((name string)) : surrogate)  
Return a newly-created or existing surrogate with name name.

unintern-surrogate (((self surrogate)) :  
Remove self from its home module and the surrogate table.

lookup-surrogate-in-module (((name string) (module module)  
(local? boolean)) : surrogate  
Return the first surrogate with name visible from module. If local? only consider 
surrogates directly interned in module. If module is null, use *MODULE* instead.

intern-surrogate-in-module (((name string) (module module)  
(local? boolean)) : surrogate  
Look for a symbol named name in module (if local? do not consider inherited mod-
ules). If none exists, intern it locally in module. Return the existing or newly-created 
symbol.

intern-derived-surrogate (((baseSymbol generalized-symbol)  
(newName string)) : surrogate  
Return a newly-created or existing surrogate with name newName which is interned 
in the same module as baseSymbol.

visible-surrogate? (((self surrogate)) : boolean)  
Return true if self is visible from the current module.

lookup-visible-surrogates-in-module (((name string)  
(module module) (enforceShadowing? boolean)) : (cons of surrogate)  
Return the list of surrogates with name visible from module. More specific surrogates 
(relative to the module precedence order defined by visible-modules) come earlier 
in the list. If module is null, start from *MODULE* instead. If enforceShadowing? 
is true, do not return any surrogates that are shadowed due to some :SHADOW 
declaration.

import-surrogate (((surrogate surrogate) (module module)) : surrogate  
Import surrogate into module and return the imported surrogate. Signal an error 
if a different surrogate with the same name already exists locally in module. Any 
surrogate with the same name visible in module by inheritance will be shadowed by 
the newly imported surrogate.
safe-import-surrogate \((\text{surrogate\ surrogate} (\text{module\ module}))\) :  [Function]

\text{SURROGATE}
Safe version of import-surrogate (which see). Only imports surrogate if no surrogate with that name is currently interned or visible in module. Returns surrogate if it was imported or the conflicting surrogate in module otherwise.

lookup-keyword \((\text{name\ STRING})\) : KEYWORD  [Function]

Return the keyword with name if it exists.

intern-keyword \((\text{name\ STRING})\) : KEYWORD  [Function]

Return a newly-created or existing keyword with name name. Storage note: a COPY of name is stored in the keyword.

gensym \((\text{prefix\ STRING})\) : SYMBOL  [Function]

Return a transient symbol with a name beginning with prefix and ending with a globally gensym’d integer.

local-gensym \((\text{prefix\ STRING})\) : SYMBOL  [Function]

Not documented.

symbol-plist \((\text{symbol\ symbol})\) : CONS  [Function]

Return the property list of symbol. The symbol-plist of a symbol can be set with setf. IMPORTANT: Property list are modified destructively, hence, if you supply it as a whole make sure to always supply a modifiable copy, e.g., by using bquote.

symbol-property \((\text{symbol\ symbol}) (\text{key\ standard-object})\) :  [Function]

OBJECT
Return the property of symbol whose key is eq? to key. Symbol properties can be set with setf.

symbol-value \((\text{symbol\ symbol})\) : OBJECT  [Function]

Return the value of symbol. Note, that this value is not visible to code that references a variable with the same name as symbol. The symbol-value is simply a special property that can always be accessed in constant time. The symbol-value of a symbol can be changed with setf.

symbolize \((\text{surrogate\ surrogate})\) : SYMBOL  [Function]

Convert surrogate into a symbol with the same name and module.

6.14 Context and Modules

get-stella-context \((\text{pathName\ STRING}) (\text{error?\ BOOLEAN})\) : CONTEXT  [Function]

Return the context located at pathName, or null if no such context exists. If error? is true, throw an exception if no context is found, otherwise silently return null.

clear-context \((\text{self\ context})\) :  [Function]

Destroy all objects belonging to self or any of its subcontexts.

within-context \((\text{contextForm\ OBJECT}) (\text{body\ CONS})\) : OBJECT  [Macro]

Execute body within the context resulting from contextForm.
**Chapter 6: Library Functions**

**destroy-context (self)** :

Make the translator happy.

**destroy-context (self)** :

Destroy the context self, and recursively destroy all contexts that inherit self.

**change-context (context)** : CONTEXT

Change the current context to be the context context.

**change-context (contextName)** : CONTEXT

Change the current context to be the context named contextName.

**cc (&rest (name name))** : CONTEXT

Change the current context to the one named name. Return the value of the new current context. If no name is supplied, return the pre-existing value of the current context. cc is a no-op if the context reference cannot be successfully evaluated.

**defmodule ((name name) &rest (options object))** :

Define (or redefine) a module named name. The accepted syntax is:

```
(defmodule <module-name>
  [:documentation <docstring>]
  [:includes <<module-name> | (<module-name>*)>]
  [:uses <<module-name> | (<module-name>*)>]
  [:lisp-package <package-name-string>]
  [:java-package <package-specification-string>]
  [:cpp-namespace <namespace-name-string>]
  [:java-catchall-class
    [:api? {TRUE | FALSE}]
    [:case-sensitive? {TRUE | FALSE}]
    [:shadow <<symbol>*>]
    [:java-catchall-class <class-name-string>]
  ]
  [<other-options>*])
```

name can be a string or a symbol.

Modules include objects from other modules via two separate mechanisms: (1) they inherit from their parents specified via the :includes option and/or a fully qualified module name, and (2) they inherit from used modules specified via the :uses option. The main difference between the two mechanisms is that inheritance from parents is transitive, while uses-links are only followed one level deep. I.e., a module A that uses B will see all objects of B (and any of B’s parents) but not see anything from modules used by B. Another difference is that only objects declared as public can be inherited via uses-links (this is not yet enforced). Note that - contrary to Lisp - there are separate name spaces for classes, functions, and variables. For example, a module could inherit the class CONS from the STELLA module, but shadow the function of the same name.

The above discussion of :includes and :uses semantics keyed on the inheritance/visibility of symbols. The PowerLoom system makes another very important distinction: If a module A is inherited directly or indirectly via :includes specification(s) by a submodule B, then all definitions and facts asserted in A are
visible in B. This is not the cases for :uses; the :uses options does not impact
inheritance of propositions at all.

The list of modules specified in the :includes option plus (if supplied) the parent in
the path used for name become the new module’s parents. If no :uses option was
supplied, the new module will use the STELLA module by default, otherwise, it will
use the set of specified modules.

If :case-sensitive? is supplied as TRUE, symbols in the module will be interned
case-sensitively, otherwise (the default), they will be converted to uppercase before
they get interned. That means that any reference from inside a case-sensitive module
to a non-case-sensitive module will have to use uppercase names for symbols in the
non-case-sensitive module. The standard system modules are all NOT case sensitive.

Modules can shadow definitions of functions and classes inherited from parents or
used modules. Shadowing is done automatically, but generates a warning unless
the shadowed type or function name is listed in the :shadow option of the module
definition.

Examples:

```lisp
(defmodule "PL-KERNEL/PL-USER"
  :uses ("LOGIC" "STELLA")
  :package "PL-USER")

(defmodule PL-USER/GENEALOGY)
```

The remaining options are relevant only for modules that contain STELLA code.
Modules used only to contain knowledge base definitions and assertions have no use
for them:

The keywords :lisp-package, :java-package, and :cpp-package specify the name
of a native package or name space in which symbols of the module should be allocated
when they get translated into one of Lisp, Java, or C++. By default, Lisp symbols are
allocated in the STELLA package, and C++ names are translated without any prefixes.
The rules that the STELLA translator uses to attach translated Java objects to classes
and packages are somewhat complex. Use :java-package option to specify a list of
package names (separated by periods) that prefix the Java object in this module. Use :
java-catchall-class to specify the name of the Java class to contain all global & special
variables, parameter-less functions and functions defined on arguments that are not
classes in the current module. The default value will be the name of the module.

When set to TRUE, the :api? option tells the PowerLoom User Manual generator
that all functions defined in this module should be included in the API section. Ad-
ditionally, the Java translator makes all API functions synchronized.

---

**get-stella-module** ((pathName string) (error? boolean)) : module

Return the module located at pathName, or null if no such module exists. The
search looks at ancestors and top-most (cardinal) modules. If error? is true, throw
an exception if no module is found.

**find-or-create-module** ((pathname string)) : module

Return a module located at pathname if one exists, otherwise create one.
clear-module (&rest (name NAME)) : [N-Command]
Destroy all objects belonging to module name or any of its children. If no name is supplied, the current module will be cleared after confirming with the user. Important modules such as STELLA are protected against accidental clearing.

destroy-module ((self MODULE)) : [Function]
Destroy the module self, and recursively destroy all contexts that inherit self.

destroy-context (self) : [Method on MODULE]
Destroy the context self, and recursively destroy all contexts that inherit self.

visible-modules ((from MODULE)) : (CONS OF MODULE) [Function]
Return a list of all modules visible from module from (or *module* if from is NULL. The generated modules are generated from most to least-specific and will start with the module from.

within-module ((moduleForm OBJECT) &body (body CONS)) : OBJECT [Macro]
Execute body within the module resulting from moduleForm. *module* is an acceptable moduleForm. It will locally rebind *module* and *context* and shield the outer bindings from changes.

in-module ((name NAME)) : MODULE [N-Command]
Change the current module to the module named name.

change-module (module) : MODULE [Method on MODULE]
Change the current module to be the module module.

change-module (moduleName) : MODULE [Method on STRING]
Change the current module to be the module named moduleName.

create-world ((parentContext CONTEXT) (name STRING)) : WORLD [Function]
Create a new world below the world or module parentContext. Optionally, specify a name.

push-world () : WORLD [Function]
Spawn a new world that is a child of the current context, and change the current context to the new world.

pop-world () : CONTEXT [Function]
Destroy the current world and change the current context to be its parent. Return the current context. Nothing happens if there is no current world.

destroy-context (self) : [Method on WORLD]
Destroy the context self, and recursively destroy all contexts that inherit self.

within-world ((worldForm OBJECT) &body (body CONS)) : OBJECT [Macro]
Execute body within the world resulting from worldForm.
6.15 Input and Output

**read-s-expression** (((stream input-stream)) : object boolean [Function])
Read one STELLA s-expression from stream and return the result. Return true as the second value on EOF.

**read-s-expression-from-string** (((string string)) : object [Function])
Read one STELLA s-expression from string and return the result.

**read-line** (((stream input-stream)) : string [Function])
Read one line from stream and return the result. This differs from native-read-line in that it is not platform-dependent. It recognizes any of the three common line ending formats: CR, LF, CR-LF in any combination. It is not as fast as native-read-line, however.

**read-character** (((inputStream input-stream)) : character boolean [Function])
Read one character from inputStream and return the result. Return true as the second value on EOF.

**unread-character** (((ch character) inputStream input-stream)) :
Unread ch from inputStream. Signal an error if ch was not the last character read.

**y-or-n?** (((message string)) : boolean [Function])
Read a line of input from STANDARD-INPUT and return true if the input was y or false if the input was n. Loop until either y or n was entered. If message is non-'null' prompt with it before the input is read. See also special variable *USER-QUERY-ACTION*.

**yes-or-no?** (((message string)) : boolean [Function])
Read a line of input from STANDARD-INPUT and return true if the input was yes or false if the input was no. Loop until either yes or no was entered. If message is non-'null' prompt with it before the input is read. See also special variable *USER-QUERY-ACTION*.

**flush-output** (((self output-stream)) :
Flush all buffered output of self.

6.16 Files

**open-input-file** (((fileName string) &rest (options keyword)) : [Function]
FILE-INPUT-STREAM
Open file fileName for input and return the resulting input stream. By default signal an error if the file does not exist. The only legal option so far is :IF-NOT-EXISTS (or :IF-NOT-EXISTS-ACTION) which specifies what to do in case the file does not exist. If its value is :ERROR then an error will be signaled. If it is :ABORT or :PROBE the opening operation will be aborted and NULL will be returned.
open-output-file \((\text{fileName\ string})\ \&\text{rest\ (options\ keyword))}\):

file-output-stream

Open file fileName for output and return the resulting output stream. By default the file will be created or overwritten if it already exists. If :IF-EXISTS (or :IF-EXISTS-ACTION) is one of the options its value specifies what to do in case the file already exists. If the value is :SUPERSEDE the pre-existing file will be overwritten. If the value is :APPEND the preexisting file will be appended to (if the file did not yet exist the file will simply be created). If the value is :ERROR then an error will be signaled. If it is :ABORT or :PROBE the opening operation will be aborted and NULL will be returned.

If :IF-NOT-EXISTS (or :IF-NOT-EXISTS-ACTION) is one of the options its value specifies what to do in case the file does not already exist. If the value is :CREATE, the file will simply be created. If the value is :ERROR then an error will be signaled. If it is :ABORT or :PROBE the opening operation will be aborted and NULL will be returned.

close-stream \((s\text{elf\ stream))\):

Close the stream self.

close-all-files ():

Close all currently open file streams. Use for emergencies or for cleanup.

with-input-file \((\text{binding\ cons})\ \&\text{body\ (body\ cons))}\): object

Sets up an unwind-protected form which opens a file for input and closes it afterwards. The stream for reading is bound to the variable provided in the macro form. Syntax is (WITH-INPUT-FILE (var filename options*) body+) where options can be any that are legal for open-input-file (which see).

with-output-file \((\text{binding\ cons})\ \&\text{body\ (body\ cons))}\): object

Sets up an unwind-protected form which opens a file for output and closes it afterwards. The stream for writing is bound to the variable provided in the macro form. Syntax is (WITH-OUTPUT-FILE (var filename options*) body+) where options can be any that are legal for open-output-file (which see).

probe-file? \((\text{fileName\ file-name))\): boolean

Return true if file fileName exists. Note that this does not necessarily mean that the file can also be read.

file-write-date \((\text{fileName\ file-name))\): calendar-date

Return the time at which file fileName was last modified or NULL if that cannot be determined.

file-length \((\text{fileName\ file-name))\): long-integer

Return the length of file fileName in bytes or NULL if that cannot be determined.

Note that this will currently overrun for files that are longer than what can be represented by a STELLA integer.

copy-file \((\text{fromFile\ file-name})\ (\text{toFile\ file-name})):\

Copy file fromFile to file toFile, clobbering any data already in toFile.

delete-file \((\text{fileName\ file-name))\):

Delete the file fileName.
**directory-file-name** ((directory file-name)) : FILE-NAME  
Return `directory` as a file name, i.e., without a terminating directory separator.

**directory-parent-directory** ((directory file-name) (level integer)) : FILE-NAME  
Return the `level`-th parent directory component of `directory` including the final directory separator, or the empty string if `directory` does not have that many parents.

**file-name-as-directory** ((file file-name)) : FILE-NAME  
Return `file` interpreted as a directory, i.e., with a terminating directory separator. If `file` is the empty string simply return the empty string, i.e., interpret it as the current directory instead of the root directory.

**file-name-directory** ((file file-name)) : FILE-NAME  
Return the directory component of `file` including the final directory separator or the empty string if `file` does not include a directory. Note that for purposes of this function, a logical host is considered part of the directory portion of `file`.

**file-name-without-directory** ((file file-name)) : FILE-NAME  
Return the file name portion of `file` by removing any directory and logical host components.

**file-name-without-extension** ((file file-name)) : FILE-NAME  
Remove `file`'s extension (or type) if there is any and return the result.

**file-extension** ((file file-name)) : STRING  
Return `file`'s extension (or type) if it has any including the separator character.

**file-base-name** ((file file-name)) : FILE-NAME  
Remove `file`'s directory (including logical host) and extension components and return the result.

**absolute-pathname?** ((pathname string)) : BOOLEAN  
Not documented.

**logical-host?** ((host string)) : BOOLEAN  
Not documented.

**logical-pathname?** ((pathname string)) : BOOLEAN  
Not documented.

**translate-logical-pathname** ((pathname string)) : STRING  
Not documented.

**directory-separator** () : CHARACTER  
Not documented.

**directory-separator-string** () : STRING  
Not documented.
6.17 Dates and Times

**get-current-date-time ()**: INTEGER INTEGER INTEGER KEYWORD

INTEGER INTEGER INTEGER INTEGER

Returns the current time in UTC as multiple values of year month day day-of-week hour minute second millisecond. Currently millisecond will always be zero (even in Java where it is technically available).

**get-local-time-zone ()**: FLOT

Returns the current time zone offset from UTC as a float, considering the effects of daylight savings time.

**make-current-date-time ()**: CALENDAR-DATE

Create a calendar date with current time and date.

**make-date-time ((year INTEGER) (month INTEGER) (day INTEGER) (hour INTEGER) (minute INTEGER) (second INTEGER) (millis INTEGER) (timezone FLOAT))**: CALENDAR-DATE

Create a calendar date with the specified components. year must be the complete year (i.e., a year of 98 is 98 A.D in the 1st century). timezone is a real number in the range -12.0 to +14.0 where UTC is zone 0.0; The number is the number of hours to add to UTC to arrive at local time.

**parse-date-time ((date-time-string STRING) (start INTEGER) (end INTEGER) (error-on-mismatch BOOLEAN))**: DECODED-DATE-DIME

Tries very hard to make sense out of the argument date-time-string and returns a time structure if successful. If not, it returns null. If error-on-mismatch is true, parse-date-time will signal an error instead of returning null. Default values are 00:00:00 local time on the current date.

**decode-calendar-date (date (timezone FLOAT))**: [Method on CALENDAR-DATE]

DECODED-DATE-DIME

Returns a decoded time object for date interpreted in timezone timezone is the number of hours added to UTC to get local time. It is in the range -12.0 to +14.0 where UTC is zone 0.0.

**encode-calendar-date (time-structure)**: [Method on DECODED-DATE-DIME]

CALENDAR-DATE

Returns a calendar date object for time-structure.

**calendar-date-to-string**

Not yet implemented.

**string-to-calendar-date ((input STRING))**: CALENDAR-DATE

Returns a calendar date object representing the date and time parsed from the input string. If no valid parse is found, null is returned.

**relative-date-to-string**

Not yet implemented.
**compute-calendar-date** ((julian-day integer)) : integer integer [Function]
integer keyword
Returns the YEAR, MONTH, DAY, DAY-OF-WEEK on which the given julian-day begins at noon.

**compute-day-of-week** ((yyy integer) (mm integer) (dd integer)) : keyword [Function]
Returns the day of the week for yyyy-mm-dd.

**compute-day-of-week-julian** ((julian-day integer)) : keyword [Function]
Returns the day of the week for julian-day

**compute-julian-day** ((yyy integer) (mm integer) (dd integer)) : integer [Function]
integer
Returns the Julian day that starts at noon on yyyy-mm-dd. yyyy is the year. mm is the month. dd is the day of month. Negative years are B.C. Remember there is no year zero.

**compute-next-moon-phase** ((n integer) (phase keyword)) : integer float [Function]
integer
Returns the Julian Day and fraction of day of the Nth occurrence since January 1, 1900 of moon PHASE. PHASE is one of :NEW-MOON, :FIRST-QUARTER, :FULL-MOON, :LAST-QUARTER

**decode-time-in-millis** ((time integer)) : integer integer integer [Function]
integer
Returns multiple values of hours, minutes, seconds, milliseconds for time specified in milliseconds.

**julian-day-to-modified-julian-day** ((julian-day integer)) : integer [Function]
Returns the modified Julian day during which julian-day starts at noon.

**modified-julian-day-to-julian-day** ((modified-julian-day integer)) : [Function]
integer
Returns the modified Julian day during which julian-day starts at noon.

**time-add** ((t1 date-time-object) (t2 date-time-object)) : date-time-object [Function]
Adds t1 to t2. If one of t1 or t2 is a calendar date, then the result is a calendar date. If both t1 and t2 are relative dates, then the result is a relative date. t1 and t2 cannot both be calendar dates.

**time-divide** ((t1 time-duration) (t2 object)) : object [Function]
Divides the relative date t1 by t2. t2 must be either a relative date or a wrapped number. If t2 is a relative date, then the return value will be a wrapped float. If t2 is a wrapped number, then the return value will be a relative date.

**time-multiply** ((t1 object) (t2 object)) : time-duration [Function]
Multiplies a relative date by a wrapped number. One of t1 or t2 must be a relative date and the other a wrapped number.
time-subtract ((t1 DATE-TIME-OBJECT) (t2 DATE-TIME-OBJECT)) :
  DATE-TIME-OBJECT
Subtract t2 from t1. If t1 is a calendar date, then t2 can be either a calendar date (in which case the return value is a relative date) or it can be a relative date (in which case the return value is a calendar date). If t1 is a relative date, then t2 must also be a relative date and a relative date is returned.

get-ticktock () : TICKTOCK
Return the current CPU time. If the current OS/Language combination does not support measuring of CPU time, return real time instead. Use ticktock-difference to measure the time difference between values returned by this function. This is an attempt to provide some platform independent support to measure (at least approximately) consumed CPU time.

ticktock-difference ((t1 TICKTOCK) (t2 TICKTOCK)) : FLOAT
The difference in two TICKTOCK time values in seconds where t1 is the earlier time. The resolution is implementation dependent but will normally be some fractional value of a second.

ticktock-resolution () : FLOAT
The minimum theoretically detectable resolution of the difference in two TICKTOCK time values in seconds. This resolution is implementation dependent. It may also not be realizable in practice, since the timing grain size may be larger than this resolution.

sleep ((seconds FLOAT)) :
The program will sleep for the indicated number of seconds. Fractional values are allowed, but the results are implementation dependent: Common Lisp uses the fractions natively, Java with a resolution of 0.001, and C++ can only use integral values.

6.18 XML Support

make-xml-element (((name STRING) (namespace-name STRING))
  (namespace STRING)) : XML-ELEMENT
Creates and interns an XML element object name using namespace-name to refer to namespace. If namespace is null, then the element will be interned in the null namespace. namespace must otherwise be a URI.

make-xml-global-attribute (((name STRING)
  (namespace-name STRING) (namespace STRING)) : XML-GLOBAL-ATTRIBUTE
Creates and interns an XML global attribute object with name using namespace-name to refer to namespace. namespace must be a URI.

make-xml-local-attribute (((name STRING) (element XML-ELEMENT)) :
  XML-LOCAL-ATTRIBUTE
Make an XML-LOCAL-ATTRIBUTE named name associated with element.

get-xml-tag (((expression CONS)) : XML-ELEMENT
Return the XML tag object of an XML expression.
**get-xml-attributes** (\((expression \ CONS)\)) : CONS

Return the list of attributes of an XML expression (may be empty).

**get-xml-content** (\((expression \ CONS)\)) : CONS

Return the list of content elements of an XML expression (may be empty).

**get-xml-cdata-content** (\((form \ CONS)\)) : STRING

Return the CDATA content of a CDATA form. Does NOT make sure that form actually is a CDATA form, so bad things can happen if it is given wrong input.

**xml-declaration?** (\((item \ object)\)) : BOOLEAN

Return true if item is an XML declaration object

**xml-element?** (\((item \ object)\)) : BOOLEAN

Return true if item is an XML element object

**xml-attribute?** (\((item \ object)\)) : BOOLEAN

Return true if item is an XML attribute object

**xml-cdata?** (\((item \ object)\)) : BOOLEAN

Return true if item is an XML CDATA tag object

**xml-cdata-form?** (\((form \ object)\)) : BOOLEAN

Return true if form is a CONS headed by a CDATA tag

**xml-element-match?** (\(tag \ (name \ string)\) (namespace \ string)) : BOOLEAN

Returns true if tag is an XML element with the name name in namespace namespace. Note that namespace is the full URI, not an abbreviation. Also, namespace may be null, in which case tag must not have a namespace associated with it.

**xml-attribute-match?** (\(attribute \ (name \ string)\) (namespace \ string)) : BOOLEAN

Return true if attribute is an XML attribute with name name in namespace namespace. Note that namespace is the full URI, not an abbreviation. Also, namespace may be null, in which case attribute must not have a namespace associated with it.

**xml-attribute-match?** (\(attribute \ (name \ string)\) (namespace \ string)) : BOOLEAN

Return true if attribute is a global XML attribute with name name in namespace namespace. Note that namespace is the full URI, not an abbreviation. Also, namespace may be null, in which case attribute must not have a namespace associated with it.

**xml-attribute-match?** (\(attribute \ (name \ string)\) (namespace \ string)) : BOOLEAN

Return true if attribute is a local XML attribute with name name. Note that namespace must be null and that the attributes parent element element is not considered by the match. To take the parent element into account use xml-local-attribute-match?.


xml-local-attribute-match? (attribute xml-local-attribute) (name string) (element-name string) (element-namespace string) : boolean

Return true if attribute is a local attribute with name and whose parent element matches element-name and element-namespace.

xml-lookup-attribute ((attributes cons) (name string) (namespace string)) : string

Find the XML attribute in attributes with name and namespace and return its value. Note that it is assumed that all attributes come from the same known tag, hence, the parent elements of any local attributes are not considered by the lookup.

xml-tag-case ((item object) &body (clauses cons)) : object

A case form for matching item against XML element tags. Each element of clauses should be a clause with the form ("tagname" ...) or ("tagname" "namespace-uri") ... The clause heads can optionally be symbols instead of strings. The key forms the parameters to the method xml-element-match?, with a missing namespace argument passed as NULL.

The namespace argument will be evaluated, so one can use bound variables in place of a fixed string. As a special case, if the namespace argument is :ANY, then the test will be done for a match on the tag name alone.

read-xml-expression ((stream input-stream) (start-tag object)) :

object boolean

Read one balanced XML expression from stream and return its s-expression representation (see xml-token-list-to-s-expression). If startTagName is non-'null', skip all tags until a start tag matching start-tag is encountered. XML namespaces are ignored for outside of the start tag. Use s-expression representation to specify start-tag, e.g., (KIF (:version "1.0")). The tag can be an XML element object, a symbol, a string or a cons. If the tag is a cons the first element can also be (name namespace) pair.

Return true as the second value on EOF.

CHANGE WARNING: It is anticipated that this function will change to a) Properly take XML namespaces into account and b) require XML element objects instead of strings as the second argument. This change will not be backwards-compatible.

xml-expressions ((stream input-stream) (regionTag object)) :

XML-EXPRESSION-ITERATOR

Return an XML-expression-iterator (which see) reading from stream. regionTag can be used to define delimited regions from which expressions should be considered. Use s-expression representation to specify regionTag, e.g., (KIF (:version "1.0")). The tag can be an XML element object, a symbol, a string or a cons. If the tag is a cons the first element can also be (name namespace) pair.

print-xml-expression ((stream output-stream) (xml-expression cons) (indent integer)) :

Prints xml-expression on stream. Indentation begins with the value of indent. If this is the null integer, no indentation is performed. Otherwise it should normally be specified as 0 (zero) for top-level calls.
It is assumed that the xml-expression is a well-formed CONS-list representation of an XML form. It expects a form like that form returned by read-XML-expression. Also handles a list of xml forms such as that returned by XML-expressions. In that case, each of the forms is indented by indent spaces.

**reset-xml-hash-tables ()**:  
[Function]  
Resets Hashtables used for interning XML elements and global attribute objects. This will allow garbage collection of no-longer used objects, but will also mean that newly parsed xml elements and global attributes will not be eq? to already existing ones with the same name.

### 6.19 Miscellaneous

This is a catch-all section for functions and methods that haven’t been categorized yet into any of the previous sections. They are in random order and many of them will never be part of the official STELLA interface. So beware!

**operating-system ()**: KEYWORD  
[Function]  
Not documented.

**1d-aref (self (i INTEGER))**: (LIKE ANY-VALUE SELF)  
[Method on DIMENSIONAL-ARRAY-MIXIN]  
Return the element of self at position [i].

**1d-aref-address (self (i INTEGER))**: INTEGER  
[Method on DIMENSIONAL-ARRAY-MIXIN]  
Return the 1D address of the element at position [i]. This is useful for fast element-wise iteration that doesn’t need arithmetic.

**1d-aref-setter (self (value (LIKE (ANY-VALUE SELF))) (i INTEGER))**: (LIKE (ANY-VALUE SELF))  
[Method on DIMENSIONAL-ARRAY-MIXIN]  
Set the element of self at position [i] to value and return the result.

**2d-aref (self (i INTEGER) (j INTEGER))**: (LIKE ANY-VALUE SELF)  
[Method on 2-DIMENSIONAL-ARRAY-MIXIN]  
Return the element of self at position [i, j].

**2d-aref-address (self (i INTEGER) (j INTEGER))**: INTEGER  
[Method on 2-DIMENSIONAL-ARRAY-MIXIN]  
Return the 1D address of the element at position [i, j]. This is useful for fast element-wise iteration that doesn’t need arithmetic.

**2d-aref-setter (self (value (LIKE (ANY-VALUE SELF))) (i INTEGER) (j INTEGER))**: (LIKE ANY-VALUE SELF)  
[Method on 2-DIMENSIONAL-ARRAY-MIXIN]  
Set the element of self at position [i, j] to value and return the result.

**abs (x)**: LONG-INTEGER  
[Method on LONG-INTEGER]  
Return the absolute value of x.
activate-demon ((demon demon)) : [Function]
  Install demon in the location(s) specified by its internal structure.

active? (self) : BOOLEAN [Method on POLYMORPHIC-RELATION]
  True if self or a superslot of self is marked active.

add-configuration-property ((property string) (value object) (configuration configuration-table)) : OBJECT
  Add value to property in configuration and return it. If a previous value exists add value to the end (listify the old value if it is not yet a list). Otherwise, create a new list containing value. Use the global system configuration table if configuration is NULL.

add-hook ((hookList hook-list) (hookFunction symbol)) : [Function]
  Insert the function named hookFunction into hookList.

add-trace (&rest (keywords generalized-symbol)) : LIST [N-Command]
  Enable trace messages identified by any of the listed keywords. After calling (add-trace <keyword>) code guarded by (trace-if <keyword> ...) will be executed when it is encountered.

advance-past-whitespace ((source string) (start integer)) : [Function]
  INTEGER
  Returns the first index into source, starting from start, of the first character that is not white space.

all-classes ((module module) (local? boolean)) : (iterator of class) [Function]
  Iterate over all classes visible from module. If local?, return only classes interned in module. If module is null, return all classes interned everywhere.

all-contexts () : (iterator of context) [Function]
  Return an iterator that generates all contexts.

all-defined? (&body (forms cons)) : OBJECT [Macro]
  Evaluate each of the forms in forms, and return TRUE if none of them are NULL.

all-functions ((module module) (local? boolean)) : (iterator of function) [Function]
  Iterate over all functions visible from module. If local?, return only functions bound to symbols interned in module. If module is null, return all functions defined everywhere.

all-included-modules ((self module)) : (iterator of module) [Function]
  Generate a sequence of all modules included by self, inclusive, starting from the highest ancestor and working down to self (which is last).

all-methods ((module module) (local? boolean)) : (iterator of method-slot) [Function]
  Iterate over all methods visible from module. If local?, return only methods interned in module. If module is null, return all methods interned everywhere.

all-modules () : (iterator of module) [Function]
  Return an iterator that generates all modules.
**all-public-functions** \((\text{module function}) (\text{local? boolean})\) :

(ITERATOR OF FUNCTION)

Iterate over all functions visible from \text{module}. If \text{local?}, return only functions bound to symbols interned in \text{module}. If module is null, return all functions defined everywhere.

**all-public-methods** \((\text{module function}) (\text{local? boolean})\) :

(ITERATOR OF METHOD-SLOT)

Iterate over all public methods visible from \text{module}. If \text{local?}, return only methods interned in \text{module}. If module is null, return all methods interned everywhere.

**all-required-systems** \((\text{system-name string})\) :

(CONS OF STRING-WRAPPER)

Returns a CONS of all of the systems required by \text{system-name}

**all-slots** \((\text{module function}) (\text{local? boolean})\) :

(ITERATOR OF SLOT)

Iterate over all slots visible from \text{module}. If \text{local?}, return only methods interned in \text{module}. If module is null, return all methods interned everywhere.

**all-subcontexts** \((\text{context context}) (\text{traversal keyword})\) :

(ALL-PURPOSE-ITERATOR OF CONTEXT)

Return an iterator that generates all subcontexts of \text{self} (not including \text{self}) in the order specified by \text{traversal} (one of :preorder, :inorder, :postorder or :topdown).

**all-surrogates** \((\text{module function}) (\text{local? boolean})\) :

(ITERATOR OF SURROGATE)

Iterate over all surrogates visible from \text{module}. If \text{local?}, return only surrogates interned in \text{module}. If module is null, return all surrogates interned everywhere.

**all-symbols** \((\text{module function}) (\text{local? boolean})\) :

(ITERATOR OF SYMBOL)

Iterate over all symbols visible from \text{module}. If \text{local?}, return only symbols interned in \text{module}. If module is null, return all symbols interned everywhere.

**all-variables** \((\text{module function}) (\text{local? boolean})\) :

(ITERATOR OF GLOBAL-VARIABLE)

Iterate over all variables visible from \text{module}. If \text{local?}, return only variables bound to symbols interned in \text{module}. If module is null, return all variables defined everywhere.

**allocate-iterator** \((\text{self})\) :

(LIKE SELF)

Iterator objects return themselves when asked for an iterator (they occupy the same position as a collection within a foreach statement).

**allocate-iterator** \((\text{self})\) :

(ITERATOR OF (LIKE ANY-VALUE SELF))

Alias for clone-memoized-iterator.

**allocation** \((\text{self})\) :

KEYWORD

Return the most specific :allocation facet, or :instance if all inherited values are NULL.

**apply** \((\text{code function-code}) (\text{arguments (cons of object)})\) :

OBJECT

Apply \text{code} to \text{arguments}, returning a value of type OBJECT.
apply-boolean-method ((code method-code) (arguments (cons of object))) : BOOLEAN
Apply code to arguments, returning a value of type BOOLEAN.

apply-float-method ((code method-code) (arguments (cons of object))) : FLOAT
Apply code to arguments, returning a value of type FLOAT.

apply-integer-method ((code method-code) (arguments (cons of object))) : INTEGER
Apply code to arguments, returning a value of type INTEGER.

apply-long-integer-method ((code method-code) (arguments (cons of object))) : LONG-INTEGER
Apply code to arguments, returning a value of type LONG-INTEGER.

apply-method ((code method-code) (arguments (cons of object))) : OBJECT
Apply code to arguments, returning a value of type OBJECT.

apply-string-method ((code method-code) (arguments (cons of object))) : STRING
Apply code to arguments, returning a value of type STRING.

base60-to-float (((l (cons of number-wrapper))) : FLOAT
Converts (x y z) into a float. The return value is x + y/60 + z/3600. This can be used to convert from Degree-Minute-Second to decimal degrees or from Hour-Minute-Second format to decimal hours.

break-program ((message string)) :
Interrupt the program and print message. Continue after confirmation with the user.

bump-log-indent () :
Increase the indentation level for subsequent log messages.

byte-array-read-sequence ((buffer tokenizer-byte-array) (stream input-stream) (start integer) (end integer)) : INTEGER
Read from stream filling buffer between start and end (depending on how many characters are available). Return the actual end pointer to the input read into buffer. EOF is indicated by the return value being equal to start.

byte-array-write-sequence ((buffer tokenizer-byte-array) (stream native-output-stream) (start integer) (end integer)) :
Write from buffer to stream, using data in the buffer starting at position start stopping just before end.

calendar-date-to-date-string (date (timezone float) (numeric-month? boolean)) : STRING
Returns the date part of the string representation of date adjusted for timezone. Format is YYYY-MMM-DD, where MMM is a three letter English abbreviation of the month if numeric-month? is false and a two digit numeric value if numeric-month? is true. The value false is recommended.
Chapter 6: Library Functions

```
Chapter 6: Library Functions

**calendar-date-to-iso8601-string** (date [Method on `calendar-date`] (timezone float) (include-timezone? boolean)) : string
Returns an ISO-8601 string representation of date adjusted for timezone. The Format is YYYY-MM-DDTh:mm:ss z:zz. The timezone as an offset hh:mm is included if include-timezone? is true.
Recommended values for the flag is true.

**calendar-date-to-string** (date (timezone float) (numeric-month? boolean) (include-timezone? boolean)) : string
Returns a string representation of date adjusted for timezone. The Format is YYYY-MM-DD hh:mm:ss z.z, where MMM is a three letter English abbreviation of the month if numeric-month? is false and a two digit numeric value if numeric-month? is true. The timezone as a float offset z.z is included if include-timezone? is true. Recommended values for the flags are false and true respectively.

**calendar-date-to-time-string** (date (timezone float) (include-timezone? boolean) (include-millis? boolean) (pad-hours? boolean)) : string
Returns the time part of the string representation of date adjusted for timezone. The timezone is included in the string if include-timezone? is true. The value true is recommended. Milliseconds will be included if include-millis? is true. Hours will be zero-padded to length 2 if pad-hours? is true.

**call-clear-module** (&rest (name name)) :
[Command]
Destroy all objects belonging to module name or any of its children. If no name is supplied, the current module will be cleared after confirming with the user. Important modules such as STELLA are protected against accidental clearing.

**cast** ((value object) (type type)) : object
[Function]
Perform a run-time type check, and then return value.

**ccc** (&rest (name name)) : context
[N-Command]
Change the current context to the one named name. Return the value of the new current context. cc is a no-op if the context reference cannot be successfully evaluated. In CommonLisp, if the new context is case sensitive, then change the readable case to :INVERT, otherwise to :UPCASE.

**cl-slot-value** ((object object) (slotName string) (dontConvert? boolean)) : lisp-code
[Function]
Lookup slot slotName on object and return the lispified slot value (see `lispify`). If dontConvert? is TRUE, the returned slot value will not be lispified. Generate a warning if no such slot exists on object. In a call directly from Lisp slotName can also be supplied as a Lisp symbol.

**cl-slot-value-setter** ((object object) (slotName string) (value lisp-code) (dontConvert? boolean)) : lisp-code
[Function]
Lookup slot slotName on object and set its value to the stellafied value (see `stellafy`). If dontConvert? is TRUE, value will not be stellafied before it gets
```
assigned. Generate a warning if no such slot exists on object, or if value has the wrong type. In a call directly from Lisp slotName can also be supplied as a Lisp symbol.

**cl-translate-file** (\((file \, \text{file-name}) \, (\text{relative?} \, \text{boolean})\)):

[Function]

Translate a Stella file to Common-Lisp. If relative?, concatenate root directory to file.

**cl-translate-system** (\((\text{system-name} \, \text{string})\)):

[Function]

Translate a Stella system named system-name to Common Lisp.

**cleanup-unfinalized-classes** ():

[Function]

Remove all finalized classes from *UNFINALIZED-CLASSES*, and set *NEWLY-UNFINALIZED-CLASSES?* to false.

**clear** (\(\text{self}\)):

[Method on VECTOR-SEQUENCE]

Clear self by setting its active length to zero.

**clear** (\(\text{self}\)):

[Method on HEAP]

Clear self by setting its active length to zero.

**clear-configuration-property** (\((\text{property} \, \text{string})\), \((\text{configuration} \, \text{configuration-table})\)):

[Function]

Remove property in configuration and return the previous value. Use the global system configuration table if configuration is NULL.

**clear-recycle-list** (\((\text{list} \, \text{recycle-list})\)):

[Function]

Reset list to its empty state.

**clear-recycle-lists** ():

[Function]

Reset all currently active recycle lists to their empty state.

**clear-system** (\((\text{name} \, \text{string})\)):

[Function]

Clears out the system definition named name. If name is null, then clear out all system definitions. This function is useful when changes have been made to the system definition, and one wants to have it reloaded from the standard location in the file system.

**clear-trace** ():

[Command]

Disable all tracing previously enabled with add-trace.

**clone-memoized-iterator** (\((\text{self} \, \text{memoizable-iterator})\)):

[Function]

Clone the memoized iterator self so it can be used to iterate over the collection represented by self, while allowing to iterate over it multiple times via multiple clones.

**clv** (\((\text{code} \, \text{object})\)):

[Macro]

Convenience macro to splice Lisp expressions into STELLA expressions without upsetting the translator during interactive sessions. If code is a currently bound Lisp variable, this tries to infer the type of the resulting expression from the object code is bound to and generates an appropriate cast. clv stands for Common-Lisp Value or Verbatim.
**coerce-&rest-to-cons** ((restVariable symbol)) : object

Macro

Coerce the argument list variable restVariable into a CONS list containing all its elements (uses argument list iteration to do so). If restVariable already is a CONS due to argument listification, this is a no-op.

**coerce-to-float** ((number number-wrapper)) : float

Function

Coerce number to a float value.

**coerce-to-hash-set** ((self object) (equalTest? boolean)) : hash-set

Function

Coerce the collection self into a HASH-SET. Use an equal test if equalTest? is TRUE (equalTest? will be ignored if self already is a HASH-SET).

**coerce-to-symbol** ((name NAME)) : generalized-symbol

Function

Return the (generalized) symbol represented by name. Return null if name is undefined or does not represent a string.

**collect** (&body (body cons)) : object

Macro

Use a VRLET to collect values. Input can have one of the following forms:

(collect <var> in <expression> [where <test> <var>])
(collect <collect-expression>
   foreach <var> in <expression>
   {as ...}*  
   [where <test> <var>]
   [do ...])

The second form really accepts an arbitrary foreach expression following the foreach keyword.

**collection-valued?** (self) : boolean

Method on SLOT

True if slot values are collections.

**command?** ((method method-slot)) : boolean

Function

Return true if method is an evaluable command.

**component?** (self) : boolean

Method on STORAGE-SLOT

True if fillers of this slot are components of the owner slot, and therefore should be deleted if the owner is deleted.

**compose-namestring**

((name-components (cons of string-wrapper)) &rest (options object)) : string

Function

name-components is a cons to be processed into a namestring. :prefix and :suffix are strings that will NOT be case-converted. :case is one of :UPCASE :TitleCase :titleCaseX :downcase :Capitalize default is :TitleCase :separator is a string that should separate word elements. It does not separate the prefix or suffix. Default is "" :translation-table should be a STRING-HASH-TABLE hash table that strings into their desired printed representation as a string. In general the argument will be strings, but that is not strictly necessary.
**compose-namestring-full** ((strings (CONS OF STRING-WRAPPER)) (prefix STRING) (suffix STRING) (outputcase KEYWORD) (outputseparator STRING) (translationtable STRING-HASH-TABLE) (useacronymheuristics? BOOLEAN)) : STRING  
Non-keyword version of `compose-namestring`, which will probably be easier to use when called from non-Lisp languages.

**compute-module-and-bare-name** ((name STRING)) : MODULE  
STRING  
Compute the module indicated by the STELLA name `name` and return it. Return the bare symbol name as the second value. `name` does not necessarily have to be qualified in which case the current module is returned. `name` is assumed to be the printed representation of a STELLA symbol, surrogate or keyword.

**configure-stella** ((file FILE-NAME)) :  
Perform STELLA run-time configuration. If supplied, load the configuration file `file` first which should be supplied with a physical pathname.

**consify** (self) : CONS  
If `object` is a CONS, return it. Otherwise, return a singleton cons list containing it.

**consify-command-line-arguments** ((count INTEGER) (arguments ARRAY () OF STRING)) : (CONS OF STRING-WRAPPER)  
Convert `count` command line arguments into a CONS list.

**continuable-error** (&body (body cons)) : OBJECT  
Signal error message, placing non-string arguments in quotes.

**copy** (self) : (VECTOR-SEQUENCE OF (LIKE ANY-VALUE self))  
Return a copy of the vector sequence `self`.

**copy** (self) : (CUSTOM-VECTOR-SEQUENCE OF (LIKE (ANY-VALUE self)))  
Return a copy of the vector sequence `self`.

**cpp-translate-system** ((systemName STRING)) :  
Translate the system `systemName` to C++.

**cpptrans** ((statement OBJECT)) :  
Translate `statement` to C++ and print the result.

**create-derived-list** ((self list)) : LIST  
Create a new list object with the same type as `self`.

**create-object** ((type TYPE) &rest (initial-value-pairs OBJECT)) : OBJECT  
Funcallable version of the `new` operator. Return an instance of the class named by `type`. If `initial-value-pairs` is supplied, it has to be a key/value list similar to what’s accepted by `new` and the named slots will be initialized with the supplied values. Similar to `new`, all required arguments for `type` must be included. Since all the slot
initialization, etc. is handled dynamically at run time, \texttt{create-object} is much slower than \texttt{new}; therefore, it should only be used if \texttt{type} cannot be known at translation time.

\textbf{deactivate-demon} \((\texttt{(demon \texttt{DEMON})})\) : \[\text{Function}\]
Detach \texttt{demon} from the location(s) specified by its internal structure.

\textbf{decoded-date-time-to-iso8601-string} \[\text{Method on \texttt{DECODED-DATE-TIME}}\]
\((\texttt{date}) : \texttt{STRING})\)
Returns an ISO-8601 string representation of \texttt{date} The Format is YYYY-MM-DDThh:mm:ss\(+zz:zz\), with the string stopping at when a null value is first encountered. The time zone will only be included (if present) if a time value is given.

\textbf{decompose-namestring} \((\texttt{(namestring \texttt{STRING})}) \&\texttt{rest (options \texttt{OBJECT})}) : (\texttt{CONS OF \texttt{STRING-WRAPPER}}) \[\text{Function}\]
Keyword options:  
\texttt{:break-on-cap} one of \texttt{:YES \texttt{:NO}}  
\texttt{:CLEVER} default is \texttt{:CLEVER}
\texttt{:break-on-number} one of \texttt{:YES \texttt{:NO}}  
\texttt{:CLEVER} default is \texttt{:CLEVER}
\texttt{:break-on-separators} string default is "\texttt{- _}" 

DECOMPOSE-NAMESTRING returns a \texttt{CONS} of \texttt{STRING-WRAPPERS} that are the decomposition of the input \texttt{STRING}. The arguments are used as follows: \texttt{namestring} is the input string.  
\texttt{:break-on-cap} is a keyword controlling whether changes in capitalization is used to indicate word boundaries. If \texttt{:YES}, then all capitalization changes delineate words. If \texttt{:CLEVER}, then unbroken runs of capitalized letters are treated as acronyms and remain grouped. If \texttt{:NO or NULL}, there is no breaking of words based on capitalization.  
\texttt{:break-on-number} is a flag controlling whether encountering a number indicates a word boundary. If \texttt{:YES}, then each run of numbers is treated as a word separate from surrounding words. If \texttt{:CLEVER}, then an attempt is made to recognize ordinal numbers (ie, \texttt{101st}) and treat them as separate words. If \texttt{:NO or NULL}, there is no breaking of words when numbers are encountered.  
\texttt{:break-on-separators} A string of characters which constitute word delimiters in the input word. This is used to determine how to break the name into individual words. Defaults are \texttt{space, - and _}.

\textbf{decompose-namestring-full} \((\texttt{(namestring \texttt{STRING})})\)  
\((\texttt{break-on-cap \texttt{KEYWORD}) (break-on-number \texttt{KEYWORD}) (break-on-separators \texttt{STRING})}) : (\texttt{CONS OF \texttt{STRING-WRAPPER}}) \[\text{Function}\]
Non-keyword version of \texttt{decompose-namestring}, which will probably be easier to use when called from non-Lisp languages.

\textbf{default-form} \((\texttt{self}) : \texttt{OBJECT}) \[\text{Method on \texttt{STORAGE-SLOT}}\]
Returns the current value of default expression when the slot has not been assigned a value.

\textbf{defdemon} \((\texttt{(name \texttt{STRING-WRAPPER}) (parameterstree \texttt{CONS})}) \&\texttt{body (optionsandbody \texttt{CONS})}) : \texttt{OBJECT}) \[\text{Macro}\]
Define a demon \texttt{name} and attach it to a class or slot.

\textbf{define-demon} \((\texttt{(name \texttt{STRING}) \&\texttt{rest (options \texttt{OBJECT})}) : \texttt{DEMON}) \[\text{Function}\]
Define a class or slot demon. Options are \texttt{:create, :destroy, :class, :slot, :guard?, :code, :method, :inherit?, and :documentation}.
define-logical-host-property ((host string) (property keyword) (value object)):
Define property with value for the logical host host. As a side-effect, this also defines host as a logical host (both property and value can be supplied as NULL). If :ROOT-DIRECTORY is specified, all pathnames with host are assumed to be relative to that directory (even if they are absolute) and will be rerooted upon translation. :ROOT-DIRECTORY can be a logical or physical pathname. If :LISP-TRANSLATIONS is specified, those will be used verbatimely as the value of (CL:logical-pathname-translations host) if we are running in Lisp, which allows us to depend on the native CL:translate-logical-pathname for more complex translation operations.

define-module ((name string) (options cons)) : MODULE
Define or redefine a module named name having the options options. Return the new module.

define-stella-class ((name type) (supers (list of type)) (slots (list of slot)) (options keyword-key-value-list)) : CLASS
Return a Stella class with name name. Caution: If the class already exists, the Stella class object gets redefined, but the native C++ class is not redefined.

define-stella-method-slot ((inputname symbol) (returntypes cons) (function? boolean) (inputParameters cons) (options keyword-key-value-list)) : METHOD-SLOT
Define a new Stella method object (a slot), and attach it to the class identified by the first parameter in inputParameters.

defined? (x) : BOOLEAN
Return true if x is defined (handled specially by all translators).

defined? (x) : BOOLEAN
Return true if x is defined (handled specially by all translators).

defmain ((varList cons) &body (body cons)) : OBJECT
Defines a function called MAIN which will have the appropriate signature for the target translation language. The signature will be: C++: public static int main (int v1, char** v2) {<body>} Java: public static void main (String [] v2) {<body>} Lisp: (defun main (&rest args) <body>) The argument varList must have two symbols, which will be the names for the INTEGER argument count and an array of STRINGs with the argument values. It can also be empty to indicate that no command line arguments will be handled. The startup function for the containing system will automatically be called before body is executed unless the option :STARTUP-SYSTEM? was supplied as FALSE. There can only be one DEFMAIN per module.

defsystem ((name symbol) &rest (options object)) : SYSTEM-DEFINITION
Define a system of files that collectively define a Stella application. Required options are: :directory – the relative path from the respective source/native/binary root directory to the directory containing the system files. Can be a string or a list of strings (do not include directory separators). :files – a list of files in the system, containing
Chapter 6: Library Functions

strings and lists of strings; the latter defines exploded paths to files in subdirectories. Optional options are: :required-systems – a list of systems (strings) that should be loaded prior to loading this system. :cardinal-module – the name (a string) of the principal module for this system. :copyright-header – string with a header for inclusion into all translated files produced by Stella. :lisp-only-files – Like the :files keyword, but these are only included :cpp-only-files in the translation for the specific language, namely :java-only-files Common Lisp, C++ or Java

deleted? (self) : BOOLEAN

Default deleted? method which always returns FALSE. Objects that inherit DYNAMIC-SLOTS-MIXIN also inherit the dynamically-allocated slot deleted-object? which is read/writable with specializations of this method.

describe ((name OBJECT) &rest (mode OBJECT)) :

[N-Command]
Print a description of an object in :verbose, :terse, or :source modes.

describe-object (self (stream OUTPUT-STREAM) (mode KEYWORD)) :

[Method on OBJECT]
Prints a description of self to stream stream. mode can be :terse, :verbose, or :source. The :terse mode is often equivalent to the standard print function.

destroy-class (self) :

[Method on CLASS]
Destroy the Stella class self. Unfinalize its subclasses (if it has any).

destroy-class-and-subclasses ((self CLASS)) :

[Function]
Destroy the Stella class self and all its subclasses.

destructure-defmethod-tree ((method-tree CONS) (options-table KEY-VALUE-LIST)) : OBJECT CONS CONS

[Function]
Return three parse trees representing the name, parameters, and code body of the parse tree method-tree. Fill options-table with a dictionary of method options. Storage note: Options are treated specially because the other return values are subtrees of method-tree, while options-table is a newly-created cons tree. Note also, the parameter and body trees are destructively removed from method-tree.

dictionary ((collectionType TYPE) &rest (alternatingkeysandvalues OBJECT)) : (ABSTRACT-DICTIONARY OF OBJECT OBJECT)

[Function]
Return a dictionary of collectionType containing values, in order. Currently supported collectionTypes are @HASH-TABLE, @STELLA-HASH-TABLE, @KEY-VALUE-LIST, @KEY-VALUE-MAP and @PROPERTY-LIST.

direct-super-classes (self) : (ITERATOR OF CLASS)

[Method on CLASS]
Returns an iterator that generates all direct super classes of self.

disable-memoization () :

[Command]
Enable memoization and use of memoized expression results.

disabled-stella-feature? ((feature KEYWORD)) : BOOLEAN

[Function]
Return true if the STELLA feature is currently disabled.
div (x (y INTEGER)) : INTEGER  
Return the integer quotient from dividing x by y.

div (x (y LONG-INTEGER)) : LONG-INTEGER  
Return the integer quotient from dividing x by y.

drop-hook ((hookList HOOK-LIST) (hookFunction SYMBOL)) :  
Remove the function named hookFunction from hookList.

drop-trace (&rest (keywords GENERALIZED-Symbol)) : list  
Disable trace messages identified by any of the listed keywords. After calling (drop-trace <keyword> ... ) code guarded by (trace-if <keyword> ...) will not be executed when it is encountered.

either ((value1 OBJECT) (value2 OBJECT)) : OBJECT  
If value1 is defined, return that, else return value2.

empty? (x) : BOOLEAN  
Return true if x is the wrapped empty string ""

empty? (self) : BOOLEAN  
Return true if self has length 0.

empty? (self) : BOOLEAN  
Return TRUE if self is empty.

enable-memoizaton () :  
Enable memoization and use of memoized expression results.

enabled-stella-feature? ((feature KEYWORD)) : BOOLEAN  
Return true if the STELLA feature is currently enabled.

ensure-file-does-not-exist ((filename STRING) (context STRING)) :  
Ensures that filename does not exist. If it does, an exception of type FILE-ALREADY-EXISTS-EXCEPTION is thrown with context supplying context for the error message.

ensure-file-exists ((filename STRING) (context STRING)) :  
Ensures that filename exists. If not, an exception of type NO-SUCH-FILE-EXCEPTION is thrown with context supplying context for the error message.

eql-except-in-whitespace? ((s1 STRING) (s2 STRING)) : BOOLEAN  
Return true if the strings s1 and s2 are the same except for the amounts of whitespace separating words. Leading or trailing whitespace is also not considered.

error (&body (body CONS)) : OBJECT  
Signal error message, placing non-string arguments in quotes.

evaluate ((expression OBJECT)) : OBJECT  
Evaluate the expression expression and return the result. Currently, only the evaluation of (possibly nested) commands and global variables is supported. The second return value indicates the actual type of the result (which might have been wrapped), and the third return value indicates whether an error occurred during the evaluation.
Expressions are simple to program in Common Lisp, since they are built into the language, and relatively awkward in Java and C++. Users of either of those languages are more likely to want to call evaluate-string.

**evaluate-string** ((expression string)) : object [Function]
Evaluate the expression represented by expression and return the result. This is equivalent to (evaluate (unstringify expression)).

even? (x) : boolean [Method on INTEGER]
Return true if x is an even number.

even? (x) : boolean [Method on LONG-INTEGER]
Return true if x is an even number.

exception-message ((e native-exception)) : string [Function]
Accesses the error message of the exception e.

**extension** (self) : class-extension [Method on CLASS]
Return the nearest class extension that records instances of the class self.

external-id-head? ((attribute object)) : boolean [Function]
Checks to see if this attribute is the literal marking either a PUBLIC or SYSTEM literal for an XML Elteral ID. (See 4.2.2)

fast-heap-root (self) : (like (any-value self)) [Method on HEAP]
Return the root of self which is assumed to be non-empty.

**fill-in-date-substitution** [Function]

((substitution-list (key-value-list of string-wrapper string-wrapper)))
Fill in substitution-list with template variable substitutions for the names YEAR and DATE which correspond to the current year and date. These substitutions can then be used with substitute-template-variables-in-string

**finalize-classes** () : [Function]
Finalize all currently unfinalized classes.

**finalize-classes-and-slots** () : [Function]
Finalize all currently unfinalized classes and slots.

**finalize-slots** () : [Function]
Finalize all currently unfinalized slots.

find-matching-prefix-length ((string1 string) (start1 integer) (end1 integer) (string2 string) (start2 integer) (end2 integer)) : integer [Function]
Finds the length of the matching prefix strings of string1 and string2, starting at position start1 and start2 respectively. The search will end when end1 or end2 is reached. If either end1 or end2 is null, then they will be set to the length of their respective strings.
**find-mismatch** ((string1 string) (start1 integer) (end1 integer)) (string2 string) (start2 integer) (end2 integer) : integer integer

Finds the first position in each of string1 and string2 where they mismatch, starting at position start1 and start2 respectively. The search will end when end1 or end2 is reached. If either end1 or end2 is null, then they will be set to the length of their respective strings. If there is no mismatch, then null values are returned.

**first-defined** (&body (forms cons)) : object

Return the result of the first form in forms whose value is defined or NULL otherwise.

**float-to-base60** ((x float) (all-integers? boolean)) : (cons of number-wrapper)

Returns a cons of x in a base-60 form. That means the first value will be the integer part of x, the next value the integer value of the fraction part of x times 60 and the third value the fraction part of x times 3600. If all-integers? is true, then the last value will be rounded to an integer. This can be used to convert from decimal degree values to Degree-Minute-Second or from decimal hours to Hour-Minute-Second format.

**fmod** ((x float) (modulus float)) : float

True modulus for floats. Return the result of x mod modulus. Note: In C++ and Java, mod has more overhead than the similar function rem. The answers returned by mod and rem are only different when the signs of x and modulus are different.

**format-with-padding** ((input string) (length integer) (padchar character) (align keyword) (truncate? boolean)) : string

Formats input to be (at least) length long, using padchar to fill if necessary. align must be one of :LEFT, :RIGHT, :CENTER and will control how input will be justified in the resulting string. If truncate? is true, then then an overlength string will be truncated, using the opposite of align to pick the truncation direction.

**free** (self) : [Method on active-object]

Remove all pointers between self and other objects, and then deallocate the storage for self.

**free** (self) : [Method on object]

Default method. Deallocate storage for self.

**free-hash-table-values** (self) : [Method on abstract-hash-table]

Call free on each value in the hash table self.

**frem** ((x float) (y float)) : float

Return the floating point remainder from dividing x by y. The sign of the result is always the same as the sign of x. This has slightly different behavior than the mod function, and has less overhead in C++ and Java, which don’t have direct support for a true modulus function.

**generate-random-uuid** () : string

Generates a random UUID (Type 4), according to the guidelines of IETF RFC 4122 (see http://www.ietf.org/rfc/rfc4122.txt)
Take 16 random bytes (octets), put them all behind each other, for the description the numbering starts with byte 1 (most significant, first) to byte 16 (least significant, last). Then put in the version and variant. To put in the version, take the 7th byte and perform an and operation using 0x0f, followed by an or operation with 0x40. To put in the variant, take the 9th byte and perform an and operation using 0x3f, followed by an or operation with 0x80. To make the string representation, take the hexadecimal presentation of bytes 1-4 (without 0x in front of it) let them follow by a -, then take bytes 5 and 6, - bytes 7 and 8, - bytes 9 and 10, - then followed by bytes 11-16.

**generate-uuid** ((uuid-type KEYWORD)) : STRING  
Generates a UUID of the specified type. Legal types are a subset of the IETF RFC 4122 (see http://www.ietf.org/rfc/rfc4122.txt ) UUID types. Currently supported are: :TYPE-4 :RANDOM A type-4 (random) UUID. These are synonyms.

**get-calendar-date** (date (timezone FLOAT)) : INTEGER INTEGER INTEGER KEYWORD  
Returns multiple values of year, month, day and day of week for date in timezone. timezone is the number of hours added to UTC to get local time. It is in the range -12.0 to +14.0 where UTC is zone 0.0

**get-global-value** ((self SURROGATE)) : OBJECT  
Return the (possibly-wrapped) value of the global variable for the surrogate self.

**get-local-standard-time-zone** () : FLOAT  
Returns the standard time zone offset from UTC as a float, without considering the effects of daylight savings time.

**get-local-time-zone-for-date** ((year INTEGER) (month INTEGER) (day INTEGER) (hour INTEGER) (minute INTEGER) (second INTEGER)) : FLOAT  
Returns the time zone offset from UTC (as a float) that is applicable to the given date. Assumes that the date is one that is valid for the underlying programming language. If not, then returns 0.0

**get-quoted-tree** ((tree-name STRING) (modulename STRING)) : CONS  
Return the quoted tree with name tree-name.

**get-slot** ((self STANDARD-OBJECT) (slot-name SYMBOL)) : SLOT  
Return the slot named slot-name on the class representing the type of self.

**get-stella-class** (class-name (error? BOOLEAN)) : CLASS  
Return a class with name class-name. If none exists, break if error?, else return null.

**get-stella-class** (class-name (error? BOOLEAN)) : CLASS  
Return a class with name class-name. If non exists, break if error?, else return null.

**get-stella-class** (class-name (error? BOOLEAN)) : CLASS  
Return a class with name class-name. If none exists, break if error?, else return null.
Chapter 6: Library Functions

get-time (date (timezone FLOAT)) : INTEGER

Returns multiple values of hours, minutes, seconds, milliseconds for the calendar date date in timezone. timezone is the number of hours added to UTC to get local time. It is in the range -12.0 to +14.0 where UTC is zone 0.0

get-token-float () : OBJECT

User-level macro to access the most recently parsed token as a float. This assumes correct signed float syntax and only checks for overflows. The main benefit for this is that it doesn’t generate strings and wrappers. Float parsing and conversion is generally hairy and we are probably not covering all special cases here; but we are fast :-)

get-token-integer () : OBJECT

User-level macro to access the most recently parsed token as an integer. This assumes correct signed integer syntax and only checks for overflows.

global-variable-type-spec ((global GLOBAL-VARIABLE)) : TYPE-SPEC

Return the type spec for the global variable global.

hash-string ((string STRING) (seedCode INTEGER)) : INTEGER

Generate a hash-code for string and return it. Two strings that are equal but not eq will generate the same code. The hash-code is based on seedCode which usually will be 0. However, seedCode can also be used to supply the result of a previous hash operation to achieve hashing on sequences of strings without actually having to concatenate them.

heap-root (self) : (LIKE (ANY-VALUE self))

Return the root of self (NULL if self is empty).

heapify (self)

Restore the heap property of self according to its predicate. Normally, this is not needed, since insert operations preserve the heap property. However, this can be useful after bulk insertion of values or if predicate has been changed.

help-advance-past-whitespace ((source STRING) (start INTEGER) (end INTEGER)) : INTEGER

Helper for advance-past-whitespace that requires end to be properly set.

help-find-matching-prefix-length ((string1 STRING) (start1 INTEGER) (end1 INTEGER) (string2 STRING) (start2 INTEGER) (end2 INTEGER)) : INTEGER

Helping function for find-matching-prefix that requires end1 and end2 to be properly set up.
help-get-stella-module ((pathName string) (error? boolean)) : [Function]
MODULE
Return the module located at pathName, or null if no such module exists. The
search looks at ancestors and top-most (cardinal) modules. If error? is true, throw
an exception if no module is found.

help-print-outline (top (stream output-stream) (current-depth integer) (depth integer) (named? boolean)) :
[Method on class]
Helper method for print-outline

help-print-outline (top (stream output-stream) (current-depth integer) (depth integer) (named? boolean)) :
[Method on module]
Helper method for print-outline

help-print-outline (top (stream output-stream) (current-depth integer) (depth integer) (named? boolean)) :
[Method on object]
Helper method for print-outline

help-print-outline (top (stream output-stream) (current-depth integer) (depth integer) (named? boolean)) :
[Method on slot]
Helper method for print-outline

help-print-outline (top (stream output-stream) (current-depth integer) (depth integer) (named? boolean)) :
[Method on context]
Helper method for print-outline

home-module (self) : module
[Method on object]
Return the home module of self.

if-output-language ((language keyword) (thenForm object) (elseForm object)) : object
[Macro]
Expand to thenForm if the current translator output language equals language. Other-
wise, expand to elseForm. This can be used to conditionally translate Stella code.

if-stella-feature ((feature keyword) (thenForm object) (elseForm object)) : object
[Macro]
Expand to thenForm if feature is a currently enabled STELLA environment feature.
Otherwise, expand to elseForm. This can be used to conditionally translate Stella
code.

ignore (&body (variables cons)) : object
[Macro]
Ignore unused variables with NoOpsetq statements.

incrementally-translate ((tree object)) : object
[Function]
Translate a single Stella expression tree and return the result. For C++ and Java print
the translation to standard output and return NIL instead.

indent-outline ((current-depth integer) (stream output-stream)) : [Function]
Helper function that indents outline printings for level current-depth on stream using
the value of the global variable *OUTLINE-INDENT-STRING*
inform (&body (body cons)) : OBJECT  [Macro]
Print informative message, placing non-string arguments in quotes, and terminating
with a newline.

initial-value (self) : OBJECT  [Method on CLASS]
Return an initial value for the class self.

initial-value (self) : OBJECT  [Method on STORAGE-SLOT]
Return an initial value for self, or null. The initial value can be defined by the slot
itself, inherited from an equivalent slot, or inherit from the :initial-value option for
the class representing the type of self.

initialize-array (self) : [Method on DIMENSIONAL-ARRAY-MIXIN]
  (initialValue (like (any-value self)))) :
Initialize the elements of self with initialValue.

initialize-hash-table (self) : [Method on STELLA-HASH-TABLE]
Initialize the STELLA hash table self. This is a no-op and primarily exists to shadow
the standard initializer inherited from ABSTRACT-HASH-TABLE. STELLA hash
tables are initialized at the first insertion operation.

initially (self) : OBJECT  [Method on STORAGE-SLOT]
Defines the value of a slot before it has been assigned a value.

insert (self) : [Method on CUSTOM-VECTOR-SEQUENCE]
  (value (like (any-value self)))) :
Append value to the END of the sequence self. Resize the array if necessary.

insert (self (value (like (any-value self)))) :
Insert value into self and restore the heap property. Signal an error if there is no
more room in self. Maintains self as a Min-heap if self's predicate has < semantics;
otherwise as a Max-heap.

insert-if-better (self (value (like (any-value self)))) :
Insert value into self and restore the heap property. If self has available room, simply
insert value. If the heap is full, only insert value if it is better than the current root
(i.e., the minimum of self if self's predicate has < semantics). In that case, replace
the root of self and restore the heap property. This is useful to build and maintain a
heap with some top-N elements (relative to predicate) where the root (or minimum)
of self is the currently weakest element at the end of the list.

integer-length ((x LONG-INTEGER)) : INTEGER  [Function]
Return the 1-based position of the left-most bit in x. If x is negative, we only count
the value bits, not the sign.

integer-to-hex-string ((i LONG-INTEGER)) : STRING  [Function]
Convert i to a string representation in hexadecimal notation and return the result.
integer-to-string-in-base ((i LONG-INTEGER) (base INTEGER)) : [Function]
  STRING
  Convert i to a string representation in base and return the result. base must be positive and not more than 36.
  Note that in the C++ version, only 8, 10 and 16 will work as base arguments, since that is all the underlying implementation supports. Other argument values will be treated as 10.

integer-valued? ((x FLOAT)) : BOOLEAN [Function]
  Returns true if x is the floating point representation of an integer.

intern-stella-name ((name STRING)) : GENERALIZED-SYMBOL [Function]
  Parse name which is assumed to be the printed representation of a STELLA symbol, surrogate or keyword, intern it into the current or specified module and return the result. This is identical to calling unstringify on name but 10-15 times faster.

interpret-command-line-arguments ((count INTEGER) (arguments ARRAY () OF STRING)) : [Function]
  Interpret any STELLA-relevant command line arguments.

isa? ((object OBJECT) (type TYPE)) : BOOLEAN [Function]
  Return true iff object is an instance of the class named type.

java-translate-system ((systemName STRING)) : [Function]
  Translate the system systemName to Java.

jptrans ((statement OBJECT)) : [N-Command]
  Translate statement to C++ and print the result.

keyword-name? ((name STRING)) : BOOLEAN [Function]
  Return TRUE if name is prefixed by :.

kvlist-to-plist ((self KEY-VALUE-LIST)) : (PROPERTY-LIST OF (LIKE (ANY-KEY SELF)) (LIKE (ANY-VALUE SELF))) [Function]
  Convert self into a property list with identical and identically ordered keys and values.

last (self) : (LIKE (ANY-VALUE SELF)) [Method on VECTOR-SEQUENCE]
  Return the last item in the vector self.

length (self) : INTEGER [Method on CONS-ITERATOR]
  Iterate over self, and count how many items there are.

length (self) : INTEGER [Method on HEAP]
  Return the length of the currently filled portion of self.

length (self) : INTEGER [Method on DIMENSIONAL-ARRAY-MIXIN]
  Return the total number of elements in self.

length (self) : INTEGER [Method on 2-DIMENSIONAL-ARRAY-MIXIN]
  Return the total number of elements in self.
lispify ((thing UNKNOWN)) : LISP-CODE   [Function]
Convert a Stella thing as much as possible into a Common-Lisp analogue. The currently supported thing types are CONS, LIST, KEY-VALUE-LIST, ITERATOR, SYMBOL, KEYWORD, and all wrapped and unwrapped literal types. BOOLEANs are translated into Lisp's CL:T and CL:NIL logic. Unsupported types are left unchanged.

lispify-boolean ((thing UNKNOWN)) : LISP-CODE   [Function]
Lispify thing which is assumed to be a (possibly wrapped) Stella boolean.

list-modules ((kb-only? BOOLEAN)) : (CONS OF MODULE)   [Command]
Returns a cons of all modules defined in PowerLoom. If kb-only? is true, then any modules which are code only or just namespaces are not returned.

listify (self) : (LIST OF (LIKE (ANY-VALUE SELF)))   [Method on CONS]
Return a list of elements in self.

listify (self) : (LIST OF (LIKE (ANY-VALUE SELF)))   [Method on LIST]
Return self.

listify (self) : (LIST OF (LIKE (ANY-VALUE SELF)))   [Method on KEY-VALUE-LIST]
Return a list of key-value pairs in self.

listify (self) : (LIST OF (LIKE (ANY-VALUE SELF)))   [Method on VECTOR]
Return a list of elements in self.

listify (self) : (LIST OF (LIKE (ANY-VALUE SELF)))   [Method on ITERATOR]
Return a list of elements generated by self.

load-configuration-file ((file FILE-NAME)) : CONFIGURATION-TABLE   [Function]
Read a configuration file and return its content as a configuration table. Also enter each property read into the global system configuration table. Assumes Java-style property file syntax. Each property name is represented as a wrapped string and each value as a wrapped string/integer/float or boolean.

load-file ((file STRING)) :   [Command]
Read STELLA commands from file and evaluate them. The file should begin with an in-module declaration that specifies the module within which all remaining commands are to be evaluated The remaining commands are evaluated one-by-one, applying the function evaluate to each of them.

load-system ((systemName STRING) &rest (language&options object)) : BOOLEAN
Natively language-compile out-of-date translated files of system systemName (only supported for Lisp at the moment) and then load them into the running system. Return true if at least one file was compiled. The following keyword/value options are recognized:

:language: can be used as an alternative to the optional language argument. If not specified, the language of the running implementation is assumed.
:force-recompilation? (default false): if true, files will be compiled whether or not their compilations are up-to-date.

:startup? (default true): if true, the system startup function will be called once all files have been loaded.

**log-level</= ((level object) (module string)) : boolean**  
[Function]  
Return TRUE if level is lower than or equal to the current log level of module. Return FALSE if any of them are undefined.

**log-message ((module string) (logLevel object) (message cons)) :**  
[Function]  
Log all elements of message to modules log stream if logLevel is the same or lower than the modules log level. Interprets EOL or :EOL to print a line terminator.

**logmsg ((module string) (logLevel object) &rest (message object)) :**  
[Function]  
Log all elements of message to modules log stream if logLevel is the same or lower than the modules log level. Interprets EOL or :EOL to print a line terminator.

**lookup-class (name) : class**  
[Method on symbol]  
Return a class with name name. Scan all visible surrogates looking for one that has a class defined for it.

**lookup-class (name) : class**  
[Method on string]  
Return a class with name name. Scan all visible surrogates looking for one that has a class defined for it.

**lookup-command ((name symbol)) : method-slot**  
[Function]  
If name names an evaluable command return its associated command object; otherwise, return null. Currently, commands are not polymorphic, i.e., they can only be implemented by functions.

**lookup-configuration-property ((property string) (defaultValue wrapper) (configuration configuration-table)) : object**  
[Function]  
Look up property in configuration and return its value. Use the global system configuration table if configuration is NULL. Return defaultValue if property is not defined.

**lookup-configuration-property-values ((property string) (defaultValue object) (configuration configuration-table)) : cons**  
[Function]  
Look up property in configuration, assume it is a multi-valued property and return its value(s) as a list. Use the global system configuration table if configuration is NULL. Return defaultValue if property is not defined or NIL is no default value is specified.

**lookup-demon (name string) : demon**  
[Function]  
Return the demon named name.

**lookup-function ((functionSymbol symbol)) : function**  
[Function]  
Return the function defined for functionSymbol, if it exists.

**lookup-function-by-name (name string) : function**  
[Function]  
Return a function with name name visible from the current module. Scan all visible symbols looking for one that has a function defined for it.
lookup-global-variable (self) : GLOBAL-VARIABLE  [Method on SURROGATE]
Return a global variable with name self.

lookup-global-variable (self) : GLOBAL-VARIABLE  [Method on GENERALIZED-Symbol]
Return a global variable with name self.

lookup-global-variable (self) : GLOBAL-VARIABLE  [Method on STRING]
Return a global variable with name self.

lookup-local-slot ((class class) (slot-name symbol)) : SLOT  [Function]
Lookup a local slot with slot-name on class.

lookup-logging-parameter ((module string) (parameter keyword) (default object)) : OBJECT  [Function]
Lookup logging parameter for module. Use default if no value is defined.

lookup-macro ((name symbol)) : METHOD-SLOT  [Function]
If name has a macro definition, return the method object holding its expander function.

lookup-slot ((class class) (slot-name symbol)) : SLOT  [Function]
Return a slot owned by the class class with name slot-name. Multiply inherited slots are disambiguated by a left-to-right class precedence order for classes with multiple parents (similar to CLOS).

lookup-with-default (self (key (like (any-key self))) (default (like (any-value self)))) : (like (any-value self))  [Method on PROPERTY-LIST]
Lookup key in self and return the result. Return default if no value was found.

lookup-with-default (self (key (like (any-key self))) (default (like (any-value self)))) : (like (any-value self))  [Method on KEY-VALUE-LIST]
Lookup key in self and return the result. Return default if no value was found.

lptrans ((statement object))  [N-Command]
Translate statement to Common-Lisp and print the result.

make-matching-name ((original string) &rest (options object)) :  [Function]
STRING
Keyword options: :break-on-cap one of :YES :NO :CLEVER default is :CLEVER
:break-on-number one of :YES :NO :CLEVER default is :CLEVER
:break-on-separators string default is "-" :remove-prefix string :remove-suffix string
:case one of :UPCASE :TitleCase :titleCaseX :downcase :Capitalize :preserve default is :TitleCase
:separator string default is "" :add-prefix string :add-suffix string
MAKE-MATCHING-NAME returns a matching name (a string) for the input name (a string). A matching name is constructed by breaking the input into words and then applying appropriate transforms. The arguments are used as follows: original is the input name. It is a string. :break-on-cap is a keyword controlling whether
changes in capitalization is used to indicate word boundaries. If :YES, then all capitalization changes delineate words. If :CLEVER, then unbroken runs of capitalized letters are treated as acronyms and remain grouped. If :NO or NULL, there is no breaking of words based on capitalization. :break-on-number is a flag controlling whether encountering a number indicates a word boundary. If :YES, then each run of numbers is treated as a word separate from surrounding words. If :CLEVER, then an attempt is made to recognize ordinal numbers (ie, 101st) and treat them as separate words. If :NO or NULL, there is no breaking of words when numbers are encountered. :break-on-separators A string of characters which constitute word delimiters in the input word. This is used to determine how to break the name into individual words. Defaults are space, - and _. :remove-prefix Specifies a prefix or suffix that is stripped from the input :remove-suffix name before any other processing. This allows the removal of any naming convention dictated prefixes or suffixes. :add-prefix Specifies a prefix or suffix that is added to the output name :add-suffix after all other processing. This allows the addition of any naming convention dictated prefixes or suffixes. :case The case of the resulting name. This is applied to the name before adding prefixes or suffixes. The two title case options differ only in how the first word of the name is treated. :TitleCase capitalizes the first letter of the first word and also the first letter of all other words. :TitleCaseX does not capitalizes the first letter of the first word but capitalizes the first letter of all subsequent words. :preserve results in no change in case. :separator This is a string specifying the word separator to use in the returned name. An empty string (the default) means that the resulting words are concatenated without any separation. This normally only makes sense when using one of the title case values for the case keyword.

**make-matching-name-full** ((originalname STRING) 
  (breakoncap KEYWORD) (breakonnumber KEYWORD) 
  (breakonseparators STRING) (removeprefix STRING) (removesuffix STRING) 
  (addprefix STRING) (addsuffix STRING) (outputcase KEYWORD) 
  (outputseparator STRING) : STRING)

Non-keyword version of make-matching-name, which will probably be easier to use when called from non-Lisp languages.

**make-system** ((systemName STRING) 
  &rest (language&options OBJECT)) : BOOLEAN

Translate all out-of-date files of system systemName into language (the first optional argument of language&options) and then compile and load them (the latter is only possible for Lisp right now). The following keyword/value options are recognized:

:language: can be used as an alternative to the optional language argument. If not specified, the language of the running implementation is assumed.

two-pass?: if true, all files will be scanned twice, once to load the signatures of objects defined in them, and once to actually translate the definitions. Otherwise, the translator will make one pass in the case that the system is already loaded (and is being remade), and two passes otherwise.

development-settings? (default false): if true translation will favor safe, readable and debuggable code over efficiency (according to the value of :development-
settings on the system definition). If false, efficiency will be favored instead (according to the value of :production-settings on the system definition).

:production-settings? (default true): inverse to :development-settings?.

:force-translation? (default false): if true, files will be translated whether or not their translations are up-to-date.

:force-recompilation? (default false): if true, translated files will be recompiled whether or not their compilations are up-to-date (only supported in Lisp right now).

:load-system? (default true): if true, compiled files will be loaded into the current STELLA image (only supported in Lisp and Java right now).

:startup? (default true): if true, the system startup function will be called once all files have been loaded.

**max** \((x \ (y \ \text{integer})) \) : \text{integer} 
[Method on \text{integer}]
Return the maximum of \(x\) and \(y\). If either is NULL, return the other.

**max** \((x \ (y \ \text{long-integer})) \) : \text{long-integer} 
[Method on \text{long-integer}]
Return the maximum of \(x\) and \(y\). If either is NULL, return the other.

**max** \((x \ (y \ \text{float})) \) : \text{float} 
[Method on \text{float}]
Return the maximum of \(x\) and \(y\). If either is NULL, return the other.

**max** \((x \ (y \ \text{number-wrapper})) \) : \text{number-wrapper} 
[Method on \text{number-wrapper}]
Return the maximum of \(x\) and \(y\). If \(y\) is NULL, return \(x\).

**member?** \((\text{self} \ (\text{value} \ \text{object})) \) : \text{boolean} 
[Method on \text{cons-iterator}]
Iterate over values of \(\text{self}\) and return TRUE if one of them is eql? to `value.

**member?** \((\text{self} \ (\text{object} \ \text{object})) \) : \text{boolean} 
[Method on \text{collection}]
Return true iff \(\text{object}\) is a member of the collection \(\text{self}\).

**member?** \((\text{self} \ (\text{value} \ \text{object})) \) : \text{boolean} 
[Method on \text{sequence}]
Return TRUE if \(\text{value}\) is a member of the sequence \(\text{self}\).

**memoize** \((\text{inputArgs cons} \ &\text{body} \ (\text{body} \ \text{cons})) \) : \text{object} 
[Macro]
Compute the value of an expression and memoize it relative to the values of \text{inputArgs}. \text{inputArgs} should characterize the complete set of values upon which the computation of the result depended. Calls to \text{memoize} should be of the form
\(\text{memoize} \ (<\text{arg}>+) \ {:<\text{option}> \ <\text{value}>}* \ <\text{expression}>\)
and have the status of an expression. The following options are supported:

:timestamps A single or list of keywords specifying the names of timestamps which when bumped should invalidate all entries currently memoized in this table. :name Names the memoization table so it can be shared by other memoization sites. By default, a gensymmed name is used. CAUTION: IT IS ASSUMED THAT ALL ENTRIES IN A MEMOIZATION TABLE DEPEND ON THE SAME NUMBER OF ARGUMENTS!! :max-values The maximum number of values to be memoized. Only the :max-values most recently used values will be kept in the memoization table,
older values will be discarded and recomputed if needed. Without a :max-values specification, the memoization table will grow indefinitely.

PERFORMANCE NOTES: For most efficient lookup, input arguments that vary the most should be listed first. Also, arguments of type STANDARD-OBJECT (and all its subtypes) can be memoized more efficiently than arguments of type OBJECT or wrapped literals (with the exception of BOOLEANs).

**merge-file-names** (((baseFile FILE-NAME) (defaults FILE-NAME)) : [Function]
FILE-NAME
Parse baseFile, supply any missing components from defaults if supplied and return the result.

**merge-null-fields** (self (default DECODED-DATE-TIME)) :
[Method on DECODED-DATE-TIME]
Replace any null valued fields in self with values from default. The day of the week will be set consistently, if possible.

**merge-superior-null-fields** (self (default DECODED-DATE-TIME)) :
Replace only null valued fields in self that represent larger time units than the smallest non-null in self with values from default. The day of the week will be set consistently, if possible. Example: if self just has the month being non-null, then only the year will be filled in from default. If the day and minute were non-null, then hour, month and year will be filled.

This can be useful when one doesn’t want to extend the precision of the answer.

**min** (x (y INTEGER)) : INTEGER
[Method on INTEGER]
Return the minimum of x and y. If either is NULL, return the other.

**min** (x (y LONG-INTEGER)) : LONG-INTEGER
[Method on LONG-INTEGER]
Return the minimum of x and y. If either is NULL, return the other.

**min** (x (y FLOAT)) : FLOAT
[Method on FLOAT]
Return the minimum of x and y. If either is NULL, return the other.

**min** (x (y NUMBER.WRAPPER)) : NUMBER.WRAPPER
[Method on NUMBER.WRAPPER]
Return the minimum of x and y. If y is NULL, return x.

**mod** (x (modulus INTEGER)) : INTEGER
[Method on INTEGER]
True modulus. Return the result of x mod modulo. Note: In C++ and Java, mod has more overhead than the similar function rem. The answers returned by mod and rem are only different when the signs of x and modulo are different.

**mod** (x (modulus LONG-INTEGER)) : LONG-INTEGER
[Method on LONG-INTEGER]
True modulus. Return the result of x mod modulo. Note: In C++ and Java, mod has more overhead than the similar function rem. The answers returned by mod and rem are only different when the signs of x and modulo are different.

**multiple-parents?** (class) : BOOLEAN
[Method on CLASS]
Return true if class has more than one direct superclass.
multiple-parents? (module) : BOOLEAN
   [Method on MODULE]
   Return TRUE if module has more than one parent.

multiple-parents? (world) : BOOLEAN
   [Method on WORLD]
   Return FALSE always, since worlds never have more than one parent.

name-to-string ((name object)) : STRING
   [Function]
   Return the string represented by name. Return null if name is undefined or does not represent a string.

native-read-line ((inputStream input-stream)) : STRING
   [Function]
   Read one line from inputStream using the native language readline algorithm and return the result. On EOF return null

next? (self) : BOOLEAN
   [Method on memoizable-iterator]
   Generate the next value of the memoized iterator self (or one of its clones) by either using one of the values generated so far or by generating and saving the next value of the base-iterator.

no-duplicates? (self) : BOOLEAN
   [Method on collection]
   Return true if the collection self forbids duplicate values.

non-empty? (x) : BOOLEAN
   [Method on string-wrapper]
   Return true if x is not the wrapped empty string ""

non-empty? (self) : BOOLEAN
   [Method on vector-sequence]
   Return true if self has length > 0.

non-matching-position ((source string) (start integer) (match string)) : integer
   [Function]
   Returns the index into source, starting from start, of the first character that is not included in match.

non-matching-position-helper ((source string) (start integer) (end integer) (match string)) : integer
   Helper for non-matching-position that requires end to not be null.

nth (self (position integer)) : (like (any-value self))
   [Method on native-vector]
   Return the element in self at position.

null? (x) : BOOLEAN
   [Method on long-integer]
   Return true if x is undefined (handled specially by all translators).

null? (x) : BOOLEAN
   [Method on array]
   Return true if x is undefined (handled specially by all translators).

number-less-than? ((x number-wrapper) (y number-wrapper)) : boolean
   [Function]
   Generic number comparison that works with integers, longs and floats.
**object-equal?** (x (y object)) : BOOLEAN  
[Method on VECTOR-SEQUENCE]
Return TRUE iff the sequences x and y are structurally equivalent. Uses **equal?** to test equality of elements.

**odd?** (x) : BOOLEAN  
[Method on INTEGER]
Return true if x is an odd number.

**odd?** (x) : BOOLEAN  
[Method on LONG-INTEGER]
Return true if x is an odd number.

**only-if** ((test object) (expression object)) : OBJECT  
[Macro]
If test is TRUE, return the result of evaluating expression.

**open-network-stream** ((host string) (port integer)) :
[Function]
INPUT-STREAM OUTPUT-STREAM
Open a TCP/IP network stream to host at port and return the result as an input/output stream pair.

**ordered?** (self) : BOOLEAN  
[Method on COLLECTION]
Return true if the collection self is ordered.

**outline-depth-exceeded?** ((current-depth integer) (depth-limit integer)) : BOOLEAN  
[Function]
Helper function that returns true if current-depth exceeds depth-limit. This functions uses the convention that a null or negative value of depth-limit means the depth is unlimited. In those cases it always returns false.

**parameters** (self) : (LIST OF SYMBOL)  
[Method on CLASS]
Returns the list of parameters names of self.

[Function]
Tries very hard to make sense out of the argument date-time-string and returns a time structure if successful. If not, it returns null. If error-on-mismatch? is true, parse-date-time will signal an error instead of returning null. Default values are 00:00:00 in the given timezone on the current date. If the given time-zone value is null, then the local time zone for the parsed date/time will be used as determined by the operating system.

**parse-date-time-relative-to-base** ((date-time-string string) (base-date-time DECODED-DATE-TIME) (start integer) (end integer) (error-on-mismatch? boolean) (merge-null-fields? boolean)) :
DECODED-DATE-TIME  
[Function]
Tries very hard to make sense out of the argument date-time-string and returns a time structure if successful. If not, it returns null. If error-on-mismatch? is true, parse-date-time will signal an error instead of returning null. Default values are passed in via base-date-time. If the timezone field that is passed in is NULL, then the local time zone for the parsed date/time will be used. If merge-null-fields? is true, then default values from base-time-date will be merged into missing components.
If false, then they won’t be merged in for null components but can still be used as
a basis for interpretation of relative time strings like "now" or "yesterday"

**parse-stella-name ((name string) (enableCaseConversion? boolean))**  
*Function*

.Parse the printed representation name of a STELLA symbol, surrogate or keyword
and return its symbol name, module name and type (which is either :SYMBOL,
:SURROGATE or :KEYWORD). name can be qualified and must use the exact same
syntax and escape characters that would be used if it were to be read by read-s-
expression-from-string (or unstringify). If enableCaseConversion? is TRUE,
the returned symbol name will be upcased if the current module is case-insensitive;
otherwise, it will be returned as is. Raises a read exception if name does not represent
a symbol. This function is available primarily for efficiency, since it is about 10-15
times faster than unstringify.

**pick-hash-table-size-prime ((minSize integer)) : integer**  
*Function*

.Return a hash table prime of at least minSize.

**plist-to-kvlist ((self property-list)) : (key-value-list of (like
(ANY-KEY SELF)) (like (ANY-VALUE SELF)))**  
*Function*

.Convert self into a key-value list with identical and identically ordered keys and values.

**plus? (x) : boolean**  
*Method on integer*

.Return true if x is greater than 0.

**plus? (x) : boolean**  
*Method on long-integer*

.Return true if x is greater than 0.

**primary-type (self) : type**  
*Method on object*

.Returns the primary type of self. Gets defined automatically for every non-abstract
subclass of OBJECT.

**primitive? (self) : boolean**  
*Method on relation*

.Return true if self is not a defined relation.

**print (&body (body cons)) : object**  
*Macro*

.Print arguments to the standard output stream.

**print-exception-context ((e native-exception) (stream output-stream)) :**  
*Function*

.Prints a system dependent information about the context of the specified exception.
For example, in Java it prints a stack trace. In Lisp, it is vendor dependent.

**print-outline ((thing object) (stream output-stream) (depth integer) (named? boolean)) :**  
*Command*

.Print an outline of thing and its subparts on stream. If depth is non-negative, only
depth levels will be printed. If named? is TRUE, then only named entities will be
printed.

This function is intended to be used on things like modules, contexts, concepts, etc.
that have hierarchical structure. If thing doesn’t have a hierarchical structure, it will
just be printed.
print-recycle-lists () : [Function]
Print the current state of all recycle lists.

print-spaces (&body (body cons)) : OBJECT [Macro]
(print-spaces [stream] N) prints N spaces onto stream. If no stream form is provided, then STANDARD-OUTPUT will be used.

print-stella-features () : [Command]
Print the list of enabled and disabled STELLA features.

print-unbound-surrogates (&rest (args object)) : [N-Command]
Print all unbound surrogates visible from the module named by the first argument (a symbol or string). Look at all modules if no module name or null was supplied. If the second argument is true, only consider surrogates interned in the specified module.

print-undefined-methods ((module module) (local? boolean)) : [Function]
Print all declared but not yet defined functions and methods in module. If local? is true, do not consider any parent modules of module. If module is NULL, look at all modules in the system. This is handy to pinpoint forward declarations that haven’t been followed up by actual definitions.

print-undefined-super-classes ((class name)) : [N-Command]
Print all undefined or bad (indirect) super classes of class.

private-class-methods ((class class)) : (iterator of method-slot) [Function]
Iterate over all private methods attached to class.

private-class-storage-slots ((class class)) : (iterator of storage-slot)
Iterate over all private storage-slots attached to class.

private? (self) : BOOLEAN [Method on RELATION]
Return true if self is not public.

process-doctype ((doctype-declaration cons)) : XML-DOTYPE [Function]
Takes an S-Expression representing a doctype and processes into a DOCTYPE object.

ptrans ((statement object)) :
Translate statement to Common-Lisp and print the result.

public-class-methods ((class class)) : (iterator of method-slot) [Function]
Iterate over all public methods attached to class.

public-class-storage-slots ((class class)) : (iterator of storage-slot)
Iterate over all public storage-slots attached to class.

public-slots (self) : (iterator of slot) [Method on CLASS]
Return an iterator over public slots of self.

public-slots (self) : (iterator of slot) [Method on OBJECT]
Return an iterator over public slots of self.
public? (self) : BOOLEAN  
True if self or one of its ancestors is marked public.

pushf ((place CONS) (value OBJECT)) : OBJECT  
Push value onto the cons list place.

qualified-stella-name? ((name STRING)) : BOOLEAN  
Return TRUE if name is a symbol or surrogate qualified with a module pathname or a module pathname ending with a /.
Assumes that name is the printed representation of a STELLA symbol (potentially containing escape characters).

read-line2 ((stream INPUT-STREAM)) : STRING KEYWORD  
Read one line from stream and return the result and a keyword that indicates the terminator for that line ending: :CR :LF :CRLF or :EOF. This is not platform-dependent and differs from read-line by returning a second value. It may hang when used on interactive streams such as terminal or network streams with only CR line endings. It should only be used on file or string input streams.

reader (self) : SYMBOL  
Name of a method called to read the value of the slot self.

regular-integer-valued? ((x LONG-INTEGER)) : BOOLEAN  
Return true if x can be represented by a regular integer.

rem (x (y INTEGER)) : INTEGER  
Return the remainder from dividing x by y. The sign of the result is always the same as the sign of x. This has slightly different behavior than the mod function, and has less overhead in C++ and Java, which don’t have direct support for a true modulus function.

rem (x (y LONG-INTEGER)) : LONG-INTEGER  
Return the remainder from dividing x by y. The sign of the result is always the same as the sign of x. This has slightly different behavior than the mod function, and has less overhead in C++ and Java, which don’t have direct support for a true modulus function.

remove-configuration-property ((property STRING) (value OBJECT) (configuration CONFIGURATION-TABLE)) : OBJECT  
Remove value from property in configuration and return it. Use the global system configuration table if configuration is NULL.

remove-duplicates (self) : (LIKE SELF)  
Return self with duplicates removed. Preserves the original order of the remaining members.

replace-heap-root (self (value (LIKE (ANY-VALUE SELF)))) :  
Replace the current root of self with value and restore the heap property. Signal an error if self is empty. Maintains self as a Min-heap if self’s predicate has < semantics; otherwise as a Max-heap.
required-slots \((self)\) : \((\text{list of symbol})\)  
Returns a list of names of required slots for \(self\).

required? \((self)\) : \(\text{boolean}\)  
True if a value must be assigned to this slot at creation time.

reset-stella-features () :  
Reset STELLA features to their default settings.

reverse-interval \((\text{lowerbound integer}) (\text{upperbound integer})\) : \(\text{reverse-integer-interval-iterator}\)  
Create a reverse interval object.

run-hooks \((\text{hooklist hook-list}) (\text{argument object})\) :  
Run all hook functions in \(\text{hooklist}\), applying each one to \(\text{argument}\).

running-as-lisp? () : \(\text{boolean}\)  
Return true if the executable code is a Common Lisp application.

running-in-language () : \(\text{keyword}\)  
Returns the keyword for the language the current implementation is running in.

running-system-information () : \(\text{string}\)  
Returns an information string about the current running system environment.

safe-equal-hash-code \((\text{self object})\) : \(\text{integer}\)  
Return a hash code for \(\text{self}\). Just like \text{equal-hash-code} - which see, but also works for NULL. \text{equal-hash-code} methods that expect to handle NULL components should use this function for recursive calls.

safe-hash-code \((\text{self object})\) : \(\text{integer}\)  
Return a hash code for \(\text{self}\). Just like \text{hash-code} - which see, but also works for NULL.

safe-lookup-slot \((\text{class class}) (\text{slot-name symbol})\) : \(\text{slot}\)  
Alias for \text{lookup-slot}. Kept for backwards compatibility.

safety \((\text{level integer-wraper}) (\text{test object}) \&\text{body} (\text{body cons})\) : \(\text{macro}\)  
Signal warning message, placing non-string arguments in quotes.

save-configuration-file \((\text{table configuration-table})\) \((\text{file file-name}) (\text{title string})\) :  
Save \(\text{table}\) as a configuration file. Uses a Java-style property file syntax.

save-configuration-value \((\text{stream output-stream}) (\text{value object})\) :  
Save \(\text{value}\) to \(\text{stream}\) as a properly formatted configuration value.

search-cons-tree-with-filter? \((\text{tree object}) (\text{value object})\) \((\text{filter cons})\) : \(\text{boolean}\)  
Return \(\text{true}\) iff the value \(\text{value}\) is embedded within the cons tree \(\text{tree}\). Uses an \text{eql}\? test. Does not descend into any cons whose first element matches an element of \(\text{filter}\).
search-for-object ((self object) (typer ef object)) : object
[Function]
If self is a string or a symbol, search for an object named self of type type. Otherwise, if self is an object, return it.

seed-random-number-generator () :
[Function]
Seeds the random number generator with the current time.

sequence ((collectiontype type) &rest (values object)) : (sequence of object)
[Function]
Return a sequence containing values, in order.

set-call-log-break-point ((count integer)) :
[Command]
Set a call log break point to count. Execution will be interrupted right at the entry of the countth logged function call.

set-configuration-property ((property string) (value object) (configuration configuration-table)) : object
[Function]
Set property in configuration to value and return it. Use the global system configuration table if configuration is NULL.

set-current-date (values-structure) :
[Method on DECoded-DATE-TIME]
Sets the current date into values-structure

set-current-date-time (values-structure) :
[Method on DECoded-DATE-TIME]
Sets the current date and time into values-structure

set-current-time (values-structure) :
[Method on DECoded-DATE-TIME]
Sets the current time into values-structure

set-global-value ((self surrogate) (value object)) : object
[Function]
Set the value of the global variable for the surrogate self to value.

set-log-level ((module string) (level object)) :
[Command]
Set the log-level for module to level. This is a convenience function for this common operation.

set-logging-parameters ((module string) &rest (params&values object)) :
[Command]
Set logging parameters for module. The supported parameters are: :LOG-LEVELS - a cons list of legal levels in ascending log level order; for example, (:NONE :LOW :MEDIUM :HIGH) or (0 1 2 3). :LEVEL - the current log level for module :STREAM - the stream to log to (defaults to STANDARD-OUTPUT) :PREFIX - the prefix to use to identify the module (defaults to module) :MAX-WIDTH - logging output lines will be kept to approximately this width (defaults to 10000, minimum width of about 30 is used to print line header information).

set-optimization-levels ((safety integer) (debug integer) (speed integer) (space integer)) :
[Function]
Set optimization levels for the qualities safety, debug, speed, and space.

set-stella-feature (&rest (features keyword)) :
[Command]
Enable all listed STELLA features.
**set-translator-output-language** \((\text{new-language \underline{keyword}})\) : [Command]

Set output language to new-language. Return previous language.

**setq?** \((\text{variable \underline{symbol}}) (\text{expression \underline{cons}})\) : \text{OBJECT} [Macro]

Assign variable the result of evaluating expression, and return TRUE if expression is not NULL else return FALSE.

**shadowed-symbol?** \((\text{symbol \underline{generalized-symbol}})\) : \text{BOOLEAN} [Function]

Return true if symbol is shadowed in its home module.

**shift-right** \((\text{arg \underline{count \underline{integer}}})\) : \text{INTEGER} [Method on \underline{integer}]

Shift arg to the right by count positions and 0-extend from the left if arg is positive or 1-extend if it is negative. This is an arithmetic shift that preserve the sign of arg and is equivalent to dividing arg by \(2^{\ast \ast \text{count}}\).

**shift-right** \((\text{arg \underline{count \underline{integer}}})\) : \text{LONG-INTEGER} [Method on \underline{long-integer}]

Shift arg to the right by count positions and 0-extend from the left if arg is positive or 1-extend if it is negative. This is an arithmetic shift that preserve the sign of arg and is equivalent to dividing arg by \(2^{\ast \ast \text{count}}\).

**signal** \((\text{\underline{type \underline{symbol}}} \&\text{body (\underline{body \underline{cons}})})\) : \text{OBJECT} [Macro]

Signal error message, placing non-string arguments in quotes.

**signal-read-error** \((\&\text{body (\underline{body \underline{cons}})})\) : \text{OBJECT} [Macro]

Specialized version of signal that throws a READ-EXCEPTION.

**sort** \((\text{self \underline{predicate \underline{function-code}}})\) : \text{(VECTOR \underline{of \underline{like \underline{any-value self}}})} [Method on \underline{vector}]

Perform a destructive sort of self according to predicate, and return the result. If predicate has a \(<\) semantics, the result will be in ascending order. If predicate is null, a suitable \(<\) predicate is chosen depending on the first element of self, and it is assumed that all elements of self have the same type (supported element types are \text{GENERALIZED-SYMBOL}, \text{STRING}, \text{INTEGER}, and \text{FLOAT}).

**sort** \((\text{self \underline{predicate \underline{function-code}}})\) : \text{(HEAP \underline{of \underline{like \underline{any-value self}}})} [Method on \underline{heap}]

Sort the heap self according to predicate (in ascending order if predicate has \(<\) semantics). If predicate is NULL simply use self’s internal predicate (the normal case). If it is different from self’s internal predicate, heapify self first according to the new predicate, store the new predicate in self and then sort the heap. Note that a sorted array automatically satisfies the heap property. This is slightly different than a regular heap sort due to the way HEAP’s are maintained; however, the complexity is the same.

**split-string** \((\text{\underline{input \underline{string}}} \underline{separator \underline{character}})\) : \text{(CONS \underline{of \underline{string-wrapper}})} [Function]

Split input into separate strings based on the separator character.

**start-function-call-logging** \((\text{\underline{file-name \underline{string}}})\) : [Command]

Start function call logging to file-name.
**starts-with** ((string string) (prefix string) (start integer)) : BOOLEAN  
Return TRUE if string starts with prefix starting from start (which defaults to 0 if it is supplied as NULL).

**stella-collection?** ((self object)) : BOOLEAN  
Return true if self is a native collection.

**stella-information** () : STRING  
Returns information about the current Stella implementation. Useful when reporting problems.

**stella-object?** ((self object)) : BOOLEAN  
Return true if self is a member of the STELLA class OBJECT.

**stella-version-string** () : STRING  
Return a string identifying the current version of STELLA.

**stellafy** ((thing lisp-code) (targetType type)) : OBJECT  
Partial inverse to lispify. Convert the Lisp object thing into a Stella analogue of type targetType. Note: See also stellify. it is similar, but guesses targetType on its own, and makes somewhat different translations.

**stellify** ((self object)) : OBJECT  
Convert a Lisp object into a STELLA object.

**stop-function-call-logging** () :  
Stop function call logging and close the current log file.

**stream-position** (self) : LONG-INTEGER  
Return the current position of the file input cursor in self.

**stream-position** (self) : LONG-INTEGER  
Return the current position of the file input cursor in self.

**stream-position-setter** (self (newpos long-integer)) : LONG-INTEGER  
Set the current position of the file input cursor in self to newpos.

**stream-position-setter** (self (newpos long-integer)) : LONG-INTEGER  
Set the current position of the file input cursor in self to newpos. If self has any tokenizer state associated with it, this will also reset to the start state of the tokenizer table; otherwise, behavior would be unpredictable unless the character class of the new position is exactly the same as the one following the most recent token.

**stream-to-string** (from) : STRING  
Read all of the input from stream and return it as a string.
string-to-calendar-date-with-default ((input-date STRING) (default-date DECODED-DATE-TIME)) : CALENDAR-DATE

Returns a calendar date object representing the date and time parsed from the input-date string. Default values for missing fields and the interpretation of relative references come from default-date. If the default-date is null, the current date will be used. If a null set of defaults is desired, use *NULL-DECODED-DATE-TIME*. If no valid parse is found, null is returned.

string-to-decoded-date-time ((input STRING)) :

DECODED-DATE-TIME

Returns a decoded date-time object representing the date and time parsed from the input string. If no valid parse is found, null is returned.

string-to-surrogate ((self STRING)) : SURROGATE

Returns a surrogate with the name self visible in the current module. Very tricky: The logic is designed to avoid returning an inherited surrogate that has no value. In that case, a new local surrogate is created that shadows the inherited surrogate.

string-to-time-duration ((duration STRING)) : TIME-DURATION

Parses and returns a time-duration object corresponding to duration. The syntax for time duration strings is "{plus|minus} N days[; M ms]" where N and M are integer values for days and milliseconds. If no valid parse is found, null is returned.

subclass-of? ((subClass CLASS) (superClass CLASS)) : BOOLEAN

Returns true if subClass is a subclass of superClass.

subsequence (string (start INTEGER) (end INTEGER)) : STRING

Returns a substring of string beginning at position start and ending up to but not including position end, counting from zero. An end value of NULL stands for the rest of the string.

substitute-characters (self (new-chars STRING) (old-chars STRING)) : STRING

Substitute all occurrences of a member of old-chars with the corresponding member of new-chars in the string self. Returns a new string.

substitute-characters (self (new-chars STRING) (old-chars STRING)) : MUTABLE-STRING

Substitute all occurrences of a member of old-chars with the corresponding member of new-chars in the string self. IMPORTANT: The return value should be used instead of relying on destructive substitution, since the substitution will not be destructive in all translated languages.

subtype-of? ((sub-type TYPE) (super-type TYPE)) : BOOLEAN

Returns true iff the class named sub-type is a subclass of the class named super-type.

super-classes (self) : (ITERATOR OF CLASS)

Returns an iterator that generates all super classes of self. Non-reflexive.
surrogate-name? ((name string)) : BOOLEAN
  Return TRUE if name is prefixed by SURROGATE-PREFIX-CHARACTER.

surrogatify (self) : SURROGATE
  Converts self into a surrogate.

surrogatify (self) : SURROGATE
  Converts self into a surrogate (same semantics as symbol-to-surrogate which see).

surrogatify (self) : SURROGATE
  Converts self into a surrogate.

sweep (self) :
  Default method. Sweep up all self-type objects.

symbol-to-surrogate ((self symbol)) : SURROGATE
  Return a surrogate with the same name as self. Very tricky: The logic is designed
to avoid returning an inherited surrogate that has no value. In that case, a new
local surrogate is created that shadows the inherited surrogate. Unlike string-to-
surrogate, the search starts first from the home context of self, and if that fails, then
it restarts in *module*.

symbol-to-type ((self symbol)) : SURROGATE
  Convert self into a surrogate with the same name and module.

system-default-value (self) : OBJECT
  Return a default value expression, or if self has dynamic storage, an initial value
expression.

system-default-value (self) : OBJECT
  Return a default value expression, or if self has dynamic storage, an initial value
expression.

system-loaded-or-started-up? ((name string)) : BOOLEAN
  Return true if system name has either been loaded or initialized with its startup
function.

system-loaded? ((name string)) : BOOLEAN
  Return true if system name has been loaded.

terminate-program () :
  Terminate and exit the program with normal exit code.

time-duration-to-string (date) : STRING
  Returns a string representation of date

time-zone-format60 ((timezone float)) : STRING
  Format zone as an hh:mm string

toggle-output-language () : KEYWORD
  Switch between Common Lisp and C++ as output languages.
**tokenize-string** ((string STRING) (punctuationchars STRING) (quotechars STRING) (escapechars STRING)) : (CONS OF CONS)

Simple tokenizer that is somewhere between Java’s StringTokenizer and StreamTokenizer in functionality. It doesn’t specially support number tokens nor comment strings/sequences even though this could be added at the expense of some extra complexity. Returns a list of (<token-string> <token-type>) pairs, where the token type is one of :TEXT, :PUNCTUATION or :QUOTE, i.e., all white space is ignored and escape characters are handled and removed. For example:

```
(tokenize-string "for(i='fo"o'; i>0; i++)" "(=<>+-; "') ^")
=>
(("for" :TEXT) ("(" :PUNCTUATION) ("i" :TEXT)
 (="" :PUNCTUATION) ("""" :QUOTE) ("fo'o" :TEXT)
 ("="" :QUOTE) ("" :PUNCTUATION) ("i" :TEXT)
 ("">" :PUNCTUATION) ("0" :TEXT) (";" :PUNCTUATION)
 ("i" :TEXT) ("++" :PUNCTUATION))
```

NOTE: this aggregates multiple punctuation characters that immediately follow each other into a single token which is (generally) useful to pickup multi-character operators such as ++, >=, etc. It’s still easy to pick them apart in a post-processing step if necessary (e.g., for the ++) case above), so we leave this for now as a feature.

**trace-if** ((keyword OBJECT) &body (body CONS)) : OBJECT

If keyword is a trace keyword that has been enabled with add-trace print all the elements in body to standard output. Otherwise, do nothing. keyword can also be a list of keywords in which case printing is done if one or more of them are trace enabled.

**translate-system** ((systemName STRING) &rest (language&options OBJECT)) : BOOLEAN

Translate all of the STELLA source files in system systemName into language (the optional first argument). The following keyword/value options are recognized:

:language: can be used as an alternative to the optional language argument. If not specified, the language of the running implementation is assumed.

:two-pass? (default false): if true, all files will be scanned twice, once to load the signatures of objects defined in them, and once to actually translate the definitions.

:force-translation? (default false): if true, files will be translated whether or not their translations are up-to-date.

:development-settings? (default false): if true translation will favor safe, readable and debuggable code over efficiency (according to the value of :development-settings on the system definition). If false, efficiency will be favored instead (according to the value of :production-settings on the system definition).

:production-settings? (default true): inverse to :development-settings?.

**translate-to-common-lisp?** () : BOOLEAN

Return true if current output language is Common-Lisp.

**translate-to-cpp?** () : BOOLEAN

Return true if current output language is C++
translate-to-java? () : BOOLEAN
   Return true if current output language is Java

truncate ((n NUMBER)) : INTEGER
   Truncate n toward zero and return the result.

try-to-evaluate ((tree OBJECT)) : OBJECT
   Variant of evaluate that only evaluates tree if it represents an evaluable expression. If it does not, tree is returned unmodified. This can be used to implement commands with mixed argument evaluation strategies.

two-argument-least-common-superclass ((class1 CLASS) (class2 CLASS)) : CLASS
   Return the most specific class that is a superclass of both class1 and class2. If there is more than one, arbitrarily pick one. If there is none, return null.

two-argument-least-common-supertype ((type1 TYPE-SPEC) (type2 TYPE-SPEC)) : TYPE-SPEC
   Return the most specific type that is a supertype of both type1 and type2. If there is more than one, arbitrarily pick one. If there is none, return @VOID. If one or both types are parametric, also try to generalize parameter types if necessary.

type (self) : TYPE
   The type of a storage slot is its base type.

type-specifier (self) : TYPE-SPEC
   If self has a complex type return its type specifier, otherwise, return type of self.

type-to-symbol ((type TYPE)) : SYMBOL
   Convert type into a symbol with the same name and module.

type-to-wrapped-type (self) : TYPE
   Return the wrapped type for the type self, or self if it is not a bare literal type.

unbound-surrogates ((module MODULE) (local? BOOLEAN)) :
   Iterate over all unbound surrogates visible from module. Look at all modules if module is null. If local?, only consider surrogates interned in module.

unbump-log-indent () :
   Decrease the indentation level for subsequent log messages.

unescape-html-string ((input STRING)) : STRING
   Replaces HTML escape sequences such as &amp; with their associated characters.

unescape-url-string ((input STRING)) : STRING
   Takes a string and replaces %-format URL escape sequences with their real character equivalent according to RFC 2396.

unset-stella-feature (&rest (features KEYWORD)) :
   Disable all listed STELLA features.
**unsigned-shift-right-by-1** (arg) : INTEGER [Method on INTEGER]
Shift arg to the right by 1 position and 0-extend from the left. This does not preserve the sign of arg and shifts the sign-bit just like a regular bit. In Common-Lisp we can’t do that directly and need to do some extra masking.

**unsigned-shift-right-by-1** (arg) : LONG-INTEGER [Method on LONG-INTEGER]
Shift arg to the right by 1 position and 0-extend from the left. This does not preserve the sign of arg and shifts the sign-bit just like a regular bit. In Common-Lisp we can’t do that directly and need to do some extra masking.

**unstringify-stella-source** ((source STRING) (module MODULE)) : OBJECT [Function]
Unstringify a STELLA source string relative to module, or *MODULE* if no module is specified. This function allocates transient objects as opposed to unstringify-in-module or the regular unstringify.

**unwrap-boolean** ((wrapper BOOLEAN-WRAPPER)) : BOOLEAN [Function]
Unwrap wrapper and return its values as a regular BOOLEAN. Map NULL onto FALSE.

**unwrap-function-code** ((wrapper FUNCTION-CODE.WRAPPER)) : FUNCTION-CODE [Function]
Unwrap wrapper and return the result. Return NULL if wrapper is NULL.

**unwrap-long-integer** ((wrapper LONG-INTEGER-WRAPPER)) : LONG-INTEGER [Function]
Unwrap wrapper and return the result. Return NULL if wrapper is NULL.

**unwrap-method-code** ((wrapper METHOD-CODE.WRAPPER)) : METHOD-CODE [Function]
Unwrap wrapper and return the result. Return NULL if wrapper is NULL.

**value-setter** (self (value (LIKE (ANY-VALUE SELF)))) : (LIKE (ANY-VALUE SELF)) [Method on ABSTRACT-DICTIONARY-ITERATOR]
Abstract method needed to allow application of this method on abstract iterator classes that do not implement it. By having this here all next? methods of dictionary iterators MUST use the slot-value paradigm to set the iterator value.

**warn** (&body (body CONS)) : OBJECT [Macro]
Signal warning message, placing non-string arguments in quotes.

**with-network-stream** ((binding CONS) &body (body CONS)) : OBJECT [Macro]
Sets up an unwind-protected form which opens a network socket stream to a host and port for input and output and closes it afterwards. Separate variables as provided in the call are bound to the input and output streams. Syntax is (WITH-NETWORK-STREAM (varIn varOut hostname port) body+)

**with-permanent-objects** (&body (body CONS)) : OBJECT [Macro]
Allocate permanent (as opposed to transient) objects within the scope of this declaration.
with-system-definition ((systemnameexpression object) [Macro]
  &body (body cons)) : OBJECT
Set *currentSystemDefinition* to the system definition named system. Set *currentSystemDefinitionSubdirectory* to match. Execute body within that scope.

with-transient-objects (&body (body cons)) : OBJECT [Macro]
Allocate transient (as opposed to permanent) objects within the scope of this declaration. CAUTION: The default assumption is the allocation of permanent objects. The scope of with-transient-objects should be as small as possible, and the user has to make sure that code that wasn’t explicitly written to account for transient objects will continue to work correctly.

wrap-boolean ((value boolean)) : BOOLEAN-WRAPPER [Function]
Return a literal object whose value is the BOOLEAN value.

wrap-function-code ((value function-code)) :
  FUNCTION-CODE-WRAPPER [Function]
Return a literal object whose value is the FUNCTION-CODE value.

wrap-integer-value ((value long-integer)) : NUMBER-WRAPPER [Function]
Return a literal object whose value is value. Choose a regular integer wrapper unless value is too large and needs to be stored in a long wrapper.

wrap-long-integer ((value long-integer)) : LONG-INTEGER-WRAPPER [Function]
Return a literal object whose value is the LONG-INTEGER value.

wrap-method-code ((value method-code)) :
  METHOD-CODE-WRAPPER [Function]
Return a literal object whose value is the METHOD-CODE value.

wrapped-type-to-type ((self type)) : TYPE [Function]
Return the unwrapped type for the wrapped type self, or self if it is not a wrapped type.

wrapper-value-type ((self wrapper)) : TYPE [Function]
Return the type of the value stored in the wrapper self.

write-html-escaping-url-special-characters ((stream native-output-stream) (input string)) :
Writes a string and replaces unallowed URL characters according to RFC 2396 with %-format URL escape sequences.

writer (self) : SYMBOL [Method on STORAGE-SLOT]
Name of a method called to write the value of the slot self.

xml-declaration-form? ((form object)) : BOOLEAN [Function]
Return true if form is a CONS headed by an XML DECLARATION tag

xml-doctype-form? ((form object)) : BOOLEAN [Function]
Return true if form is a CONS headed by a DOCTYPE tag
Chapter 6: Library Functions

xml-element-form? ((form object)) : BOOLEAN
Return true if form is a CONS headed by an XML ELEMENT tag

xml-processing-instruction-form? ((form object)) : BOOLEAN
Return true if form is a CONS headed by an XML PROCESSING INSTRUCTION tag

xml-processing-instruction? ((item object)) : BOOLEAN
Return true if item is an XML processing instruction object

xml-token-list-to-s-expression ((tokenList tokenizer-token) (doctype xml-doctype) (doctype-definition? boolean)) : OBJECT
Convert the XML tokenList (using doctype for guidance) into a representative s-expression and return the result. The doctype argument is currently only used for expansion of entity references. It can be null. The flag doctype-definition? should be true only when processing the DTD definition of a DOCTYPE tag, since it enables substitution of parameter entity values.

Every XML tag is represented as a cons-list starting with the tag as its header, followed by a possibly empty list of keyword value pairs representing tag attributes, followed by a possibly empty list of content expressions which might themselves be XML expressions. For example, the expression

<\n\n<a a1=v1 a2='v2'> foo <b a3=v3/> bar </a>

becomes

(<a> (<a1> "v1" <a2> "v2") "foo" (<b> (<a3> "v3") "bar")

when represented as an s-expression. The tag names are subtypes of XML-OBJECT such as XML-ELEMENT, XML-LOCAL-ATTRIBUTE, XML-GLOBAL-ATTRIBUTE, etc. ?, ! and [ prefixed tags are encoded as their own subtypes of XML-OBJECT, namely XML-PROCESSING-INSTRUCTION, XML-DECLARATION, XML-SPECIAL, XML-COMMENT, etc. CDATA is an XML-SPECIAL tag with a name of CDATA.

The name is available using class accessors.

yield-define-stella-class ((class CLASS)) : CONS
Return a cons tree that (when evaluated) constructs a Stella class object.

zero-pad-integer ((value INTEGER) (size INTEGER)) : STRING
Returns a string representing value of at least length size, padded if necessary with 0 characters.

zero? (x) : BOOLEAN
Return true if x is 0.

zero? (x) : BOOLEAN
Return true if x is 0.
# Function Index

<table>
<thead>
<tr>
<th>*</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>27</td>
</tr>
<tr>
<td>++</td>
<td>28</td>
</tr>
<tr>
<td>-</td>
<td>27</td>
</tr>
<tr>
<td>--</td>
<td>28</td>
</tr>
<tr>
<td>/</td>
<td>28</td>
</tr>
<tr>
<td>//</td>
<td>28</td>
</tr>
<tr>
<td>&lt;</td>
<td>28</td>
</tr>
<tr>
<td>&lt;=</td>
<td>28</td>
</tr>
<tr>
<td>=</td>
<td>28</td>
</tr>
<tr>
<td>&gt;</td>
<td>28</td>
</tr>
<tr>
<td>&gt;=</td>
<td>28</td>
</tr>
<tr>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>1+</td>
<td>28</td>
</tr>
<tr>
<td>1-</td>
<td>28</td>
</tr>
<tr>
<td>1d-aref</td>
<td>74</td>
</tr>
<tr>
<td>1d-aref-address</td>
<td>74</td>
</tr>
<tr>
<td>1d-aref-setter</td>
<td>74</td>
</tr>
<tr>
<td>2</td>
<td>74</td>
</tr>
<tr>
<td>2d-aref</td>
<td>74</td>
</tr>
<tr>
<td>2d-aref-address</td>
<td>74</td>
</tr>
<tr>
<td>2d-aref-setter</td>
<td>74</td>
</tr>
</tbody>
</table>

A

<table>
<thead>
<tr>
<th>abs on FLOAT</th>
<th>29</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs on INTEGER</td>
<td>29</td>
</tr>
<tr>
<td>abs on LONG-INTEGER</td>
<td>74</td>
</tr>
<tr>
<td>absolute-pathname?</td>
<td>68</td>
</tr>
<tr>
<td>acos</td>
<td>30</td>
</tr>
<tr>
<td>activate-demon</td>
<td>75</td>
</tr>
<tr>
<td>active? on POLYMORPHIC-RELATION</td>
<td>75</td>
</tr>
<tr>
<td>add-configuration-property</td>
<td>75</td>
</tr>
<tr>
<td>add-hook</td>
<td>75</td>
</tr>
<tr>
<td>add-trace</td>
<td>75</td>
</tr>
<tr>
<td>advance on ITERATOR</td>
<td>59</td>
</tr>
<tr>
<td>advance-past-whitespace</td>
<td>75</td>
</tr>
<tr>
<td>all-classes</td>
<td>75</td>
</tr>
<tr>
<td>all-contexts</td>
<td>75</td>
</tr>
<tr>
<td>all-defined?</td>
<td>75</td>
</tr>
<tr>
<td>all-functions</td>
<td>75</td>
</tr>
<tr>
<td>all-included-modules</td>
<td>75</td>
</tr>
<tr>
<td>all-lower-case-string?</td>
<td>32</td>
</tr>
<tr>
<td>all-methods</td>
<td>75</td>
</tr>
<tr>
<td>all-modules</td>
<td>75</td>
</tr>
<tr>
<td>all-public-functions</td>
<td>76</td>
</tr>
<tr>
<td>all-public-methods</td>
<td>76</td>
</tr>
<tr>
<td>all-required-systems</td>
<td>76</td>
</tr>
<tr>
<td>all-slots</td>
<td>76</td>
</tr>
<tr>
<td>all-subcontexts</td>
<td>76</td>
</tr>
<tr>
<td>all-surrogates</td>
<td>76</td>
</tr>
<tr>
<td>all-symbols</td>
<td>76</td>
</tr>
<tr>
<td>all-upper-case-string?</td>
<td>32</td>
</tr>
<tr>
<td>all-variables</td>
<td>76</td>
</tr>
<tr>
<td>allocate-iterator on ABSTRACT-ITERATOR</td>
<td>76</td>
</tr>
<tr>
<td>allocate-iterator on CONS</td>
<td>40</td>
</tr>
<tr>
<td>allocate-iterator on KEY-VALUE-LIST</td>
<td>49</td>
</tr>
<tr>
<td>allocate-iterator on KEY-VALUE-MAP</td>
<td>57</td>
</tr>
<tr>
<td>allocate-iterator on LIST</td>
<td>44</td>
</tr>
<tr>
<td>allocate-iterator on MEMOIZABLE-ITERATOR</td>
<td>76</td>
</tr>
<tr>
<td>allocate-iterator on PROPERTY-LIST</td>
<td>47</td>
</tr>
<tr>
<td>allocate-iterator on STELLA-HASH-TABLE</td>
<td>54</td>
</tr>
<tr>
<td>allocation on STORAGE-SLOT</td>
<td>76</td>
</tr>
<tr>
<td>append</td>
<td>38</td>
</tr>
<tr>
<td>apply</td>
<td>76</td>
</tr>
<tr>
<td>apply-boolean-method</td>
<td>77</td>
</tr>
<tr>
<td>apply-float-method</td>
<td>77</td>
</tr>
<tr>
<td>apply-integer-method</td>
<td>77</td>
</tr>
<tr>
<td>apply-long-integer-method</td>
<td>77</td>
</tr>
<tr>
<td>apply-method</td>
<td>77</td>
</tr>
<tr>
<td>apply-string-method</td>
<td>77</td>
</tr>
<tr>
<td>asin</td>
<td>29</td>
</tr>
<tr>
<td>atan</td>
<td>30</td>
</tr>
<tr>
<td>atan2</td>
<td>30</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>base60-to-float</td>
<td>77</td>
</tr>
<tr>
<td>break-program</td>
<td>77</td>
</tr>
<tr>
<td>bump-log-indent</td>
<td>77</td>
</tr>
<tr>
<td>but-last on CONS</td>
<td>37</td>
</tr>
<tr>
<td>but-last on LIST</td>
<td>42</td>
</tr>
<tr>
<td>but-last on VECTOR</td>
<td>50</td>
</tr>
<tr>
<td>byte-array-read-sequence</td>
<td>77</td>
</tr>
<tr>
<td>byte-array-write-sequence</td>
<td>77</td>
</tr>
<tr>
<td>calendar-date-to-date-string on CALENDAR-DATE</td>
<td>77</td>
</tr>
<tr>
<td>calendar-date-to-is08601-string on CALENDAR-DATE</td>
<td>78</td>
</tr>
<tr>
<td>calendar-date-to-string</td>
<td>69</td>
</tr>
<tr>
<td>calendar-date-to-string on CALENDAR-DATE</td>
<td>78</td>
</tr>
<tr>
<td>calendar-date-to-time-string on CALENDAR-DATE</td>
<td>78</td>
</tr>
<tr>
<td>call-clear-module</td>
<td>78</td>
</tr>
<tr>
<td>case</td>
<td>9</td>
</tr>
<tr>
<td>cast</td>
<td>78</td>
</tr>
<tr>
<td>cc</td>
<td>63</td>
</tr>
<tr>
<td>ceiling</td>
<td>29</td>
</tr>
<tr>
<td>change-context on CONTEXT</td>
<td>63</td>
</tr>
<tr>
<td>change-context on STRING</td>
<td>63</td>
</tr>
<tr>
<td>change-module on MODULE</td>
<td>65</td>
</tr>
<tr>
<td>change-module on STRING</td>
<td>65</td>
</tr>
<tr>
<td>character-capitalizie</td>
<td>31</td>
</tr>
<tr>
<td>character-code</td>
<td>30</td>
</tr>
<tr>
<td>character-downcase</td>
<td>31</td>
</tr>
<tr>
<td>character-to-string</td>
<td>31, 36</td>
</tr>
<tr>
<td>character-upcase</td>
<td>31</td>
</tr>
<tr>
<td>choose</td>
<td>9</td>
</tr>
<tr>
<td>cl-slot-value</td>
<td>78</td>
</tr>
<tr>
<td>cl-slot-value-setter</td>
<td>78</td>
</tr>
<tr>
<td>cl-translate-file</td>
<td>79</td>
</tr>
<tr>
<td>cl-translate-system</td>
<td>79</td>
</tr>
<tr>
<td>cleanup-unfinalized-classes</td>
<td>79</td>
</tr>
<tr>
<td>clear on HEAP</td>
<td>79</td>
</tr>
<tr>
<td>clear on KEY-VALUE-LIST</td>
<td>49</td>
</tr>
<tr>
<td>clear on KEY-VALUE-MAP</td>
<td>49</td>
</tr>
<tr>
<td>clear on LIST</td>
<td>44</td>
</tr>
<tr>
<td>clear on PROPERTY-LIST</td>
<td>47</td>
</tr>
<tr>
<td>clear on STELLA-HASH-TABLE</td>
<td>54</td>
</tr>
<tr>
<td>clear on VECTOR</td>
<td>51</td>
</tr>
<tr>
<td>clear on VECTOR-SEQUENCE</td>
<td>79</td>
</tr>
<tr>
<td>clear-configuration-property</td>
<td>79</td>
</tr>
<tr>
<td>clear-context</td>
<td>62</td>
</tr>
<tr>
<td>clear-module</td>
<td>65</td>
</tr>
<tr>
<td>clear-recycle-list</td>
<td>79</td>
</tr>
<tr>
<td>clear-recycle-lists</td>
<td>79</td>
</tr>
<tr>
<td>clear-system</td>
<td>79</td>
</tr>
<tr>
<td>clear-trace</td>
<td>79</td>
</tr>
<tr>
<td>clone-memoized-iterator</td>
<td>79</td>
</tr>
<tr>
<td>close-all-files</td>
<td>67</td>
</tr>
<tr>
<td>close-stream</td>
<td>67</td>
</tr>
<tr>
<td>clv</td>
<td>79</td>
</tr>
<tr>
<td>code-character</td>
<td>30</td>
</tr>
<tr>
<td>coerce-#rest-to-cons</td>
<td>80</td>
</tr>
<tr>
<td>coerce-to-float</td>
<td>80</td>
</tr>
<tr>
<td>coerce-to-hash-set</td>
<td>80</td>
</tr>
<tr>
<td>coerce-to-symbol</td>
<td>80</td>
</tr>
<tr>
<td>collect</td>
<td>80</td>
</tr>
<tr>
<td>collection-valued? on SLOT</td>
<td>80</td>
</tr>
<tr>
<td>command?</td>
<td>80</td>
</tr>
<tr>
<td>component? on STORAGE-SLOT</td>
<td>80</td>
</tr>
<tr>
<td>compose-namestring</td>
<td>80</td>
</tr>
<tr>
<td>compose-namestring-full</td>
<td>81</td>
</tr>
<tr>
<td>compute-calendar-date</td>
<td>70</td>
</tr>
<tr>
<td>compute-day-of-week</td>
<td>70</td>
</tr>
<tr>
<td>compute-day-of-week-julian</td>
<td>70</td>
</tr>
<tr>
<td>compute-julian-day</td>
<td>70</td>
</tr>
<tr>
<td>compute-module-and-bare-name</td>
<td>81</td>
</tr>
<tr>
<td>compute-next-moon-phase</td>
<td>70</td>
</tr>
<tr>
<td>concatenate on CONS</td>
<td>38</td>
</tr>
<tr>
<td>concatenate on ITERATOR</td>
<td>59</td>
</tr>
<tr>
<td>concatenate on LIST</td>
<td>44</td>
</tr>
<tr>
<td>concatenate on SET</td>
<td>46</td>
</tr>
<tr>
<td>concatenate on STRING</td>
<td>34</td>
</tr>
<tr>
<td>cond</td>
<td>9</td>
</tr>
<tr>
<td>configure-stella</td>
<td>81</td>
</tr>
<tr>
<td>cons</td>
<td>37</td>
</tr>
<tr>
<td>cons-list</td>
<td>39</td>
</tr>
<tr>
<td>cons-tree-match?</td>
<td>40</td>
</tr>
<tr>
<td>cons-tree-nth</td>
<td>39</td>
</tr>
<tr>
<td>cons-tree-nth-rest</td>
<td>39</td>
</tr>
<tr>
<td>consify on CONS</td>
<td>40</td>
</tr>
<tr>
<td>consify on HASH-SET</td>
<td>58</td>
</tr>
<tr>
<td>consify on ITERATOR</td>
<td>60</td>
</tr>
<tr>
<td>consify on KEY-VALUE-LIST</td>
<td>49</td>
</tr>
<tr>
<td>consify on KEY-VALUE-MAP</td>
<td>57</td>
</tr>
<tr>
<td>consify on LIST</td>
<td>44</td>
</tr>
<tr>
<td>consify on OBJECT</td>
<td>81</td>
</tr>
<tr>
<td>consify on STELLA-HASH-TABLE</td>
<td>54</td>
</tr>
<tr>
<td>consify on VECTOR</td>
<td>51</td>
</tr>
<tr>
<td>consify-command-line-arguments</td>
<td>81</td>
</tr>
<tr>
<td>continuable-error</td>
<td>81</td>
</tr>
<tr>
<td>copy on CUSTOM-VECTOR-SEQUENCE</td>
<td>81</td>
</tr>
<tr>
<td>copy on HASH-SET</td>
<td>58</td>
</tr>
<tr>
<td>copy on KEY-VALUE-LIST</td>
<td>49</td>
</tr>
<tr>
<td>copy on KEY-VALUE-MAP</td>
<td>57</td>
</tr>
<tr>
<td>copy on LIST</td>
<td>44</td>
</tr>
<tr>
<td>copy on PROPERTY-LIST</td>
<td>47</td>
</tr>
<tr>
<td>copy on STELLA-HASH-TABLE</td>
<td>54</td>
</tr>
<tr>
<td>copy on STRING</td>
<td>34</td>
</tr>
<tr>
<td>copy on VECTOR</td>
<td>51</td>
</tr>
<tr>
<td>copy on VECTOR-SEQUENCE</td>
<td>81</td>
</tr>
<tr>
<td>copy-cons-list</td>
<td>39</td>
</tr>
<tr>
<td>copy-cons-tree</td>
<td>39</td>
</tr>
<tr>
<td>copy-file</td>
<td>67</td>
</tr>
<tr>
<td>copy-kv-cons-list</td>
<td>47</td>
</tr>
<tr>
<td>cos</td>
<td>29</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>cpp-translate-system</td>
<td>81</td>
</tr>
<tr>
<td>cpptrans</td>
<td>81</td>
</tr>
<tr>
<td>create-derived-list</td>
<td>81</td>
</tr>
<tr>
<td>create-object</td>
<td>81</td>
</tr>
<tr>
<td>create-world</td>
<td>65</td>
</tr>
<tr>
<td>directory-separator-string</td>
<td>68</td>
</tr>
<tr>
<td>disable-memoization</td>
<td>84</td>
</tr>
<tr>
<td>disabled-stella-feature?</td>
<td>84</td>
</tr>
<tr>
<td>div</td>
<td>28</td>
</tr>
<tr>
<td>div on INTEGER</td>
<td>85</td>
</tr>
<tr>
<td>div on LONG-INTEGER</td>
<td>85</td>
</tr>
<tr>
<td>drop-hook</td>
<td>85</td>
</tr>
<tr>
<td>drop-trace</td>
<td>85</td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>deactivate-demon</td>
<td>82</td>
</tr>
<tr>
<td>decode-calendar-date on CALENDAR-DATE</td>
<td>69</td>
</tr>
<tr>
<td>decode-time-in-millis</td>
<td>70</td>
</tr>
<tr>
<td>decoded-date-time-to-iso8601-string on DECODED-DATE-TIME</td>
<td>82</td>
</tr>
<tr>
<td>decompose-namestring</td>
<td>82</td>
</tr>
<tr>
<td>decompose-namestring-full</td>
<td>82</td>
</tr>
<tr>
<td>default-form on STORAGE-SLOT</td>
<td>82</td>
</tr>
<tr>
<td>def demon</td>
<td>82</td>
</tr>
<tr>
<td>define-demon</td>
<td>82</td>
</tr>
<tr>
<td>define-logical-host-property</td>
<td>83</td>
</tr>
<tr>
<td>define-module</td>
<td>83</td>
</tr>
<tr>
<td>define-stella-class</td>
<td>83</td>
</tr>
<tr>
<td>define-stella-method-slot</td>
<td>83</td>
</tr>
<tr>
<td>defined-list?</td>
<td>42</td>
</tr>
<tr>
<td>defined? on ARRAY</td>
<td>83</td>
</tr>
<tr>
<td>defined? on CHARACTER</td>
<td>27</td>
</tr>
<tr>
<td>defined? on CODE</td>
<td>27</td>
</tr>
<tr>
<td>defined? on FLOAT</td>
<td>27</td>
</tr>
<tr>
<td>defined? on INTEGER</td>
<td>27</td>
</tr>
<tr>
<td>defined? on LONG-INTEGER</td>
<td>83</td>
</tr>
<tr>
<td>defined? on MUX-STRING</td>
<td>26</td>
</tr>
<tr>
<td>defined? on NATIVE-VECTOR</td>
<td>26</td>
</tr>
<tr>
<td>defined? on OBJECT</td>
<td>26</td>
</tr>
<tr>
<td>defined? on SECOND-CLASS-OBJECT</td>
<td>26</td>
</tr>
<tr>
<td>defined? on STRING</td>
<td>26</td>
</tr>
<tr>
<td>defmain</td>
<td>83</td>
</tr>
<tr>
<td>defmodule</td>
<td>63</td>
</tr>
<tr>
<td>defsystem</td>
<td>83</td>
</tr>
<tr>
<td>delete-file</td>
<td>67</td>
</tr>
<tr>
<td>deleted? on OBJECT</td>
<td>84</td>
</tr>
<tr>
<td>describe</td>
<td>84</td>
</tr>
<tr>
<td>describe-object on OBJECT</td>
<td>84</td>
</tr>
<tr>
<td>destroy-class on CLASS</td>
<td>84</td>
</tr>
<tr>
<td>destroy-class-and-subclasses</td>
<td>84</td>
</tr>
<tr>
<td>destroy-context on CONTEXT</td>
<td>63</td>
</tr>
<tr>
<td>destroy-context on MODULE</td>
<td>65</td>
</tr>
<tr>
<td>destroy-context on STRING</td>
<td>63</td>
</tr>
<tr>
<td>destroy-context on WORLD</td>
<td>65</td>
</tr>
<tr>
<td>destroy-module</td>
<td>65</td>
</tr>
<tr>
<td>destroy-defmethod-tree</td>
<td>84</td>
</tr>
<tr>
<td>dictionary</td>
<td>84</td>
</tr>
<tr>
<td>difference on CONS</td>
<td>41</td>
</tr>
<tr>
<td>difference on HASH-SET</td>
<td>59</td>
</tr>
<tr>
<td>difference on LIST</td>
<td>45</td>
</tr>
<tr>
<td>digit-character?</td>
<td>30</td>
</tr>
<tr>
<td>direct-super-classes on CLASS</td>
<td>84</td>
</tr>
<tr>
<td>directory-file-name</td>
<td>68</td>
</tr>
<tr>
<td>directory-parent-directory</td>
<td>68</td>
</tr>
<tr>
<td>directory-separator</td>
<td>68</td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>either</td>
<td>85</td>
</tr>
<tr>
<td>empty? on CONS</td>
<td>36</td>
</tr>
<tr>
<td>empty? on HEAP</td>
<td>85</td>
</tr>
<tr>
<td>empty? on ITERATOR</td>
<td>59</td>
</tr>
<tr>
<td>empty? on KEY-VALUE-LIST</td>
<td>48</td>
</tr>
<tr>
<td>empty? on KEY-VALUE-MAP</td>
<td>57</td>
</tr>
<tr>
<td>empty? on LIST</td>
<td>42</td>
</tr>
<tr>
<td>empty? on PROPERTY-LIST</td>
<td>46</td>
</tr>
<tr>
<td>empty? on STELLA-HASH-TABLE</td>
<td>54</td>
</tr>
<tr>
<td>empty? on STRING</td>
<td>31</td>
</tr>
<tr>
<td>empty? on STRING-WRAPPER</td>
<td>85</td>
</tr>
<tr>
<td>empty? on VECTOR</td>
<td>49</td>
</tr>
<tr>
<td>empty? on VECTOR-SEQUENCE</td>
<td>85</td>
</tr>
<tr>
<td>enable-memoization</td>
<td>85</td>
</tr>
<tr>
<td>enabled-stella-feature?</td>
<td>85</td>
</tr>
<tr>
<td>encode-calendar-date on DECODED-DATE-TIME</td>
<td>69</td>
</tr>
<tr>
<td>ensure-file-does-not-exist</td>
<td>85</td>
</tr>
<tr>
<td>ensure-file-exists</td>
<td>85</td>
</tr>
<tr>
<td>eq?</td>
<td>27</td>
</tr>
<tr>
<td>eql-except-in-whitespace?</td>
<td>85</td>
</tr>
<tr>
<td>eql?</td>
<td>27</td>
</tr>
<tr>
<td>equal-cons-trees?</td>
<td>36</td>
</tr>
<tr>
<td>equal-hash-code on CONS</td>
<td>37</td>
</tr>
<tr>
<td>equal-hash-code on HASH-SET</td>
<td>59</td>
</tr>
<tr>
<td>equal-hash-code on KEY-VALUE-LIST</td>
<td>48</td>
</tr>
<tr>
<td>equal-hash-code on KEY-VALUE-MAP</td>
<td>57</td>
</tr>
<tr>
<td>equal-hash-code on LIST</td>
<td>42</td>
</tr>
<tr>
<td>equal-hash-code on OBJECT</td>
<td>55</td>
</tr>
<tr>
<td>equal-hash-code on PROPERTY-LIST</td>
<td>46</td>
</tr>
<tr>
<td>equal-hash-code on SET</td>
<td>46</td>
</tr>
<tr>
<td>equal-hash-code on STELLA-HASH-TABLE</td>
<td>54</td>
</tr>
<tr>
<td>equal-hash-code on VECTOR</td>
<td>50</td>
</tr>
<tr>
<td>equal?</td>
<td>27</td>
</tr>
<tr>
<td>equivalent-sets? on CONS</td>
<td>41</td>
</tr>
<tr>
<td>equivalent-sets? on HASH-SET</td>
<td>58</td>
</tr>
<tr>
<td>equivalent-sets? on LIST</td>
<td>45</td>
</tr>
<tr>
<td>error</td>
<td>85</td>
</tr>
<tr>
<td>evaluate</td>
<td>85</td>
</tr>
<tr>
<td>evaluate-string</td>
<td>86</td>
</tr>
<tr>
<td>even?</td>
<td>28</td>
</tr>
<tr>
<td>even? on INTEGER</td>
<td>86</td>
</tr>
<tr>
<td>even? on LONG-INTEGER</td>
<td>86</td>
</tr>
<tr>
<td>exception-message</td>
<td>86</td>
</tr>
<tr>
<td>exp</td>
<td>29</td>
</tr>
<tr>
<td>exp?</td>
<td>29</td>
</tr>
<tr>
<td>extension on CLASS</td>
<td>86</td>
</tr>
<tr>
<td>Function Name</td>
<td>Page</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
</tr>
<tr>
<td>external-id-head?</td>
<td>86</td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>26</td>
</tr>
<tr>
<td>fast-heap-root on HEAP</td>
<td>86</td>
</tr>
<tr>
<td>fifth on CONS</td>
<td>37</td>
</tr>
<tr>
<td>fifth on LIST</td>
<td>42</td>
</tr>
<tr>
<td>fifth on MUTABLE-STRING</td>
<td>33</td>
</tr>
<tr>
<td>fifth on STRING</td>
<td>33</td>
</tr>
<tr>
<td>fifth on VECTOR</td>
<td>50</td>
</tr>
<tr>
<td>file-base-name</td>
<td>68</td>
</tr>
<tr>
<td>file-extension</td>
<td>68</td>
</tr>
<tr>
<td>file-length</td>
<td>67</td>
</tr>
<tr>
<td>file-name-as-directory</td>
<td>68</td>
</tr>
<tr>
<td>file-name-directory</td>
<td>68</td>
</tr>
<tr>
<td>file-name-without-directory</td>
<td>68</td>
</tr>
<tr>
<td>file-name-without-extension</td>
<td>68</td>
</tr>
<tr>
<td>file-write-date</td>
<td>67</td>
</tr>
<tr>
<td>fill-in-date-substitution</td>
<td>86</td>
</tr>
<tr>
<td>finalize-classes</td>
<td>86</td>
</tr>
<tr>
<td>finalize-classes-and-slots</td>
<td>86</td>
</tr>
<tr>
<td>finalize-slots</td>
<td>86</td>
</tr>
<tr>
<td>find-matching-prefix-length</td>
<td>86</td>
</tr>
<tr>
<td>find-mismatch</td>
<td>87</td>
</tr>
<tr>
<td>find-or-create-module</td>
<td>64</td>
</tr>
<tr>
<td>first on CONS</td>
<td>37</td>
</tr>
<tr>
<td>first on LIST</td>
<td>42</td>
</tr>
<tr>
<td>first on MUTABLE-STRING</td>
<td>32</td>
</tr>
<tr>
<td>first on STRING</td>
<td>32</td>
</tr>
<tr>
<td>first on VECTOR</td>
<td>50</td>
</tr>
<tr>
<td>first-defined</td>
<td>87</td>
</tr>
<tr>
<td>float-to-base60</td>
<td>87</td>
</tr>
<tr>
<td>float-to-string</td>
<td>30, 35</td>
</tr>
<tr>
<td>floor</td>
<td>29</td>
</tr>
<tr>
<td>flush-output</td>
<td>66</td>
</tr>
<tr>
<td>fmod</td>
<td>87</td>
</tr>
<tr>
<td>format-float</td>
<td>30, 36</td>
</tr>
<tr>
<td>format-with-padding</td>
<td>87</td>
</tr>
<tr>
<td>fourth on CONS</td>
<td>37</td>
</tr>
<tr>
<td>fourth on LIST</td>
<td>42</td>
</tr>
<tr>
<td>fourth on MUTABLE-STRING</td>
<td>33</td>
</tr>
<tr>
<td>fourth on STRING</td>
<td>33</td>
</tr>
<tr>
<td>fourth on VECTOR</td>
<td>50</td>
</tr>
<tr>
<td>free on ACTIVE-OBJECT</td>
<td>87</td>
</tr>
<tr>
<td>free on OBJECT</td>
<td>87</td>
</tr>
<tr>
<td>free-hash-table-values on ABSTRACT-HASH-TABLE</td>
<td>87</td>
</tr>
<tr>
<td>frem</td>
<td>87</td>
</tr>
<tr>
<td>G</td>
<td></td>
</tr>
<tr>
<td>gcd</td>
<td>29</td>
</tr>
<tr>
<td>generate-random-uuid</td>
<td>87</td>
</tr>
<tr>
<td>generate-uuid</td>
<td>88</td>
</tr>
<tr>
<td>gensym</td>
<td>88</td>
</tr>
<tr>
<td>get-calendar-date on CALENDAR-DATE</td>
<td>88</td>
</tr>
<tr>
<td>get-current-date-time</td>
<td>69</td>
</tr>
<tr>
<td>get-global-value</td>
<td>88</td>
</tr>
<tr>
<td>get-local-standard-time-zone</td>
<td>88</td>
</tr>
<tr>
<td>get-local-time-zone</td>
<td>69</td>
</tr>
<tr>
<td>get-local-time-zone-for-date</td>
<td>88</td>
</tr>
<tr>
<td>get-quoted-tree</td>
<td>88</td>
</tr>
<tr>
<td>get-slot</td>
<td>88</td>
</tr>
<tr>
<td>get-stella-class on STRING</td>
<td>88</td>
</tr>
<tr>
<td>get-stella-class on SURROGATE</td>
<td>88</td>
</tr>
<tr>
<td>get-stella-class on SYMBOL</td>
<td>88</td>
</tr>
<tr>
<td>get-stella-context</td>
<td>62</td>
</tr>
<tr>
<td>get-stella-module</td>
<td>64</td>
</tr>
<tr>
<td>get-ticktock</td>
<td>71</td>
</tr>
<tr>
<td>get-time on CALENDAR-DATE</td>
<td>89</td>
</tr>
<tr>
<td>get-token-float</td>
<td>89</td>
</tr>
<tr>
<td>get-token-integer</td>
<td>89</td>
</tr>
<tr>
<td>get-token-long-integer</td>
<td>89</td>
</tr>
<tr>
<td>get-xml-attributes</td>
<td>72</td>
</tr>
<tr>
<td>get-xml-cdata-content</td>
<td>72</td>
</tr>
<tr>
<td>get-xml-content</td>
<td>72</td>
</tr>
<tr>
<td>get-xml-tag</td>
<td>71</td>
</tr>
<tr>
<td>global-variable-type-spec</td>
<td>89</td>
</tr>
<tr>
<td>H</td>
<td></td>
</tr>
<tr>
<td>hash-code on BOOLEAN-WRAPPER</td>
<td>55</td>
</tr>
<tr>
<td>hash-code on CHARACTER</td>
<td>55</td>
</tr>
<tr>
<td>hash-code on CHARACTER-WRAPPER</td>
<td>55</td>
</tr>
<tr>
<td>hash-code on FLOAT</td>
<td>55</td>
</tr>
<tr>
<td>hash-code on FLOAT-WRAPPER</td>
<td>55</td>
</tr>
<tr>
<td>hash-code on INTEGER</td>
<td>55</td>
</tr>
<tr>
<td>hash-code on INTEGER-WRAPPER</td>
<td>55</td>
</tr>
<tr>
<td>hash-code on OBJECT</td>
<td>55</td>
</tr>
<tr>
<td>hash-code on STANDARD-OBJECT</td>
<td>55</td>
</tr>
<tr>
<td>hash-code on STRING</td>
<td>55</td>
</tr>
<tr>
<td>hash-code on STRING-WRAPPER</td>
<td>55</td>
</tr>
<tr>
<td>hash-set</td>
<td>58</td>
</tr>
<tr>
<td>hash-string</td>
<td>89</td>
</tr>
<tr>
<td>hashmod</td>
<td>55</td>
</tr>
<tr>
<td>heap-root on HEAP</td>
<td>89</td>
</tr>
<tr>
<td>heapify on HEAP</td>
<td>89</td>
</tr>
<tr>
<td>help-advance-past-whitespace</td>
<td>89</td>
</tr>
<tr>
<td>help-find-matching-prefix-length</td>
<td>89</td>
</tr>
<tr>
<td>help-get-stella-module</td>
<td>90</td>
</tr>
<tr>
<td>help-print-outline on CLASS</td>
<td>90</td>
</tr>
<tr>
<td>help-print-outline on CONTEXT</td>
<td>90</td>
</tr>
<tr>
<td>help-print-outline on MODULE</td>
<td>90</td>
</tr>
<tr>
<td>help-print-outline on OBJECT</td>
<td>90</td>
</tr>
<tr>
<td>help-print-outline on SLOT</td>
<td>90</td>
</tr>
<tr>
<td>home-module on OBJECT</td>
<td>90</td>
</tr>
<tr>
<td>Function Name</td>
<td>Page</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
</tr>
<tr>
<td>log</td>
<td>29</td>
</tr>
<tr>
<td>log-level&lt;=</td>
<td>94</td>
</tr>
<tr>
<td>log-message</td>
<td>94</td>
</tr>
<tr>
<td>log10</td>
<td>29</td>
</tr>
<tr>
<td>logical-host?</td>
<td>68</td>
</tr>
<tr>
<td>logical-pathname?</td>
<td>68</td>
</tr>
<tr>
<td>logmsg</td>
<td>94</td>
</tr>
<tr>
<td>lookup on HASH-TABLE</td>
<td>52</td>
</tr>
<tr>
<td>lookup on INTEGER-HASH-TABLE</td>
<td>53</td>
</tr>
<tr>
<td>lookup on KEY-VALUE-LIST</td>
<td>48</td>
</tr>
<tr>
<td>lookup on KEY-VALUE-MAP</td>
<td>56</td>
</tr>
<tr>
<td>lookup on PROPERTY-LIST</td>
<td>46</td>
</tr>
<tr>
<td>lookup on STELLA-HASH-TABLE</td>
<td>53</td>
</tr>
<tr>
<td>lookup on STRING-HASH-TABLE</td>
<td>52</td>
</tr>
<tr>
<td>lookup on STRING-TO-INTEGER-HASH-TABLE</td>
<td>52</td>
</tr>
<tr>
<td>lookup-class on STRING</td>
<td>94</td>
</tr>
<tr>
<td>lookup-class on SYMBOL</td>
<td>94</td>
</tr>
<tr>
<td>lookup-command</td>
<td>94</td>
</tr>
<tr>
<td>lookup-configuration-property</td>
<td>94</td>
</tr>
<tr>
<td>lookup-configuration-property-values</td>
<td>94</td>
</tr>
<tr>
<td>lookup-demon</td>
<td>94</td>
</tr>
<tr>
<td>lookup-function</td>
<td>94</td>
</tr>
<tr>
<td>lookup-function-by-name</td>
<td>94</td>
</tr>
<tr>
<td>lookup-global-variable on GENERALIZED-SYMBOL</td>
<td>95</td>
</tr>
<tr>
<td>lookup-global-variable on STRING</td>
<td>95</td>
</tr>
<tr>
<td>lookup-global-variable on SURROGATE</td>
<td>95</td>
</tr>
<tr>
<td>lookup-keyword</td>
<td>62</td>
</tr>
<tr>
<td>lookup-local-slot</td>
<td>95</td>
</tr>
<tr>
<td>lookup-logging-parameter</td>
<td>95</td>
</tr>
<tr>
<td>lookup-macro</td>
<td>95</td>
</tr>
<tr>
<td>lookup-slot</td>
<td>95</td>
</tr>
<tr>
<td>lookup-surrogate</td>
<td>61</td>
</tr>
<tr>
<td>lookup-surrogate-in-module</td>
<td>61</td>
</tr>
<tr>
<td>lookup-symbol</td>
<td>60</td>
</tr>
<tr>
<td>lookup-symbol-in-module</td>
<td>60</td>
</tr>
<tr>
<td>lookup-visible-surrogates-in-module</td>
<td>61</td>
</tr>
<tr>
<td>lookup-visible-symbols-in-module</td>
<td>60</td>
</tr>
<tr>
<td>lookup-with-default on KEY-VALUE-LIST</td>
<td>95</td>
</tr>
<tr>
<td>lookup-with-default on PROPERTY-LIST</td>
<td>95</td>
</tr>
<tr>
<td>lower-case-character?</td>
<td>31</td>
</tr>
<tr>
<td>lptrans</td>
<td>95</td>
</tr>
<tr>
<td>match-cons-tree</td>
<td>40</td>
</tr>
<tr>
<td>max</td>
<td>29</td>
</tr>
<tr>
<td>max on FLOAT</td>
<td>97</td>
</tr>
<tr>
<td>max on INTEGER</td>
<td>97</td>
</tr>
<tr>
<td>max on LONG-INTEGER</td>
<td>97</td>
</tr>
<tr>
<td>max on NUMBER-WRAPPER</td>
<td>97</td>
</tr>
<tr>
<td>mem? on CONS</td>
<td>38</td>
</tr>
<tr>
<td>mem? on LIST</td>
<td>43</td>
</tr>
<tr>
<td>member? on COLLECTION</td>
<td>97</td>
</tr>
<tr>
<td>member? on CONS</td>
<td>37</td>
</tr>
<tr>
<td>member? on CONS-ITERATOR</td>
<td>97</td>
</tr>
<tr>
<td>member? on HASH-SET</td>
<td>58</td>
</tr>
<tr>
<td>member? on ITERATOR</td>
<td>59</td>
</tr>
<tr>
<td>member? on LIST</td>
<td>43</td>
</tr>
<tr>
<td>member? on SEQUENCE</td>
<td>97</td>
</tr>
<tr>
<td>member? on STRING</td>
<td>33</td>
</tr>
<tr>
<td>member? on VECTOR</td>
<td>50</td>
</tr>
<tr>
<td>memoize</td>
<td>97</td>
</tr>
<tr>
<td>merge-file-names</td>
<td>98</td>
</tr>
<tr>
<td>merge-null-fields on DECODED-DATA-TIME</td>
<td>98</td>
</tr>
<tr>
<td>merge-superior-null-fields on DECODED-DATA-TIME</td>
<td>98</td>
</tr>
<tr>
<td>min</td>
<td>29</td>
</tr>
<tr>
<td>min on FLOAT</td>
<td>98</td>
</tr>
<tr>
<td>min on INTEGER</td>
<td>98</td>
</tr>
<tr>
<td>min on LONG-INTEGER</td>
<td>98</td>
</tr>
<tr>
<td>min on NUMBER-WRAPPER</td>
<td>98</td>
</tr>
<tr>
<td>mod</td>
<td>29</td>
</tr>
<tr>
<td>mod on INTEGER</td>
<td>98</td>
</tr>
<tr>
<td>mod on LONG-INTEGER</td>
<td>98</td>
</tr>
<tr>
<td>modified-julian-day-to-julian-day</td>
<td>70</td>
</tr>
<tr>
<td>multiple-parents? on CLASS</td>
<td>98</td>
</tr>
<tr>
<td>multiple-parents? on MODULE</td>
<td>99</td>
</tr>
<tr>
<td>multiple-parents? on WORLD</td>
<td>99</td>
</tr>
<tr>
<td>mutable-string-to-string</td>
<td>35</td>
</tr>
</tbody>
</table>

**N**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>name-to-string</td>
<td>99</td>
</tr>
<tr>
<td>native-read-line</td>
<td>99</td>
</tr>
<tr>
<td>next? on ALL-PURPOSE-ITERATOR</td>
<td>60</td>
</tr>
<tr>
<td>next? on CONS-ITERATOR</td>
<td>40</td>
</tr>
<tr>
<td>next? on KV-LIST-ITERATOR</td>
<td>49</td>
</tr>
<tr>
<td>next? on LIST-ITERATOR</td>
<td>44</td>
</tr>
<tr>
<td>next? on MEMOIZABLE-ITERATOR</td>
<td>99</td>
</tr>
<tr>
<td>next? on PROPERTY-LIST-ITERATOR</td>
<td>47</td>
</tr>
<tr>
<td>nil?</td>
<td>36</td>
</tr>
<tr>
<td>no_duplicates? on COLLECTION</td>
<td>99</td>
</tr>
<tr>
<td>non-empty? on CONS</td>
<td>36</td>
</tr>
<tr>
<td>non-empty? on KEY-VALUE-LIST</td>
<td>48</td>
</tr>
<tr>
<td>non-empty? on KEY-VALUE-MAP</td>
<td>57</td>
</tr>
<tr>
<td>non-empty? on LIST</td>
<td>42</td>
</tr>
<tr>
<td>non-empty? on PROPERTY-LIST</td>
<td>46</td>
</tr>
<tr>
<td>non-empty? on STELLA-HASH-TABLE</td>
<td>54</td>
</tr>
<tr>
<td>non-empty? on STRING</td>
<td>31</td>
</tr>
<tr>
<td>non-empty? on STRING-WRAPPER</td>
<td>99</td>
</tr>
<tr>
<td>non-empty? on VECTOR</td>
<td>50</td>
</tr>
<tr>
<td>non-empty? on VECTOR-SEQUENCE</td>
<td>99</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>non-matching-position</td>
<td>99</td>
</tr>
<tr>
<td>non-matching-position-helper</td>
<td>99</td>
</tr>
<tr>
<td>nth on CONS</td>
<td>37</td>
</tr>
<tr>
<td>nth on LIST</td>
<td>42</td>
</tr>
<tr>
<td>nth on MUTABLE-STRING</td>
<td>33</td>
</tr>
<tr>
<td>nth on NATIVE-VECTOR</td>
<td>99</td>
</tr>
<tr>
<td>nth on STRING</td>
<td>33</td>
</tr>
<tr>
<td>nth on VECTOR</td>
<td>50</td>
</tr>
<tr>
<td>nth-rest on CONS</td>
<td>37</td>
</tr>
<tr>
<td>null-list?</td>
<td>42</td>
</tr>
<tr>
<td>null? on ARRAY</td>
<td>99</td>
</tr>
<tr>
<td>null? on CHARACTER</td>
<td>26</td>
</tr>
<tr>
<td>null? on CODE</td>
<td>26</td>
</tr>
<tr>
<td>null? on FLOAT</td>
<td>26</td>
</tr>
<tr>
<td>null? on INTEGER</td>
<td>26</td>
</tr>
<tr>
<td>null? on LONG-INTEGER</td>
<td>99</td>
</tr>
<tr>
<td>null? on MUTABLE-STRING</td>
<td>26</td>
</tr>
<tr>
<td>null? on NATIVE-VECTOR</td>
<td>26</td>
</tr>
<tr>
<td>null? on OBJECT</td>
<td>26</td>
</tr>
<tr>
<td>null? on SECOND-CLASS-OBJECT</td>
<td>26</td>
</tr>
<tr>
<td>null? on STRING</td>
<td>26</td>
</tr>
<tr>
<td>number-less-than?</td>
<td>99</td>
</tr>
<tr>
<td>object-equal? on CONS</td>
<td>37</td>
</tr>
<tr>
<td>object-equal? on HASH-SET</td>
<td>59</td>
</tr>
<tr>
<td>object-equal? on KEY-VALUE-LIST</td>
<td>48</td>
</tr>
<tr>
<td>object-equal? on KEY-VALUE-MAP</td>
<td>57</td>
</tr>
<tr>
<td>object-equal? on LIST</td>
<td>42</td>
</tr>
<tr>
<td>object-equal? on OBJECT</td>
<td>27</td>
</tr>
<tr>
<td>object-equal? on PROPERTY-LIST</td>
<td>46</td>
</tr>
<tr>
<td>object-equal? on SET</td>
<td>46</td>
</tr>
<tr>
<td>object-equal? on STELLA-HASH-TABLE</td>
<td>54</td>
</tr>
<tr>
<td>object-equal? on VECTOR</td>
<td>50</td>
</tr>
<tr>
<td>object-equal? on VECTOR-SEQUENCE</td>
<td>100</td>
</tr>
<tr>
<td>object-equal? on WRAPPER</td>
<td>27</td>
</tr>
<tr>
<td>object-hash-code</td>
<td>54</td>
</tr>
<tr>
<td>odd?</td>
<td>28</td>
</tr>
<tr>
<td>odd? on INTEGER</td>
<td>100</td>
</tr>
<tr>
<td>odd? on LONG-INTEGER</td>
<td>100</td>
</tr>
<tr>
<td>only-if</td>
<td>100</td>
</tr>
<tr>
<td>open-input-file</td>
<td>66</td>
</tr>
<tr>
<td>open-network-stream</td>
<td>100</td>
</tr>
<tr>
<td>open-output-file</td>
<td>67</td>
</tr>
<tr>
<td>operating-system</td>
<td>74</td>
</tr>
<tr>
<td>ordered? on COLLECTION</td>
<td>100</td>
</tr>
<tr>
<td>outline-depth-exceeded?</td>
<td>100</td>
</tr>
<tr>
<td>plus?</td>
<td>28</td>
</tr>
<tr>
<td>plus? on INTEGER</td>
<td>101</td>
</tr>
<tr>
<td>plus? on LONG-INTEGER</td>
<td>101</td>
</tr>
<tr>
<td>pop on HASH-SET</td>
<td>58</td>
</tr>
<tr>
<td>pop on ITERATOR</td>
<td>59</td>
</tr>
<tr>
<td>pop on KEY-VALUE-LIST</td>
<td>49</td>
</tr>
<tr>
<td>pop on LIST</td>
<td>44</td>
</tr>
<tr>
<td>pop-world</td>
<td>65</td>
</tr>
<tr>
<td>popq</td>
<td>39</td>
</tr>
<tr>
<td>position on CONS</td>
<td>38</td>
</tr>
<tr>
<td>position on LIST</td>
<td>43</td>
</tr>
<tr>
<td>position on STRING</td>
<td>33</td>
</tr>
<tr>
<td>position on VECTOR</td>
<td>50</td>
</tr>
<tr>
<td>prepend on CONS</td>
<td>38</td>
</tr>
<tr>
<td>prepend on LIST</td>
<td>44</td>
</tr>
<tr>
<td>primary-type on OBJECT</td>
<td>101</td>
</tr>
<tr>
<td>primitive? on RELATION</td>
<td>101</td>
</tr>
<tr>
<td>print</td>
<td>101</td>
</tr>
<tr>
<td>print-exception-context</td>
<td>101</td>
</tr>
<tr>
<td>print-outline</td>
<td>101</td>
</tr>
<tr>
<td>print-recycle-lists</td>
<td>102</td>
</tr>
<tr>
<td>print-spaces</td>
<td>102</td>
</tr>
<tr>
<td>print-stella-features</td>
<td>102</td>
</tr>
<tr>
<td>print-unbound-surrogates</td>
<td>102</td>
</tr>
<tr>
<td>print-undefined-methods</td>
<td>102</td>
</tr>
<tr>
<td>print-undefined-super-classes</td>
<td>102</td>
</tr>
<tr>
<td>print-xml-expression</td>
<td>73</td>
</tr>
<tr>
<td>private-class-methods</td>
<td>102</td>
</tr>
<tr>
<td>private-class-storage-slots</td>
<td>102</td>
</tr>
<tr>
<td>private? on RELATION</td>
<td>102</td>
</tr>
<tr>
<td>Probe-file?</td>
<td>67</td>
</tr>
<tr>
<td>process-doctype</td>
<td>102</td>
</tr>
<tr>
<td>ptrans</td>
<td>102</td>
</tr>
<tr>
<td>public-class-methods</td>
<td>102</td>
</tr>
<tr>
<td>public-class-storage-slots</td>
<td>102</td>
</tr>
<tr>
<td>public-slots on CLASS</td>
<td>102</td>
</tr>
<tr>
<td>public-slots on OBJECT</td>
<td>102</td>
</tr>
<tr>
<td>public? on SLOT</td>
<td>103</td>
</tr>
<tr>
<td>push on KEY-VALUE-LIST</td>
<td>49</td>
</tr>
<tr>
<td>push on LIST</td>
<td>43</td>
</tr>
<tr>
<td>push on SET</td>
<td>45</td>
</tr>
<tr>
<td>push-world</td>
<td>65</td>
</tr>
<tr>
<td>pushf</td>
<td>103</td>
</tr>
<tr>
<td>pushq</td>
<td>39</td>
</tr>
<tr>
<td>pushq-new</td>
<td>39</td>
</tr>
<tr>
<td>qualified-stella-name?</td>
<td>103</td>
</tr>
</tbody>
</table>
**Function Index**

**R**

- random ........................................ 30
- read-character ............................... 66
- read-line ..................................... 66
- read-line2 .................................... 103
- read-s-expression ........................... 66
- read-s-expression-from-string .......... 66
- read-xml-expression ....................... 73
- reader on STORAGE-SLOT ................. 103
- regular-integer-valued? .................. 103
- relative-date-to-string ................... 69
- rem ............................................. 28
- rem on INTEGER .............................. 103
- rem on LONG-INTEGER ....................... 103
- remove on CONS .............................. 38
- remove on HASH-SET ........................ 58
- remove on LIST .............................. 43
- remove on STRING ........................... 34
- remove on VECTOR-SEQUENCE .............. 51
- remove-at on HASH-TABLE ................. 52
- remove-at on KEY-VALUE-LIST .......... 48
- remove-at on KEY-VALUE-MAP .......... 57
- remove-at on PROPERTY-LIST .......... 47
- remove-at on STELLA-HASH-TABLE .... 54
- remove-at on STRING-HASH-TABLE .... 52
- remove-configuration-property .......... 103
- remove-deleted-members on LIST .......... 43
- remove-duplicates on COLLECTION ...... 103
- remove-duplicates on CONS ............... 38
- remove-duplicates on LIST ............... 43
- remove-entry on KEY-VALUE-LIST ....... 48
- remove-if on CONS .......................... 38
- remove-if on HASH-SET ..................... 58
- remove-if on LIST .......................... 43
- replace-heap-root on HEAP ............... 103
- replace-substrings .......................... 34
- required-slots on CLASS ................. 104
- required? on STORAGE-SLOT .............. 104
- reset-stella-features ..................... 104
- reset-xml-hash-tables ..................... 74
- resize-vector ................................ 51
- rest on LIST .................................. 42
- rest on STRING .............................. 33
- reverse on CONS ............................. 38
- reverse on KEY-VALUE-LIST .............. 48
- reverse on LIST .............................. 43
- reverse-interval ............................ 104
- rotate-hash-code ........................... 56
- round ......................................... 29
- run-hooks ................................. 104
- running-as-lisp? ......................... 104
- running-in-language ....................... 104
- running-system-information ............. 104

**S**

- safe-equal-hash-code ...................... 104
- safe-hash-code ............................. 104
- safe-import-surrogate .......................... 62
- safe-import-symbol .......................... 61
- safe-lookup-slot ............................ 104
- safe-tree-size ................................ 39
- safety ....................................... 104
- save-configuration-file .................... 104
- save-configuration-value .................. 104
- search-cons-tree-with-filter? .......... 104
- search-cons-tree? ......................... 39
- search-for-object ........................... 105
- second on CONS .............................. 37
- second on LIST .............................. 42
- second on MUTABLE-STRING ............... 33
- second on STRING ........................... 33
- second on VECTOR ........................... 50
- seed-random-number-generator .......... 105
- sequence ..................................... 105
- set .......................................... 46
- set-call-log-break-point ................. 105
- set-configuration-property ............... 105
- set-current-date on DECODED-DATE-TIME .. 105
- set-current-date-time on DECODED-DATE-TIME ...................... 105
- set-current-time on DECODED-DATE-TIME ...................... 105
- set-global-value ......................... 105
- set-log-level ............................... 105
- set-logging-parameters .................... 105
- set-optimization-levels ................... 105
- set-stella-feature .......................... 105
- set-translator-output-language .......... 106
- set? ........................................... 106
- shadowed-symbol? ........................... 106
- shift-right on INTEGER .................... 106
- shift-right on LONG-INTEGER ............. 106
- signal ....................................... 106
- signal-read-error ........................... 106
- sin ........................................... 29
- sleep ......................................... 71
- sort on CONS .................................. 40
- sort on HEAP .................................. 106
- sort on LIST .................................. 44
- sort on VECTOR .............................. 106
- sort-tuples on CONS ....................... 40
- split-string .................................. 106
- sqrt .......................................... 29
- start-function-call-logging .............. 106
- starts-with? ................................ 107
- stella-collection? ......................... 107
- stella-information ......................... 107
- stella-object? ................................ 107
- stella-version-string ...................... 107
- stellafy ..................................... 107
- stellify ..................................... 107
- stop-function-call-logging ............... 107
- stream-position on INPUT-FILE-STREAM ...... 107
Function Index 123

stream-position on OUTPUT-FILE-STREAM ... 107
stream-position-setter on INPUT-FILE-STREAM
stream-position-setter on OUTPUT-FILE-STREAM ... 107
stream-to-string on INPUT-STREAM 107
string-capitalize 34
string-compare 32
string-downcase 34
string-eq? 31
string-equal? 31
string-greater-equal? 32
string-greater? 32
string-less-equal? 32
string-less? 32
string-search 34
string-to-calendar-date 69
string-to-calendar-date-with-default 108
string-to-decoded-date-time 108
string-to-float 30, 35
string-to-integer 30, 35
string-to-mutable-string 35
string-to-surrogate 108
string-to-time-duration 108
string-upcase 34
string< 32
string<= 32
string?> 32
string=> 32
stringify 36
stringify-in-module 36
subclass-of? 108
subsequence on MUTABLE-STRING 108
subsequence on STRING 34
subset? on CONS 41
subset? on HASH-SET 58
subset? on LIST 45
substitute on CONS 38
substitute on HASH-SET 58
substitute on LIST 44
substitute on MUTABLE-STRING 34
substitute on SET 45
substitute on STRING 34
substitute-characters on MUTABLE-STRING
substitute-characters on STRING 108
substitute-cons-tree 39
subtract on CONS 41
subtract on HASH-SET 59
subtract on LIST 45
subtype-of? 108
super-classes on CLASS 108
surrogate-name? 109
surrogatify on STRING 109
surrogatify on SURROGATE 109
surrogatify on SYMBOL 109
sweep on OBJECT 109
symbol-plist 62
symbol-property 62
symbol-to-surrogate 109
symbol-to-type 109
symbol-value 62
symbolize 62
system-default-value on SLOT 109
system-default-value on STORAGE-SLOT 109
system-loaded-or-started-up? 109
system-loaded? 109

tan 29
terminate-program 109
third on CONS 37
third on LIST 42
third on MUTABLE-STRING 33
third on STRING 33
third on VECTOR 50
ticktock-difference 71
ticktock-resolution 71
time-add 70
time-divide 70
time-duration on TIME-DURATION 109
time-duration-to-string on TIME-DURATION 109
time-multiply 70
time-subtract 71
time-zone-format60 109
toggle-output-language 109
tokenize-string 110	trace-if 110
translate-logical-pathname 68
translate-system 110
translate-to-common-lisp? 110
translate-to-cpp? 110
translate-to-java? 111
tree-size 39
tree? 26
truncatem 111
try-to-evaluate 111
two-argument-least-common-superclass 111
two-argument-least-common-supertype 111
type on SLOT 111
type-specifier on SLOT 111
type-to-sym 111
type-to-symbol 111
type-to-wrapped-type on SURROGATE 111
typecase 10
unbound-surrogates 111
unbump-log-indent 111
unescape-html-string 111
unescape-url-string 111
unintern-surrogate 61
unintern-symbol 60
union on CONS 41
union on HASH-SET 59
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>union on LIST</td>
<td>45</td>
</tr>
<tr>
<td>unless</td>
<td>9</td>
</tr>
<tr>
<td>unread-character</td>
<td>66</td>
</tr>
<tr>
<td>unset-stella-feature</td>
<td>111</td>
</tr>
<tr>
<td>unsigned-shift-right-by-1 on INTEGER</td>
<td>112</td>
</tr>
<tr>
<td>unsigned-shift-right-by-1 on LONG-INTEGER</td>
<td>112</td>
</tr>
<tr>
<td>unstringify</td>
<td>36</td>
</tr>
<tr>
<td>unstringify-in-module</td>
<td>36</td>
</tr>
<tr>
<td>unstringify-stella-source</td>
<td>112</td>
</tr>
<tr>
<td>unwrap-boolean</td>
<td>112</td>
</tr>
<tr>
<td>unwrap-character</td>
<td>31</td>
</tr>
<tr>
<td>unwrap-float</td>
<td>30</td>
</tr>
<tr>
<td>unwrap-function-code</td>
<td>112</td>
</tr>
<tr>
<td>unwrap-integer</td>
<td>30</td>
</tr>
<tr>
<td>unwrap-long-integer</td>
<td>112</td>
</tr>
<tr>
<td>unwrap-method-code</td>
<td>112</td>
</tr>
<tr>
<td>unwrap-mutable-string</td>
<td>35</td>
</tr>
<tr>
<td>unwrap-string</td>
<td>35</td>
</tr>
<tr>
<td>upper-case-character?</td>
<td>31</td>
</tr>
</tbody>
</table>

**V**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>value-setter</td>
<td>112</td>
</tr>
<tr>
<td>ABSTRACT-DICTIONARY-ITERATOR</td>
<td>112</td>
</tr>
<tr>
<td>vector</td>
<td>50</td>
</tr>
<tr>
<td>visible-modules</td>
<td>65</td>
</tr>
<tr>
<td>visible-surrogate?</td>
<td>61</td>
</tr>
<tr>
<td>visible-symbol?</td>
<td>60</td>
</tr>
</tbody>
</table>

**W**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>warn</td>
<td>112</td>
</tr>
<tr>
<td>when</td>
<td>8</td>
</tr>
<tr>
<td>white-space-character?</td>
<td>31</td>
</tr>
<tr>
<td>with-input-file</td>
<td>67</td>
</tr>
<tr>
<td>with-network-stream</td>
<td>112</td>
</tr>
<tr>
<td>with-output-file</td>
<td>67</td>
</tr>
<tr>
<td>with-permanent-objects</td>
<td>112</td>
</tr>
<tr>
<td>with-system-definition</td>
<td>113</td>
</tr>
<tr>
<td>with-transient-objects</td>
<td>113</td>
</tr>
<tr>
<td>within-context</td>
<td>62</td>
</tr>
<tr>
<td>within-module</td>
<td>65</td>
</tr>
<tr>
<td>within-world</td>
<td>65</td>
</tr>
<tr>
<td>wrap-boolean</td>
<td>113</td>
</tr>
<tr>
<td>wrap-character</td>
<td>31</td>
</tr>
<tr>
<td>wrap-float</td>
<td>30</td>
</tr>
<tr>
<td>wrap-function-code</td>
<td>113</td>
</tr>
<tr>
<td>wrap-integer</td>
<td>30</td>
</tr>
<tr>
<td>wrap-integer-value</td>
<td>113</td>
</tr>
<tr>
<td>wrap-long-integer</td>
<td>113</td>
</tr>
<tr>
<td>wrap-method-code</td>
<td>113</td>
</tr>
<tr>
<td>wrap-mutable-string</td>
<td>35</td>
</tr>
<tr>
<td>wrap-string</td>
<td>35</td>
</tr>
<tr>
<td>wrapped-type-to-type</td>
<td>113</td>
</tr>
<tr>
<td>wrapper-value-type</td>
<td>113</td>
</tr>
<tr>
<td>write-html-escaping-url-special-characters</td>
<td>113</td>
</tr>
<tr>
<td>writer on STORAGE-SLOT</td>
<td>113</td>
</tr>
</tbody>
</table>

**X**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml-attribute-match? on XML-ATTRIBUTE</td>
<td>72</td>
</tr>
<tr>
<td>xml-attribute-match? on XML-GLOBAL-ATTRIBUTE</td>
<td>72</td>
</tr>
<tr>
<td>xml-attribute-match? on XML-LOCAL-ATTRIBUTE</td>
<td>72</td>
</tr>
<tr>
<td>xml-attribute?</td>
<td>72</td>
</tr>
<tr>
<td>xml-cdata-form?</td>
<td>72</td>
</tr>
<tr>
<td>xml-cdata?</td>
<td>72</td>
</tr>
<tr>
<td>xml-declaration-form?</td>
<td>113</td>
</tr>
<tr>
<td>xml-declaration?</td>
<td>72</td>
</tr>
<tr>
<td>xml-doctype-form?</td>
<td>113</td>
</tr>
<tr>
<td>xml-element-form?</td>
<td>114</td>
</tr>
<tr>
<td>xml-element-match? on XML-ELEMENT</td>
<td>72</td>
</tr>
<tr>
<td>xml-element?</td>
<td>72</td>
</tr>
<tr>
<td>xml-expressions</td>
<td>73</td>
</tr>
<tr>
<td>xml-local-attribute-match?</td>
<td>73</td>
</tr>
<tr>
<td>xml-lookup-attribute</td>
<td>73</td>
</tr>
<tr>
<td>xml-processing-instruction-form?</td>
<td>114</td>
</tr>
<tr>
<td>xml-processing-instruction?</td>
<td>114</td>
</tr>
<tr>
<td>xml-tag-case</td>
<td>73</td>
</tr>
<tr>
<td>xml-token-list-to-s-expression</td>
<td>114</td>
</tr>
</tbody>
</table>

**Y**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>y-or-n?</td>
<td>66</td>
</tr>
<tr>
<td>yes-or-no?</td>
<td>66</td>
</tr>
<tr>
<td>yield-define-stella-class</td>
<td>114</td>
</tr>
</tbody>
</table>

**Z**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero-pad-integer</td>
<td>114</td>
</tr>
<tr>
<td>zero?</td>
<td>28</td>
</tr>
<tr>
<td>zero? on INTEGER</td>
<td>114</td>
</tr>
<tr>
<td>zero? on LONG-INTEGER</td>
<td>114</td>
</tr>
</tbody>
</table>
Variable Index

*  
  *printpretty?* ................................. 40  
  *printprettycode?* ............................ 41  
  *printfreadably?* .............................. 40

A  
  any-key of ............................... 46, 47, 51, 52, 53  
  any-value of ............................... 36, 41, 45, 46, 47, 48, 49, 51, 52, 53  
  array-size of .............................. 49

C  
  crossover-point of .......................... 56

E  
  equal-test? of .............................. 53, 56

F  
  free-elements of ............................ 53

I  
  initial-size of .............................. 53, 56

K  
  key of  ........................................ 47

N  
  nil ............................................ 36  
  nil-list ....................................... 42

P  
  pi .............................................. 27

R  
  rest of ........................................ 36, 47

S  
  size of ........................................ 53

T  
  the-array of ................................... 49  
  the-cons-list of .............................. 41  
  the-kv-list of ................................ 48  
  the-map of ..................................... 56  
  the-plist of ................................... 46  
  the-table of ................................... 53

V  
  value of ........................................ 36, 47
Concept Index

(Index is nonexistent)