STELLA Manual

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

Version: $Revision$

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 ............................................................. 66
6.16 Files ............................................................................... 66
6.17 Dates and Times .............................................................. 69
6.18 XML Support ................................................................. 71
6.19 Miscellaneous ............................................................... 74

Function Index ................................................................. 132

Variable Index ................................................................. 133

Concept Index ................................................................. 134
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 incoceivable 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, excep-

tion 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.
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 placeholders 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

```
(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:

```
% 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):

```
% 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:

```
% 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 redefinition 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... [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:

\[(\text{condition statement...})\]

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:

\[
\begin{align*}
\text{(cond ((symbol? x) (print "x is a symbol" EOL)) ((cons? x) (print "x is a cons" EOL)) (otherwise (print "x is an object" EOL))})
\end{align*}
\]

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:

\[
\begin{align*}
\text{(setq face (choose happy? :smile :frown))}
\end{align*}
\]

case expression clause...

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

\[(\text{key statement...})\]
\[(\text{key... statement...})\]

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:

\[
\begin{align*}
\text{(case car-make ("Yugo" (setq price :cheap)) ("VW" (setq price :medium)) ("Ferrari" "Rolls Royce") (setq price :expensive)) (otherwise (setq price :unknown))}
\end{align*}
\]

typecase expression clause...

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

\[(\text{type statement...})\]
((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)
  (CONS
    (print "it is a cons"))
  ((SYMBOL KEYWORD)
    (print "it is a symbol"))
  (STANDARD-OBJECT
    (print "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
  (CONS
    (print "it is a cons with value " (first x)))
  ((SYMBOL KEYWORD)
    (print "it is a symbol with name " (symbol-name x)))
  (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:

   (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:

   (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/
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':

### 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 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.

```lisp
STELLA(4): (make-system "hello-world" :cpp)
```

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'...
    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 \texttt{factorial} from your Emacs-to-Lisp interface. Simply put your cursor somewhere inside the definition in the source buffer and evaluate it by typing \texttt{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:

\begin{verbatim}
STELLA(4): (factorial 6)
720
\end{verbatim}

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 \texttt{stella::defun} is actually bound to a Lisp macro that calls all the necessary translation machinery to convert the STELLA \texttt{defun} into Lisp code. Look at the file \texttt{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 \texttt{defun}:

1. The STELLA operator \texttt{defun} used to define STELLA functions.
2. The Lisp macro \texttt{stella::defun} that resides in the STELLA Lisp package and is only available for convenience in Lisp versions of STELLA.
3. The Lisp macro \texttt{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 \texttt{defun}. For example, if you use the Allegro CL interface you would position the cursor on the opening parenthesis of the \texttt{defun} and then type \texttt{M-M}. Any errors discovered by the STELLA translator are reported in the Lisp buffer window. The expansion will be a \texttt{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 \texttt{lptrans} macro. For example, executing

\begin{verbatim}
(lptrans
 (defun (factorial INTEGER) ((n INTEGER))
  (if (eql? n 0)
   (return 1)
   (return (* n (factorial (1- n))))))
\end{verbatim}

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

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

\begin{verbatim}
STELLA(8): (lptrans
\end{verbatim}
(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),
                        Stella.NIL))).theConsList;

   while (!(iter005 == Stella.NIL)) {
      { element = iter005.value;
        iter005 = iter005.rest;
      }
      java.lang.System.out.println(element);
   }
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{
  \textbf{STELLA(11):} (eval
    (foreach element in (list 1 2 3)
      do (print element EOL)))
  \)

This translates the loop and executes the result, which prints the wrapped numbers (hence, the \texttt{|L|} prefix) to standard output. The () 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.

\texttt{eval} is also necessary to access STELLA symbols and surrogates in the Lisp buffer. For example, to access a STELLA symbol, you can use \texttt{quote} (again, this is the STELLA \texttt{quote} not \texttt{CL:quote}):

\texttt{(eval (quote foo))}

This returns the STELLA symbol \texttt{foo}. We explicitly used \texttt{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 ‘\texttt{foo}, since it will be read directly by the STELLA reader that knows how to interpret it correctly.

\texttt{lptrans}, \texttt{cpptrans} and \texttt{jptrans} are evaluable STELLA commands that can also be evaluated by the C++ and Java version of STELLA. For example, to generate a Java trans-
lation 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:eq 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 l 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

```
(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

```
(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

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).

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

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

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

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.

mod [???]
Not yet implemented.

gcd ((x long-integer) (y long-integer)) : long-integer [Function]
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.

**exp** ((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 FLOAT)) : FLOAT
Return the arccosine of n in radians.

**atan** ((n FLOAT)) : FLOAT
Return the arc tangent of n in radians.
atan2 ((x float) (y float)) : float
Return the arc tangent of \( x / y \) in radians.

random ((n integer)) : integer
Generate a random integer in the interval [0..n-1]. The random number generator is seeded based on the current time every time STELLA starts up; however, your mileage may vary depending on the native language implementation.

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. Use parse-long-integer if the syntax of string needs to be checked for errors.

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. Use parse-float if the syntax of string needs to be checked for errors.

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.

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

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

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

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

6.3 Characters

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

code-character ((code integer)) : character
Return the character encoded by \( code \) (0 \( \leq \) code \( \leq \) 255).

digit-character? ((ch character)) : 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.

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

character-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, character-capitalize should be used instead.

character-capitalize ((ch character)) : CHARACTER
Return the capitalized character for ch. This is generally the same as the uppercase character, 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.

character-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 "".
string-compare \((x \text{ string}) \ (y \text{ string}) \ (\text{case-sensitive? boolean})\) : \text{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.

\(\text{string}<\) \((x \text{ string}) \ (y \text{ string})\) : \text{boolean}

Return true if \(x\) is lexicographically < \(y\), considering case.

\(\text{string}<=\) \((x \text{ string}) \ (y \text{ string})\) : \text{boolean}

Return true if \(x\) is lexicographically \(\leq\) \(y\), considering case.

\(\text{string}>=\) \((x \text{ string}) \ (y \text{ string})\) : \text{boolean}

Return true if \(x\) is lexicographically \(\geq\) \(y\), considering case.

\(\text{string}>\) \((x \text{ string}) \ (y \text{ string})\) : \text{boolean}

Return true if \(x\) is lexicographically > \(y\), considering case.

\(\text{string-less?}\) \((x \text{ string}) \ (y \text{ string})\) : \text{boolean}

Return true if \(x\) is lexicographically < \(y\), ignoring case.

\(\text{string-less-equal?}\) \((x \text{ string}) \ (y \text{ string})\) : \text{boolean}

Return true if \(x\) is lexicographically \(\leq\) \(y\), ignoring case.

\(\text{string-greater-equal?}\) \((x \text{ string}) \ (y \text{ string})\) : \text{boolean}

Return true if \(x\) is lexicographically \(\geq\) \(y\), ignoring case.

\(\text{string-greater?}\) \((x \text{ string}) \ (y \text{ string})\) : \text{boolean}

Return true if \(x\) is lexicographically > \(y\), ignoring case.

\(\text{all-upper-case-string?}\) \((s \text{ string})\) : \text{boolean}

Return TRUE if all letters in \(s\) are upper case.

\(\text{all-lower-case-string?}\) \((s \text{ string})\) : \text{boolean}

Return TRUE if all letters in \(s\) are lower case.

\text{make-string} \((\text{size integer}) \ (\text{initchar character})\) : \text{string}

Return a new string filled with \text{size} initchars.

\text{make-mutable-string} \((\text{size integer}) \ (\text{initchar character})\) : \text{mutable-string}

Return a new mutable string filled with \text{size} initchars.

\text{make-raw-mutable-string} \((\text{size integer})\) : \text{mutable-string}

Return a new uninitialized mutable string of \text{size}.

\text{first} \((\text{self})\) : \text{character}

Return the first character of \(\text{self}\).

\text{first} \((\text{self})\) : \text{character}

Return the first character of \(\text{self}\) (settable via \text{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. The comparison is exact.

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.
Chapter 6: Library Functions

**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)) :

MUTABLE-STRING-WRAPPER

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)) :

MUTABLE-STRING

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. Use parse-long-integer if the syntax of string needs to be checked for errors.

**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. Use parse-float if the syntax of string needs to be checked for errors.
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.

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

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.

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

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

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

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.

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.
object-equal? (tree1 (tree2 object)) : BOOLEAN
   [Method on CONS]
   Return true iff the cons trees tree1 and tree2 are structurally equivalent. Uses equal? to test equality of subtrees.

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

cons ((value OBJECT) (rest CONS)) : CONS
   [Function]
   Return a cons record that points to value and rest.

first (self) : (LIKE (ANY-VALUE self))
   [Method on CONS]
   Return the first element of self. The first element of self can be set with setf. Note that (first NIL) = null.

second (self) : (LIKE (ANY-VALUE self))
   [Method on CONS]
   Return the second element of self. The second element of self can be set with setf. Note that (second NIL) = null.

third (self) : (LIKE (ANY-VALUE self))
   [Method on CONS]
   Return the third element of self. The third element of self can be set with setf. Note that (third NIL) = null.

fourth (self) : (LIKE (ANY-VALUE self))
   [Method on CONS]
   Return the fourth element of self. The fourth element of self can be set with setf. Note that (fourth NIL) = null.

fifth (self) : (LIKE (ANY-VALUE self))
   [Method on CONS]
   Return the fifth element of self. The fifth element of self can be set with setf. Note, that (fifth NIL) = null.

nth (self (position INTEGER)) : (LIKE (ANY-VALUE self))
   [Method on CONS]
   Return the element of self at position. The nth element of self can be set with setf. Note, that (nth NIL <pos>) = null.

nth-rest (self (position INTEGER)) : (LIKE self)
   [Method on CONS]
   Apply rest position times to self.

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

but-last (self) : (ITERATOR OF (LIKE (ANY-VALUE self)))
   [Method on CONS]
   Generate all but the last element of the cons list self.

last-cons ((self CONS)) : (CONS OF (LIKE (ANY-VALUE self)))
   [Function]
   Return the last cons of self.

length (self) : INTEGER
   [Method on CONS]
   Return the length of the CONS list self.

member? (self (object OBJECT)) : BOOLEAN
   [Method on CONS]
   Return true iff object is a member of the cons list self (uses an 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 nil, insure 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)
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:

```lisp
(match-cons-tree (quote (a (b (a d) e) (a d) f g))
(quote (a (?x ?y ?) ?y ? g))
NULL)
=> |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 if 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

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**

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**

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**

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**

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**

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**

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

Not documented.

**any-value** : OBJECT

Not documented.

**the-cons-list** : (CONS OF (LIKE (ANY-VALUE SELF)))

Not documented.

**nil-list** : LIST

Not documented.
defined-list? ((self list)) : BOOLEAN
   Return TRUE unless self is NULL or the NIL-LIST.

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

empty? (self) : BOOLEAN
   Return TRUE if the list self has no members.

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

object-equal? (x (y object)) : BOOLEAN
   Return TRUE iff the lists 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. Note that this is O(N) in the number of elements of self.

list (&rest (values object)) : LIST
   Return a list containing values, in order.

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

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

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

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

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

nth (self (position integer)) : (LIKE (ANY-VALUE SELF))
   Return the nth item in the list self, or NULL if empty.

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

last (self) : (LIKE (ANY-VALUE SELF))
   Return the last element of self.

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

length (self) : INTEGER
   Not documented.
member? (self object object) : BOOLEAN
Return TRUE iff object is a member of the list self (uses an eql? test).

memb? (self object (like (any-value self))) : 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 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 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))) : 
Add value to the front of the list self.

push (self value (like (any-value self))) : 
Add value to the front of the list self.

insert-new (self value (like (any-value self))) : 
Add value to the front of the list self unless its already a member.

insert-last (self value (like (any-value self))) : 
Insert value as the last entry in the list self.

reverse (self) : (like self)
Reverse the members of self (in place).

remove (self value (like (any-value self))) : (like self)
Destructively remove all entries in self that match value.

remove-duplicates (self) : (like self)
Destructively remove duplicates from self and return the result. Preserves the original order of the remaining members.

remove-deleted-members (self) : (like self)
Not documented.

remove-if (self test? function-code) : (like self)
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))
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)
Destructively replace each appearance of outValue by inValue in the list self.

**concatenate** (list1 (list2 list) &rest (otherLists list)) : 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)
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)))
Return a copy of the list self. The conses in the copy are freshly allocated.

**clear** (self) :
Make self an empty list.

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

**allocate-iterator** (self) : (list-iterator of (like (any-value self)))
Not documented.

**next?** (self) : BOOLEAN
Not documented.

**sort** (self (predicate function-code)) : (list 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. 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
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.
Chapter 6: Library Functions

**equivalent-sets?** (self (otherList list)) : BOOLEAN  
[Method on LIST]  
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  
[Method on 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  
[Method on 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  
[Method on LIST]  
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 list)) : LIST  
[Method on 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  
[Class]  
Not documented.

**any-value** : OBJECT  
[Class Parameter of ]  
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.
Chapter 6: Library Functions

**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)))))

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)

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

**Clear** (self)

Make `self` an empty property list.

**Allocate-iterator** (self)

Not documented.

**Next?** (self)

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.

**Copy-KV-Cons-List** ((kvconslist KV-Cons))

Return a copy of the cons list `consList`.

**Key-Value-List** : DICTIONARY

Not documented.
the-kv-list : KV-CONS
           [Slot of ]
           Not documented.

empty? (self) : BOOLEAN
           [Method on KEY-VALUE-LIST]
           Not documented.

non-empty? (self) : BOOLEAN
           [Method on KEY-VALUE-LIST]
           Not documented.

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

equal-hash-code (self) : INTEGER
           [Method on KEY-VALUE-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 KEY-VALUE-LIST]
           Not documented.

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

reverse (self) : (LIKE SELF)
           [Method on KEY-VALUE-LIST]
           Destructively reverse the members of the list self.

insert-at (self (key (LIKE (ANY-KEY SELF))))
           (value (LIKE (ANY-VALUE SELF)))) :
           [Method on KEY-VALUE-LIST]
           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)))) :
           [Method on KEY-VALUE-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.

insert-entry (self (key (LIKE (ANY-KEY SELF))))
           (value (LIKE (ANY-VALUE SELF)))) :
           [Method on KEY-VALUE-LIST]
           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)))) :
           [Method on KEY-VALUE-LIST]
           Remove the entry that matches <'key', value>. Assumes that more than one entry can match key.

push (self (value KV-CONS)) :
           [Method on KEY-VALUE-LIST]
           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**

```
(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**

```
pop (self) : (like (any-value 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**

```
copy (self) : (like self) : 
```

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

**clear**

```
clear (self) : 
```

Make self an empty dictionary.

**consify**

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

Return a list of key-value pairs in self.

**allocate-iterator**

```
allocate-iterator (self) : (kv-list-iterator of
                       (like (any-key self)) (like (any-value self))) 
```

Not documented.

**next?**

```
next? (self) : BOOLEAN : 
```

Not documented.

### 6.8 Vectors

**VECTOR**

```
VECTOR : SEQUENCE
```

Not documented.

**any-value**

```
any-value : OBJECT
```

Not documented.

**array-size**

```
array-size : INTEGER
```

Not documented.

**the-array**

```
the-array : (native-vector of (like (any-value self)))
```

Not documented.

**EXTENSIBLE-VECTOR**

```
EXTENSIBLE-VECTOR : VECTOR
```

Not documented.

**empty?**

```
empty? (self) : BOOLEAN
```

Return true if self has length 0.

**non-empty?**

```
on-empty? (self) : BOOLEAN
```

Return true if self has length > 0.

**object-equal?**

```
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)) :
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 (self (offset INTEGER))
(value (LIKE (ANY-VALUE SELF)))) :
Not documented.
Chapter 6: Library Functions

copy (self) : (VECTOR OF (LIKE (ANY-VALUE SELF)))  [Method on VECTOR]
    Return a copy of the vector self.

clear (self) :  [Method on VECTOR]
    Not documented.

resize-vector ((self VECTOR) (size INTEGER)) :  [Function]
    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 (self) : (CONS OF (LIKE (ANY-VALUE SELF)))  [Method on VECTOR]
    Return a list of elements in self.

insert-at (self (offset INTEGER) (value (LIKE (ANY-VALUE SELF)))) :  [Method on EXTENSIBLE-VECTOR]
    Not documented.

insert (self (value (LIKE (ANY-VALUE SELF)))) :  [Method on VECTOR-SEQUENCE]
    Append value to the END of the sequence self. Resize the array if necessary.

remove (self (value (LIKE (ANY-VALUE SELF)))) :  [Method on VECTOR-SEQUENCE]
    VECTOR-SEQUENCE
    Remove value from the sequence self, and left shift the values after it to close the gap.

length (self) : INTEGER  [Method on VECTOR-SEQUENCE]
    Not documented.

6.9 Hash Tables

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

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

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

STRING-HASH-TABLE : ABSTRACT-HASH-TABLE  [Class]
    Not documented.

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

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

STRING-TO-INTEGER-HASH-TABLE : ABSTRACT-HASH-TABLE  [Class]
    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)) :
Not documented.
Chapter 6: Library Functions

lookup (self (key integer)) : (like (any-value self))
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 (any-value self))
: [Method on STELLA-HASH-TABLE]
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.
**Chapter 6: Library Functions**

**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

Return a hash code for self (can be negative). Two objects that are eql? are guaranteed to generate the same hash code. Two objects that are not eql? do not necessarily generate different hash codes.

hash-code (self) : integer

Not documented.

hash-code (self) : integer

Not documented.

hash-code (self) : integer

Not documented.

hash-code (self) : integer

Not documented.

hash-code (self) : integer

Not documented.

hash-code (self) : integer

Not documented.

hash-code (self) : integer

Not documented.

hash-code (self) : integer

Not documented.

hash-code (self) : integer

Not documented.

equal-hash-code (self) : integer

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

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.
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 eql? 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 eql? 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 eql? (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 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 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 eql? test by default or equal? if equal-test? of self is TRUE.
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) : [Method on KEY-VALUE-MAP] 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 (&rest (values object)) : hash-set  [Function]
   Return an eql? HASH-SET containing values.

member? (self (object object)) : BOOLEAN  [Method on hash-set]
   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)))) :  [Method on hash-set]
   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)))) : (like self)  [Method on hash-set]
   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)) : (like self)  [Method on hash-set]
   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) : (like (any-value self))  [Method on hash-set]
   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)) : (like self)  [Method on hash-set]
   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) : (like self)  [Method on hash-set]
   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) : (cons of (like (any-value self)))  [Method on hash-set]
   Collect all entries of self into a cons list and return the result.

subset? (self (otherSet hash-set)) : BOOLEAN  [Method on hash-set]
   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  [Method on hash-set]
   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 [Method on 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 [Method on 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 [Method on 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 [Method on 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 [Method on hash-set]

Return TRUE if 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 [Method on hash-set]

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 [Method on iterator]

Return TRUE if the sequence represented by self has no elements. Side-effect free.

member? (self (value object)) : boolean [Method on iterator]

Iterate over values of self, returning TRUE if one of them is eql to ‘value.

length (self) : integer [Method on abstract-iterator]

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)) [Method on iterator]

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) [Method on iterator]

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 [Method on 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.
Chapter 6: Library Functions
safe-import-symbol ((symbol symbol) (module module)) : symbol  [Function]
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  [Function]
Return the first surrogate with name visible from the current module.

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

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

lookup-surrogate-in-module ((name string) (module module)  [Function]
  (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)  [Function]
  (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)  [Function]
  (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  [Function]
Return true if self is visible from the current module.

lookup-visible-surrogates-in-module ((name string)  [Function]
  (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)) :  [Function]
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 ((surrogate surrogate) (module module)) :  [Function]
   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 ((name string)) :  [Function]
   Return the keyword with name if it exists.

intern-keyword ((name string)) :  [Function]
   Return a newly-created or existing keyword with name name. Storage note: a COPY
   of name is stored in the keyword

gensym ((prefix string)) : symbol
   Retun a transient symbol with a name beginning with prefix and ending with a
   globally gensym'd integer.

local-gensym ((prefix string)) : symbol
   Not documented.

symbol-plist ((symbol symbol)) : cons
   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 ((symbol symbol) (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 ((symbol symbol)) : object
   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 ((surrogate surrogate)) : symbol
   Convert surrogate into a symbol with the same name and module.

6.14 Context and Modules

get-stella-context ((pathName string) (error? boolean)) :
   [Function]
   context
   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 ((self context)) :
   [Function]
   Destroy all objects belonging to self or any of its subcontexts.

within-context ((contextForm object) &body (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)**: Change the current context to be the context context.

**change-context (contextName)**: Change the current context to be the context named contextName.

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

```
(deftemplate <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:

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

```
(defunmodule 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. Additionally, the Java translator makes all API functions synchronized.

---

**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.

**find-or-create-module** ((pathname string)) : module [Function]

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 [Command]
Spawn a new world that is a child of the current context, and change the current context to the new world.

pop-world () : CONTEXT [Command]
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)) : [Function]
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)) : [Function]
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 ((fileName string) &rest (options keyword)):
  [Function]
  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 spec-
  ifies 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 :CREATE-PATH the file will be created plus any
  non-existing directories in the directory path of fileName will also 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 ((self stream)):
  [Function]
  Close the stream self.

close-all-files ():
  [Function]
  Close all currently open file streams. Use for emergencies or for cleanup.

with-input-file ((binding cons) &body (body cons)):
  [Macro]
  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 ((binding cons) &body (body cons)):
  [Macro]
  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? ((filename file-name)):
  [Function]
  Return true if file fileName exists. Note that this does not necessarily mean that the
  file can also be read. IMPORTANT Java idiosyncrasy: if file foo/bar exists and is
  not a directory, Java will also say foo/bar/ exists, which is different behavior than
  in Lisp and C++. For this reason, make sure to always use probe-directory? to test
  whether a directory exists.

file-write-date ((filename file-name)):
  [Function]
  Return the time at which file fileName was last modified or NULL if that cannot be
determined.

file-length ((filename file-name)):
  [Function]
  Return the length of file fileName in bytes or NULL if that cannot be determined.

copy-file ((fromFile file-name) (toFile file-name)):
  [Function]
  Copy file fromFile to file toFile, clobbering any data already in toFile.
delete-file ((fileName file-name)) : [Function]
   Delete the file fileName.

rename-file ((fromFile file-name) (toFile file-name)) :
   Rename the file fromFile to toFile.

make-temporary-file-name ((prefix string) (suffix string)) :
   STRING
   Return a file name of the form <prefix>NNNNNN<suffix> which is guaranteed to not refer to any existing file. A null prefix defaults to tmpfile, a null suffix defaults to the empty string. The number portion NNNNNN will correspond to a random number between 0 and 999999. If no qualifying filename can be found after 100 attempts, NULL will be returned. Note that it is possible due to multi-threading or processing that the generated filename becomes used by another thread or OS process. If necessary, this case can be handled by the caller.

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 files extension (or type) if there is any and return the result.

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

file-base-name ((file file-name)) : FILE-NAME
   Remove files 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

generate-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).

generate-local-time-zone () : FLOAT
   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-TIME
   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)) : DECODED-DATE-TIME
   Returns a decoded time object for date interpreted in timezone timezone is the num-
   ber 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.
Chapter 6: Library Functions

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

    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* [Function]

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 integer keyword* [Function]

    integer integer
    Returns the YEAR, MONTH, DAY, DAY-OF-WEEK on which the given julian-day begins at noon.

**compute-day-of-week** *((yyyy 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** *((yyyy 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]

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* [Function]

integer 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]

integer
Returns the modified Julian day during which julian-day starts at noon.

**modified-julian-day-to-julian-day** *((modified-julian-day integer)) : integer* [Function]

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
Add 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
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
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)) :
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.
Chapter 6: Library Functions

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)) : 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)) : [Function]
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)) : [Function]
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)) : [Method on DIMENSIONAL-ARRAY-MIXIN]
INTEGER
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)) : [Method on 2-DIMENSIONAL-ARRAY-MIXIN]
(like (any-value self))
Return the element of self at position [i, j].
2d-aref-address (self (i integer) (j integer)) : integer
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))
Set the element of self at position [i, j] to value and return the result.

abs (x) : 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
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-current-date-substitution
((substitution-list (key-value-list of string-wrapper string-wrapper)))
: [Function]
Fill in substitution-list with date information for the current date and time. See add-date-substitution for details.

add-date-substitution ((date calendar-date) (substitution-list (key-value-list of string-wrapper string-wrapper)))
: [Function]
Fill in substitution-list with template variable substitutions for the names YEAR, MONTH, MON, DAY, HOUR, MINUTE, SECOND, TIMEZONE, DAY-OF-WEEK, DOW with their values for date. Also, pre-formatted DATE, TIME and ISO8601 variables are set.
TIMEZONE is in the format "{+|-}hmm". MONTH is the full English month name and MON is the numeric month. DAY-OF-WEEK is an English string and DOW is the first three letters. Minutes and seconds are zero-padded.
These substitutions can be used with substitute-template-variables-in-string.

add-hook ((hookList hook-list) (hookFunction symbol)) : [Function]
Insert the function named hookFunction into hookList.

add-load-path ((path string)) : (cons of string-wrapper) [Command]
Append the directories listed in the |-separated path to the end of the STELLA load path. Return the resulting load path.
add-property-value ((property name) (value object)):

Add value to the end of property's (a string or symbol) value list in the configuration table. Coerces the current value to a list or initializes the list if it is as yet undefined. Allows incremental addition of values to list-valued properties. Note that property is evaluated and will need to be quoted if supplied as a symbol. Symbols will also be upcased if this command is run in a non-case-sensitive module.

add-trace (&rest (keywords generalized-symbol)):

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)):

Returns the first index into source, starting from start, of the first character that is not white space.

all-classes ((module module) (local? boolean)):

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 ():

Return an iterator that generates all contexts.

all-defined? (&body (forms cons)):

Evaluate each of the forms in forms, and return TRUE if none of them are NULL.

all-functions ((module module) (local? boolean)):

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)):

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)):

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 ():

Return an iterator that generates all modules.

all-public-functions ((module module) (local? boolean)):

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.
Chapter 6: Library Functions

all-public-methods ((module module) (local? boolean)) : [Function]
  (iterator of method-slot)
  Iterate over all public methods visible from module. If local?, return only methods interned in module. If module is null, return all methods interned everywhere.

all-required-systems ((system-name string)) : (cons of string-wrapper)
  Returns a CONS of all of the systems required by system-name

all-slots ((module module) (local? boolean)) : (iterator of slot)
  Iterate over all slots visible from module. If local?, return only methods interned in module. If module is null, return all methods interned everywhere.

all-subcontexts ((context context) (traversal keyword)) :
  (all-purpose-iterator of context)
  Return an iterator that generates all subcontexts of self (not including self) in the order specified by traversal (one of :preorder, :inorder, :postorder or :topdown).

all-surrogates ((module module) (local? boolean)) : (iterator of surrogate)
  Iterate over all surrogates visible from module. If local?, return only surrogates interned in module. If module is null, return all surrogates interned everywhere.

all-symbols ((module module) (local? boolean)) : (iterator of symbol)
  Iterate over all symbols visible from module. If local?, return only symbols interned in module. If module is null, return all symbols interned everywhere.

all-variables ((module module) (local? boolean)) : (iterator of global-variable)
  Iterate over all variables visible from module. If local?, return only variables bound to symbols interned in module. If module is null, return all variables defined everywhere.

allocate-cross-product-iterator ((domains (cons of cons))) :
  (cross-product-iterator)
  Allocate a cross product iterator for a list of domains.

allocate-iterator (self) : (like self) [Method on abstract-iterator]
  Iterator objects return themselves when asked for an iterator (they occupy the same position as a collection within a foreach statement).

allocate-iterator (self) : (iterator of (like (any-value self))) [Method on memoizable-iterator]
  Alias for clone-memoized-iterator.

allocation (self) : keyword [Method on storage-slot]
  Return the most specific :allocation facet, or :instance if all inherited values are NULL.

append-character (self (char character)) : [Method on buffered-string]
  Append char to the END of the string self. Resize the buffer if necessary.
append-string (self (value STRING)) : [Method on BUFFERED-STRING]
    Append value to the END of the string self. Resize the buffer if necessary.

apply ((code FUNCTION-CODE) (arguments (CONS OF OBJECT))) : [Function]
    Apply code to arguments, returning a value of type OBJECT. Currently limited to
    at most 10 arguments.

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))) : [Function]
    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.

autoload ((qualifiedName STRING) (systemName STRING)
    (cache SURROGATE) (error? BOOLEAN)) : FUNCTION-CODE
    Autoload function qualifiedName from system systemName. If it is already present
    in the system, simply return its code. If cache is defined, return its value if defined,
    otherwise, set its value to the function found. If the function failed to be defined
    by loading systemName and error? is true, raise an error. Otherwise, simply return
    NULL.

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.

blank-string? ((string STRING)) : BOOLEAN
    Return true if string is either NULL, empty, or only contains white space characters.

break-program ((message STRING)) : [Command]
    Interrupt the program and print message. Continue after confirmation with the user.
bump-log-indent () :  [Command]
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.

calendar-date-to-iso8601-string (date (timezone float) (include-timezone? boolean)) : string
Returns an ISO-8601 string representation of date adjusted for timezone. The Format is YYYY-MM-DDThh: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-MMM-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.
canonical-slot ((slot storage-slot)) : storage-slot
[Function]
Computes the slot at the base of a renaming chain for slot.

canonical-slot-name ((slot storage-slot)) : symbol
[Function]
Name of the slot at the base of a renaming chain for slot.

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. If no name is supplied, return the pre-existing value of the current context. ccc 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 the value of CL-USER::*STELLA-CASE-SENSITIVE-READ-MODE* [default = :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 file-name) (relative? boolean)) :
[Function]
Translate a Stella file to Common-Lisp. If relative?, concatenate root directory to file.

cl-translate-system ((system-name 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 (self) :
[Method on vector-sequence]
Clear self by setting its active length to zero.

clear (self) :
[Method on heap]
Clear self by setting its active length to zero.

clear (self) :
[Method on buffered-string]
Clear self by setting its active length to zero.
clear-configuration-property ((property string) (configuration configuration-table)) : object
Remove property in configuration and return the previous value. Use the global system configuration table if configuration is NULL.

clear-input ((self input-stream)) :
Clear all buffered raw and tokenized input of self.

clear-recycle-list ((list recycle-list)) :
Reset list to its empty state.

clear-recycle-lists () :
Reset all currently active recycle lists to their empty state.

clear-system ((name string)) :
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 () :
Disable all tracing previously enabled with add-trace.

clone-memoized-iterator ((self memoizable-iterator)) :
(iterator of (like (any-value self)))
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 ((code object)) : object
Convenience macro to splice Lisp expressions into STELLA expressions without up-setting 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
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-option-value ((value object) (type type)) : object
Coerce value to type. Return NULL if not possible.

coerce-to-boolean ((object object)) : boolean-wrapper
Return the boolean object represented by object. Return NULL if coercion is not possible.

coerce-to-float ((object object)) : float
Coerce number to a float value or NULL if not possible.
coerce-to-hash-set ((self object) (equalTest? boolean)) : [Function]
  HASH-SET
  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-string ((object object)) : STRING [Function]
  Coerce object into a string. If no standard coercion is possible, simply stringify object.

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.

coerce-value-to-boolean ((value object) (error? boolean)) : [Function]
  BOOLEAN-WRAPPER
  Return the boolean object represented by value. Return NULL if coercion is not possible or raise an error if error? is TRUE.

coerce-value-to-float ((value object) (error? boolean)) : FLOAT [Function]
  Coerce value to a float value if possible, return NULL otherwise or raise an error if error? is true.

coerce-value-to-string ((value object) (error? boolean)) : [Function]
  STRING
  Coerce value into a string if possible, return NULL otherwise or raise an error if error? is true.

coerce-value-to-type ((value object) (type type) (error? boolean)) : OBJECT [Function]
  Coerce value to type. Return NULL if not possible or raise an error if error? is TRUE.

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.

compare-strings ((x string) (y string) (collation keyword)) : [Function]
  INTEGER
  Compare x and y and return -1, 0, or 1, depending on whether x is less than, equal, or greater than y relative to collation. Currently supported values for collation are
:ascii-case-sensitive, :ascii-case-insensitive :ascii-case-normalized. The first two correspond to string-compare called with the appropriate third argument. :ascii-case-normalized calls the function string-compare-case-normalized (which see).

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
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 [Function]
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.

compute-system-root-directories ((system system-definition)) : [Function]
Compute any root directories for sources, native and Lisp binary directories in case they were not explicitly specified in systems definition. Raises an error in case any of those root directories is missing and can’t be computed.

concatenate-file-names ((file file-name) &rest (otherFiles file-name)) : file-name
Intelligently join file and otherFiles into a concatenated path where each component is separated by a single separator (similar to Python’s os.path.join). Keeps the result a logical pathname as long as possible, but converts to physical as soon as any of otherFiles contain a physical directory separator.

config-file-option-handler ((option cmd-line-option) (value object)) : [Function]
Load the configuration file value. This will modify currently set system properties defined in value with new values but leave all other currently set properties as they are.
configure-stella ((file file-name)) : [Command]
Perform STELLA run-time configuration. If supplied, load the configuration file file
first which should be supplied with a physical pathname.

consify (self) : CONS [Method on OBJECT]
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 [Macro]
Signal error message, placing non-string arguments in quotes.

copy (self) : (VECTOR-SEQUENCE OF (LIKE (ANY-VALUE self))) [Method on VECTOR-SEQUENCE]
Return a copy of the vector sequence self.

copy (self) : (CUSTOM-VECTOR-SEQUENCE OF (LIKE (ANY-VALUE self))) [Method on CUSTOM-VECTOR-SEQUENCE]
Return a copy of the vector sequence self.

copy (self) : (HEAP OF (LIKE (ANY-VALUE self))) [Method on HEAP]
Return a copy of the heap self.

copy (string) : MUTABLE-STRING [Method on MUTABLE-STRING]
Return a copy of string.

copy-stream-to-stream ((in input-stream) (out output-stream)) : [Function]
Copy in verbatimely to out. Does the right thing for binary data.

cpp-find-native-slot-value-offset ((self STANDARD-OBJECT) (slot STORAGE-SLOT)) : INTEGER [Function]
Dynamically determine the byte offset where the native value for slot starts in self.
This basically performs the function of the C++ macro offsetof but does it dynamically.
Of course, it relies on a working read/write-slot-value mechanism and we
currently primarily use this for the Python API. This basically writes value changes
of slot to determine a byte position where a change occurred and then restores the
original value (which can be NULL). This handles all native OBJECT and LITERAL
slots but does not handle special slots such as dynamic-slots or hardwired slots or
other non-standard types.

cpp-translate-system ((systemName string)) : [Function]
Translate the system systemName to C++.

cpptrans ((statement object)) : [N-Command]
Translate statement to C++ and print the result.

create-derived-list ((self list)) : LIST [Function]
Create a new list object with the same type as self.
create-directories ((directory string)) :  
[Command]
If directory does not yet exist, create it and any of its parents that do not yet exist. If directory already does exist, this simply is a no-op.

create-object ((type TYPE) &rest (initial-value-pairs OBJECT)) :  
[Function]
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, create-object is much slower than new; therefore, it should only be used if type cannot be known at translation time.

date-to-version-string ((date CALENDAR-DATE) (granularity KEYWORD)) : STRING  
[Function]
Transform date into a version string according to granularity which is at least :days and defaults to :seconds.

deactivate-demon ((demon DEMON)) :  
[Function]
Detach demon from the location(s) specified by its internal structure.

decoded-date-time-to-iso8601-string  
(\(date\)) : STRING  
[Method on DECODED-DATE-TIME]
Returns an ISO-8601 string representation of 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.

decompose-namestring ((namestring STRING) &rest (options OBJECT)) : (CONS OF STRING-WRAPPER)  
[Function]
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 "- _ "

DECOMPOSE-NAMESTRING returns a cons of STRING-WRAPPERS that are the decomposition of the input STRING. The arguments are used as follows: namestring is the input 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 _.
decompose-namestring-full ((namestring string) (break-on-cap keyword) (break-on-number keyword) (break-on-separators string)) : (cons of string-wrapper)
Non-keyword version of decompose-namestring, which will probably be easier to use when called from non-Lisp languages.

default-cmd-line-option-handler ((option cmd-line-option) (value object)) :
Default handler that tries to set a system property based on option and value.

default-form (self) : object
Returns the current value of default expression when the slot has not been assigned a value.

defdemon ((name string-wrapper) (parameterstree cons) &body (optionsandbody cons)) : object
Define a demon name and attach it to a class or slot.

define-demon ((name string) &rest (options object)) : demon

define-global-variable-object ((definition string) (nativeObject native-object-pointer)) :
Define a STELLA global variable object based on definition, and initialize its variable-native-object slot to nativeObject (which is primarily needed for C++).

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-global-variable-from-stringified-source ((stringifiedSource string)) :
Define a stella global variable using a parse tree derived from stringifiedSource.
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)) :

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 strings and lists of strings; the latter defines exploded paths to files in subdirectories. Optional options are: :data-files – a list of files like the :files keyword, which contain data or other content that should not be processed, but instead copied verbatim to the native directory :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)) :

Print a description of an object in :verbose, :terse, or :source modes.
describe-object (self (stream output-stream) (mode keyword)):
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):
Destroy the Stella class self. Unfinalize its subclasses (if it has any).

destroy-class-and-subclasses ((self class)):
Destroy the Stella class self and all its subclasses.

destructure-defmethod-tree ((method-tree cons) (options-table key-value-list)):
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 pa-
rameter and body trees are destructively removed from method-tree.

dictionary ((collectionType type) &rest (alternatingkeysandvalues object)):
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):
Returns an iterator that generates all direct super classes of self.

disable-memoization ():
Enable memoization and use of memoized expression results.

disabled-stella-feature? ((feature keyword)):
Return true if the STELLA feature is currently disabled.

disjoint-sets? (self (otherList cons)):
Return true if the intersection of self and otherList is empty. This is always true if at
least one of the two sets is the empty set. Uses an eq? test and a simple quadra-
tic-time algorithm. Note that this does not check whether self and otherList actually
are sets.

div (x (y integer)):
Return the integer quotient from dividing x by y.

div (x (y 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-load-path ((path string)):
Remove the directories listed in the l-separated path from the PowerLoom load path.
drop-slot-value ((self standard-object) (slot storage-slot) (value object)):
Clear or remove value for the slot slot on self. CAUTION: Assumes that collection-valued slots are non-null.

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.

eight-bit-character? ((ch character)) : boolean
Return TRUE if ch is in the upper half of the byte set such as UTF-8 codes, etc.

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.

ensure-directories-exist ((fileName file-name)) :
Ensure all physical directories in the directory portion of fileName exist.

ensure-file-does-not-exist ((filename string) (context string)) : [Function]
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-free-space (self (size integer)) : mutable-string
Ensure self is big enough to accommodate an additional item of size.
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.

eval-in-module-option-handler ((option cmd-line-option) (value object)) :
  Interpret an –eval-in-module option. value is expected to be of the form (<module-name> <s-expression>).

eval-option-handler ((option cmd-line-option) (value object)) :
  Interpret an –eval option by evaluating value.

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
  Evaluate the expression represented by expression and return the result. This is equivalent to (evaluate (unstringify expression)).

even? (x) : boolean
  Return true if x is an even number.

even? (x) : boolean
  Return true if x is an even number.

exception-context ((e native-exception)) : string
  Print the exception context of e to a string and return the result.

exception-message ((e native-exception)) : string
  Accesses the error message of the exception e.

extension (self) : class-extension
  Return the nearest class extension that records instances of the class self.

external-id-head? ((attribute object)) : boolean
  Checks to see if this attribute is the literal marking either a PUBLIC or SYSTEM literal for an XML Elternal ID. (See 4.2.2)

fast-heap-root (self) : (like (any-value self))
  Return the root of self which is assumed to be non-empty.
file-name-without-device ((file file-name)) : file-name
Return the file name portion of file by removing any physical device components. This is just like logical-pathname-without-host but only looks for one-character device strings as used in Windows.

file-to-string ((file string)) : string
Read the content of file and return it as a string.

fill-in-date-substitution ((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 DEPRECATED. Use add-date-substitution or add-current-date-substitution instead.

finalize-classes () :
Finalize all currently unfinalized classes.

finalize-classes-and-slots () :
Finalize all currently unfinalized classes and slots.

finalize-slots () :
Finalize all currently unfinalized slots.

find-file-in-load-path ((file string) (extensions (cons of string-wrapper))) : string
Try to find file in the current load path and, if found, return its full name. If file can’t be found literally, try to find it with any of the listed extensions added. If extensions is NULL it defaults to *stella-file-extensions*, therefore, to not default to any extensions the value has to be supplied as NIL.

find-matching-prefix-length ((string1 string) (start1 integer) (end1 integer) (string2 string) (start2 integer) (end2 integer)) :
integer
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.

find-source-from-native-file-name ((nativeFileName string)) :
string
Try to find a STELLA source file based on the name and system components of nativeFileName. This will generally search the current STELLA systems path, since
the tree where a native file resides is not necessarily the one where the corresponding source file is also. This will only work for Lisp and C++ files where the basename of translated files corresponds to the source file from which they were derived. The main purpose for this is to support the Emacs Lisp/STELLA interface to find function definitions.

**find-system-definition-file** ((name string)) : file-name

Try to find an existing system definition file for a system named name in the current STELLA_SYSTEMS_PATH or built-in implicit systems path. Return NULL if no such definition file can be found. See build-stella-systems-path on where it looks.

**find-system-root-directory** (system system-definition) :

Given a system definition system try to determine the root directory of the STELLA installation containing this system based on its definition file.

**find-system-root-directory-from-file** ((sysDefFile file-name)) :

Given a system definition file sysDefFile try to determine the root directory of the STELLA installation containing this system by looking for top-level native directories.

**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 time 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-date** ((date object) (timezone object) (control string)) :

Format date just like format-date-to-stream (which see), but return the result as a string.

**format-date-to-stream** ((date object) (timezone object) (control string) (stream output-stream)) :

Perform formatted printing of date relative to timezone onto stream. date has to be either a calendar-date or a decoded-date-time object. timezone can be null to indicate the local timezone, or a float or supported timezone name. control is a format control string whose characters are printed literally, unless they are a %-code such as one of these (inspired by the codes supported by the Unix date command): %% a literal
% %a abbreviated weekday name (e.g., Sun) %A full weekday name (e.g., Sunday)
%b abbreviated month name (e.g., Jan) %B full month name (e.g., January) %d day of month (e.g., 01) %D date; same as %m/%d/%y
%H hour (00..23) %I hour (01..12) %m month (01..12) %M minute (00..59) %p
either AM or PM %r 12-hour clock time (e.g., 11:11:04 PM) %R 24-hour hour and minute; same as %H:%M
%S second (00..60) %T time; same as %H:%M:%S %y last two digits of year (00..99) %Y year %z +hhmm numeric time zone (e.g., -04:00) %:z
+hhmm numeric time zone (e.g., -04:00) %Z alphabetic time zone abbreviation (e.g., EDT) By default, numeric fields are padded with zeroes. The following optional flags may follow %: - (hyphen) do not pad the field _ (underscore) pad with spaces 0 (zero) pad with zeros ^ use upper case if possible , use lower case if possible

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 OBJECT]
Default method. Deallocate storage for self.

free (self) : [Method on ACTIVE-OBJECT]
Remove all pointers between self and other objects, and then deallocate the storage for self.

free-hash-table-values (self) : [Method on ABSTRACT-HASH-TABLE]
Call free on each value in the hash table self.

free-space (self) : INTEGER [Method on BUFFERED-STRING]
Return the amount of free space in self.

frem ((x float) (y float)) : float [Function]
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.

gc-protect-object ((object native-object-pointer)) : native-object-pointer [Function]
Protect object from garbage collection. Returns a handle that can later be passed to gc-release-object to release object from garbage collection protection.

gc-release-object ((handle native-object-pointer)) : native-object-pointer [Function]
Release the object pointed to by handle from garbage collection protection and return the protected object. This frees handle but does not necessarily mean that the object will be garbage collected, it just can be again, once all references to it disappear.
Chapter 6: Library Functions 94

**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) : [Method on CALANDAR-DATE]**

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-generic-slot-accessor (slot storage-slot class class) (warn? boolean) : FUNCTION-CODE**

Return the accessor code to read slot on an object of type class, or NULL if no such accessor exists. In that case, print a warning if warn? is TRUE. NOTE: the accessor expects canonical-slot-names (which see).

**get-global-variable-native-object (variable global-variable) : NATIVE-OBJECT-POINTER**

Return the native object of variable which can be used to access its runtime value. In Lisp this is the Lisp symbol which holds the variable value, in C++ this is the address of the variable, and in Java this is the declared Field object. For C++ native object addresses are initialized at startup time, for Lisp and Java native objects will be accessed upon first call and then cached in the slot.

**get-language-subdirectory-from-file-type (type keyword) : STRING**

Compute a language-specific native subdirectory that should be used for files of type. If type does not naturally suggest which native language it is associated with, key in on the current translation output language. Returns the empty string for unhandled types.

**get-load-path () : (CONS OF STRING-WRAPPER)**

Return the current STELLA load path.
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-log-stream ((module string)) : OUTPUT-STREAM
Return a valid log stream for module.

get-logical-host-property ((host object) (property object)) :
OBJECT
Lookup the value of property for the logical host host.

get-native-source-subdirectory-from-file-type
((type keyword)) : STRING
Compute a native source subdirectory that should be used for files of type. If type does not naturally suggest which native language it is associated with, key in on the current translation output language. Returns the empty string for unhandled types.

get-property ((property name) &rest (defaultValue object)) :
OBJECT
Lookup property (a string or symbol) in the configuration table and return its value. If it is undefined, return the optional defaultValue. Note that property is evaluated and will need to be quoted if supplied as a symbol. Symbols will also be upcased if this command is run in a non-case-sensitive module.

get-quoted-tree ((tree-name string) (modulename string)) : CONS
Return the quoted tree with name tree-name.

get-root-directory-from-file-extension ((file file-name)) :
FILE-NAME
Given a file, determine which STELLA installation root directory tree it naturally belongs to based on its extension (i.e., source, native or binary).

get-root-directory-from-file-type ((type keyword)) : STRING
Given a normalized file type, determine which STELLA installation root directory tree it naturally belongs to (i.e., source, native or binary).

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  [Method on SYMBOL]
Return a class with name class-name. If non exists, break if error?, else return null.

get-stella-class (class-name (error? boolean)) : class  [Method on STRING]
Return a class with name class-name. If none exists, break if error?, else return null.

get-system-definition ((name string) (error? boolean)) :
FUNCTION
SYSTEM-DEFINITON
Return a system named name. If no such system is currently defined, look for a system
definition file for name in standard locations and try to load it. If no matching system
could be found or loaded and error? is true, raise an error, otherwise, simply return
NULL.

get-system-last-modified-date ((system system-definition)) :
FUNCTION
CALENDAR-DATE
Compute a last-modified date for system based on its most recently modified source
file.

get-system-subdirectory-from-file-type ((type keyword)) :
FUNCTION
STRING
Compute a native system subdirectory that should be used for files of type. If type
does not naturally suggest which native language it is associated with, key in on the
current translation output language. Returns the empty string for unhandled types.

get-system-version-string ((system system-definition)) :
FUNCTION
STRING
Compute a version string for system which is either based on an explicit :version anno-
tation on systems definition, the readable value of an existing *<system>-version[-
string]*, variable or on its most recent modification date of any of its source files.

get-temp-directory () : string
FUNCTION
Return a suitable directory for temporary files. Uses the value of
stella.tempDirectory if defined; otherwise, it will use a suitable OS-
specific default. The returned directory will end in a separator for immediate
concatenation with a physical filename.

get-time (date (timezone float)) : integer  [Method on CALENDAR-DATE]
INTERGER
INTERGER
INTERGER
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
Macro
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
Macro
User-level macro to access the most recently parsed token as an integer. This assumes
correct signed integer syntax and only checks for overflows.
get-token-long-integer () : OBJECT
User-level macro to access the most recently parsed token as a long integer. This
assumes correct signed long-integer syntax and only checks for overflows.

get-xml-base-attribute-value ((expression cons)) : STRING
Return the last base url attribute in the attribute list of this element if it exists.
Otherwise NULL.

getenv ((varname string) &rest (deflt string)) : STRING
Return the value of the OS environment variable varname. If it is not defined or if
its value is the empty string and a deflt is provided, return the default value instead.

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). The root contains the minimum
element of a min-heap with < predicate.

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)) :
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-option-handler ((option cmd-line-option) (value object)) :
Print documentation about all currently registered option handlers.
help-print-outline (top (stream output-stream) (current-depth integer) (depth integer) (named? boolean)):
    Helper method for print-outline

help-print-outline (top (stream output-stream) (current-depth integer) (depth integer) (named? boolean)):
    Helper method for print-outline

help-print-outline (top (stream output-stream) (current-depth integer) (depth integer) (named? boolean)):
    Helper method for print-outline

help-print-outline (top (stream output-stream) (current-depth integer) (depth integer) (named? boolean)):
    Helper method for print-outline

hex-character-value ((c character)) : integer
    Convert the base-16 hex character c into a base-10 number.

home-module (self) : module
    Return the home module of self.

if-output-language ((language keyword) (thenForm object) (elseForm object)) : object
    Expand to thenForm if the current translator output language equals language. Otherwise, expand to elseForm. This can be used to conditionally translate Stella code.

if-stella-feature ((feature keyword) (thenForm object) (elseForm object)) : object
    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
    Ignore unused variables with NoOp setq statements.

incrementally-translate ((tree object)) : object
    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))
    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
    Print informative message, placing non-string arguments in quotes, and terminating with a newline.
initial-value (self) : OBJECT  
Return an initial value for the class self.

initial-value (self) : OBJECT  
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  
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) : [Method on HEAP]  
(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) : [Method on HEAP]  
(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., if value is greater than the minimum of self for the case of a min-heap where 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.

int-to-string ((i integer)) : STRING  
Convert i to its string representation and return the result. This is a convenience function that expects regular integers as opposed to longs which is useful in contexts where we do automatic unwrapping based on the target.

integer-length ((x long-integer)) : INTEGER  
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-length10 ((x long-integer)) : INTEGER  
Return the 1-based position of the left-most non-zero digit in the base-10 representation of x. If x is negative, we only consider its absolute value, not the sign. This effectively computes the minimum number of base-10 digits to represent x, with the exception of x=0.
**integer-to-hex-string** ((i LONG-INTEGER)) : STRING  
Convert i to a string representation in hexadecimal notation and return the result.

**integer-to-string-in-base** ((i LONG-INTEGER) (base INTEGER)) : 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  
Returns true if x is the floating point representation of an integer.

**intern-stella-name** ((name STRING)) : GENERALIZED-SYMBOL  
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)) :  
Old name for process-command-line-arguments (which see).

**isa?** ((object OBJECT) (type TYPE)) : BOOLEAN  
Return true iff object is an instance of the class named type.

**java-translate-system** ((systemName STRING)) :  
Translate the system systemName to Java.

**jptrans** ((statement OBJECT)) :  
Translate statement to C++ and print the result.

**keyword-name?** ((name STRING)) : BOOLEAN  
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)))  
Convert self into a property list with identical and identically ordered keys and values.

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

**last** (self) : (LIKE (ANY-VALUE SELF))  
Return the last item in the heap self which will be the largest or best item if self is a sorted min-heap with a < predicate.

**last** (self) : CHARACTER  
Return the last character in self.

**length** (self) : INTEGER  
Iterate over self, and count how many items there are.
**length** (*self*) : INTEGER  
Return the length of the currently filled portion of *self*.

**length** (*self*) : INTEGER  
Return the total number of elements in *self*.

**length** (*self*) : INTEGER  
Return the total number of elements in *self*.

**length-setter** (*self* (*newLength* INTEGER)) : INTEGER  
Reset the length of *self* to *newLength*. Fill in NULL characters if *newLength* is greater than the current length. This is the safe way to reset the fill pointer.

**lispify** ((*thing* UNKNOWN)) : LISP-CODE  
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  
Lispify *thing* which is assumed to be a (possibly wrapped) Stella boolean.

**list-directory-files** ((*directory* STRING)) : (CONS OF STRING.WRAPPER)  
Return all the files and sub-directories in *directory* sorted by name. Each returned file is a bare file name without a *directory* component. If a file is a directory, it will look just like a plain file. This means consumers might have to explicitly test whether a file is a directory depending on what they do. Excludes . and .. directories. Handles logical pathnames but resulting files will always use physical pathname syntax. This is mostly consistent across native languages, but some differences still exist - e.g., Lisp will convert . or .. into absolute pathnames.

**list-directory-files-recursively** ((*directory* STRING)) : (CONS OF STRING.WRAPPER)  
Just like **list-directory-files** (which see) but also recurses into subdirectories. Files at the top level of *directory* will be bare file names without a *directory* component. Files in subdirectories will be prefixed with the relative subdirectory path starting right below *directory*. The sort order is lexicographic within directories which results in a depth-first presentation order of files.

**list-modules** ((*kb-only?* BOOLEAN)) : (CONS OF MODULE)  
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)))  
Return a list of elements in *self*.

**listify** (*self*) : (LIST OF (LIKE (ANY-VALUE SELF)))  
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)**):  
[Command]  
**CONFIGURATION-TABLE**  
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-file-option-handler** (**(option cmd-line-option) (value object)**):  
[Function]  
Load the file **value** using the STELLA **load-file** command.

**load-path-option-handler** (**(option cmd-line-option) (value object)**):  
[Function]  
Modify the current file load path according to **option** and **value**.

**load-system** (**(systemName string) &rest (language&options object)**): **BOOLEAN**  
[Command]  
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.

:**recursive?** (default false): if true, perform **load-system** with the provided options on **systemName** as well as all its required systems and so on. Required systems will be processed first. Note that even without this option, any required systems that have not yet been loaded or started up will also be processed, since that is assumed when loading **systemName** and supporting modules from a definition file.

:**root-source-directory**, **:root-native-directory**, **:root-binary-directory**: if specified these directories will be used to override the respective paths provided in system definitions or computed as defaults from a system’s home location.
log-level<= ((level object) (module string)) : boolean
  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)) :
  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.

log2 ((n float)) : float
  Return the logarithm (base 2) of n.

logmsg ((module string) (logLevel object) &rest (message object)) :
  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
  Return a class with name name. Scan all visible surrogates looking for one that has a class defined for it.

lookup-class (name) : class
  Return a class with name name. Scan all visible surrogates looking for one that has a class defined for it.

lookup-class-by-qualified-name ((qualifiedName string)) : class
  Variant of string.lookup-class that can specify a starting module through a qualifiedName.

lookup-class-cpp-type-by-name ((qualifiedName string)) : string
  Return a namespace-qualified C++ translation of the class type named by qualifiedName or NULL if no such STELLA class exists.

lookup-command ((name symbol)) : method-slot
  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-command-like-function ((name symbol)) : method-slot
  Look up a function with name that can be evaluated via apply just like a command, regardless of whether it was marked as such.

lookup-configuration-property ((property string) (defaultValue object) (configuration configuration-table)) : object
  Lookup 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
  Lookup 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
Return the demon named name.

lookup-function ((functionSymbol symbol)) : FUNCTION
Return the function defined for functionSymbol, if it exists.

lookup-function-by-name ((name string)) : 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-function-by-qualified-name ((qualifiedName string)) :
METHOD-SLOT
Variant of lookup-function-by-name that can specify a starting module through a qualifiedName.

lookup-global-variable (self) : GLOBAL-VARIABLE
Return a global variable with name self.

lookup-global-variable (self) :
GLOBAL-VARIABLE
Return a global variable with name self.

lookup-global-variable (self) : GLOBAL-VARIABLE
Return a global variable with name self.

lookup-global-variable-by-name ((name string)) :
GLOBAL-VARIABLE
Return a global variable with name visible from the current module. Scan all visible symbols looking for one that has a global variable defined for it. NOTE: this is more robust than STRING.lookup-global-variable which will fail if there is a local symbol with name which shadows one higher up that is the name of a global.

lookup-global-variable-by-qualified-name
((qualifiedName string)) : GLOBAL-VARIABLE
Variant of lookup-global-variable-by-name that can specify a starting module through a qualifiedName.

lookup-local-slot ((class class) (slot-name symbol)) : SLOT
Look up a local slot with slot-name on class.

lookup-logging-parameter ((module string) (parameter keyword) (default object)) : OBJECT
Look up logging parameter for module. Use default if no value is defined.

lookup-macro ((name symbol)) : METHOD-SLOT
If name has a macro definition, return the method object holding its expander function.

lookup-slot ((class class) (slot-name symbol)) : SLOT
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-slot-and-class-by-qualified-name  
\[\text{Function}\]
\[
(\text{qualifiedname}
\text{string}) : \text{slot class}
\]
Lookup a slot via dotted notation with a possibly qualified class name. Examples: "cons.first" or "stella/cons.first". This is a variant of \text{string-to-slot} in \text{describe.ste} that actually obeys the class module if it is given. This also returns the specified class as a second value which might differ from the slot’s owner class.

lookup-slot-by-qualified-name  
\[\text{Function}\]
\[
(\text{qualifiedname}
\text{string}) : \text{slot}
\]
Lookup a slot via dotted notation with a possibly qualified class name. Examples: "cons.first" or "stella/cons.first". This is a variant of \text{string-to-slot} in \text{describe.ste} that actually obeys the class module if it is given.

lookup-stella-name  
\[\text{Function}\]
\[
(name
\text{string}) : \text{generalized-symbol}
\]
Parse \text{name} which is assumed to be the printed representation of a STELLA symbol, surrogate or keyword, and try to look it up according to its type. This is a lookup version of \text{intern-stella-name} (which see) for cases where we don’t necessarily want to create a symbol in case it doesn’t exist already.

lookup-with-default  
\[\text{Method on property-list}\]
\[
(key
(\text{like}
(any-key
self))
(default
(\text{like}
(any-value
self)))) : \text{like}
\]
\[\text{Method on key-value-list}\]
\[
(key
(\text{like}
(any-key
self))
(default
(\text{like}
(any-value
self)))) : \text{like}
\]
Lookup \text{key} in \text{self} and return the result. Return \text{default} if no value was found.

lptrans  
\[\text{N-Command}\]
\[
((\text{statement}
\text{object})) : \text{end}
\]
Translate \text{statement} to Common-Lisp and print the result.

make-file-name  
\[\text{Function}\]
\[
((\text{filePath}
\text{string})
(type
\text{keyword})
(relative?
\text{boolean})) : \text{file-name}
\]
Make an absolute file-name string from \text{filePath} with directory location and file extension determined by \text{type} and \text{relative?}. \text{type} is the main determiner for whether we are creating a source, native or binary pathname in the respective root directories, and it also controls the new extension of the resulting pathname replacing any one on \text{filePath}. If \text{relative?} is \text{TRUE}, then the respective root directory prefix is appended, plus any additional relevant language, system and source subdirectories. If \text{relative?} is \text{FALSE}, then the current directory prefix of \text{filePath} as indicated by its extension is switched to the new root directory indicated by \text{type}. In this case, only the root and language directories are inserted, but all the remaining source directory components are left the same (thus, this will generally not do the right thing for Java file names). This function takes the dynamic state of currently active systems and modules into account.
**Chapter 6: Library Functions**

**make-matching-name** ((original string) &rest (options object)) : [Function]

**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.
- **:recursive?** (default false): if true, perform **make-system** with the provided options on `systemName` as well as all its required systems and so on. Required systems will be processed first. Note that even without this option, any required systems that have not yet been loaded or started up will also be processed, since that is assumed when loading `systemName` and supporting modules from a definition file.

- **:root-source-directory**, **:root-native-directory**, **:root-binary-directory**: if specified these directories will be used to override the respective paths provided in system definitions or computed as defaults from a system’s home location.

**make-system-definition-file-name** ((name string)) : file-name

Make a canonical system definition file name for a system named `name`.

**make-temporary-file** ((prefix string) (suffix string)) : string

Variant of **make-temporary-file-name** that actually allocates the file to prevent other processes from using that name. This is still not fully thread safe - for that we would need a file lock - but maybe a bit better in avoiding collisions.

**max** (x (y integer)) : integer

Return the maximum of `x` and `y`. If either is NULL, return the other.

**max** (x (y long-integer)) : long-integer

Return the maximum of `x` and `y`. If either is NULL, return the other.
max (x (y FLOAT)) : FLOAT  
Return the maximum of x and y. If either is NULL, return the other.

max (x (y NUMBER-WRAPPER)) : NUMBER-WRAPPER  
Return the maximum of x and y. If y is NULL, return x.

member? (self (value OBJECT)) : BOOLEAN  
Iterate over values of self and return TRUE if one of them is eql? to 'value.

member? (self (object OBJECT)) : BOOLEAN  
Return true iff object is a member of the collection self.

member? (self (value OBJECT)) : BOOLEAN  
Return TRUE if value is a member of the sequence self.

memoize ((inputArgs cons) &body (body cons)) : OBJECT  
Compute the value of an expression and memoize it relative to the values of inputArgs. 
inputArgs should characterize the complete set of values upon which the computation of the result depended. Calls to memoize should be of the form 
(memoize (<arg>+) {}<option> <value>* <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 gensymed 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)) : FILE-NAME 
Parse baseFile, supply any missing components from defaults if supplied and return the result.

merge-null-fields (self (default 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 \textit{self} just has the month being non-null, then only the year
will be filled in from \textit{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.

\textbf{min} \((x \ (y \ \text{INTEGER})) : \text{INTEGER} \quad \text{[Method on INTEGER]}
\) Return the minimum of \(x\) and \(y\). If either is NULL, return the other.

\textbf{min} \((x \ (y \ \text{LONG-INTEGER})) : \text{LONG-INTEGER} \quad \text{[Method on LONG-INTEGER]}
\) Return the minimum of \(x\) and \(y\). If either is NULL, return the other.

\textbf{min} \((x \ (y \ \text{FLOAT})) : \text{FLOAT} \quad \text{[Method on FLOAT]}
\) Return the minimum of \(x\) and \(y\). If either is NULL, return the other.

\textbf{min} \((x \ (y \ \text{NUMBER.WRAPPER})) : \text{NUMBER.WRAPPER} \quad \text{[Method on NUMBER.WRAPPER]}
\) Return the minimum of \(x\) and \(y\). If \(y\) is NULL, return \(x\).

\textbf{mod} \((x \ (\text{modulus} \ \text{INTEGER})) : \text{INTEGER} \quad \text{[Method on INTEGER]}
\) True modulus. Return the result of \(x \mod \text{modulo}\). Note: In C++ and Java, \textit{mod} has
more overhead than the similar function \textit{rem}. The answers returned by \textit{mod} and \textit{rem}
are only different when the signs of \(x\) and \textit{modulo} are different.

\textbf{mod} \((x \ (\text{modulus} \ \text{LONG-INTEGER})) : \text{LONG-INTEGER} \quad \text{[Method on LONG-INTEGER]}
\) True modulus. Return the result of \(x \mod \text{modulo}\). Note: In C++ and Java, \textit{mod} has
more overhead than the similar function \textit{rem}. The answers returned by \textit{mod} and \textit{rem}
are only different when the signs of \(x\) and \textit{modulo} are different.

\textbf{multiple-parents?} \((\text{class}) : \text{BOOLEAN} \quad \text{[Method on CLASS]}
\) Return \textit{true} if \textit{class} has more than one direct superclass.

\textbf{multiple-parents?} \((\text{module}) : \text{BOOLEAN} \quad \text{[Method on MODULE]}
\) Return \textit{TRUE} if \textit{module} has more than one parent.

\textbf{multiple-parents?} \((\text{world}) : \text{BOOLEAN} \quad \text{[Method on WORLD]}
\) Return \textit{FALSE} always, since worlds never have more than one parent.

\textbf{name-to-string} \((\text{name} \ \text{OBJECT}) : \text{STRING} \quad \text{[Function]}
\) Return the string represented by \textit{name}. Return \textit{null} if \textit{name} is undefined or does
not represent a string.

\textbf{native-delete-file} \((\text{fileName} \ \text{file-name}) : \quad \text{[Function]}
\) Delete the file \textit{fileName}. This does not handle any necessary pathname translations
or error conditions.

\textbf{native-file-length} \((\text{fileName} \ \text{file-name}) : \text{LONG-INTEGER} \quad \text{[Function]}
\) Return the length of file \textit{fileName} in bytes or NULL if that cannot be determined.
This does not handle any necessary pathname translations or error conditions.

\textbf{native-file-write-date} \((\text{fileName} \ \text{file-name}) : \text{CALENDAR-DATE} \quad \text{[Function]}
\) Return the time at which file \textit{fileName} was last modified or NULL if that cannot
be determined. This does not handle any necessary pathname translations or error
conditions.
native-probe-directory? ((file-name)) : boolean  [Function]
Return true if file file-name exists and is a directory. Note that this does not necessarily mean that the directory can also be read. This does not handle any necessary pathname translations or error conditions.

native-probe-file? ((file-name)) : boolean  [Function]
Return true if file file-name exists. Note that this does not necessarily mean that the file can also be read. This does not handle any necessary pathname translations or error conditions.

native-read-line ((input-stream)) : string  [Function]
Read one line from input-stream using the native language readline algorithm and return the result. On EOF return null

native-rename-file ((from-file) (to-file)) :  [Function]
Rename the file from-file to to-file. This does not handle any necessary pathname translations or error conditions.

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-empty? (self) : boolean  [Method on BUFFERED-STRING]
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  [Function]
Helper for non-matching-position that requires end to not be null.

nth (self (position integer)) : character  [Method on BUFFERED-STRING]
Return the character in self at position.

nth (self (position integer)) : (like (any-value self))  [Method on NATIVE-VECTOR]
Return the element in self at position.
nth-setter (self (ch character) (position integer)) : character
Set the character in self at position to ch.

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 if 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)) : input-stream output-stream
[Function]
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.

parse-date-time-in-time-zone ((date-time-string string) (time-zone float) (start integer) (end integer) (error-on-mismatch? 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 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 given date and 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
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-float ((value String)) : Float
Convert a float value string into a float. Leading and trailing whitespace is allowed. Raise an error if we have illegal number syntax which makes it preferable over the naive C++ stringToFloat conversion function which just returns 0.

parse-integer ((value String)) : Integer
Convert an integer value string into an integer. Leading and trailing whitespace is allowed. Raise an error if we have illegal number syntax which makes it preferable over the naive C++ stringToInteger conversion function which just returns 0.

parse-long-integer ((value String)) : Long-Integer
Convert a long integer value string into a long integer. Leading and trailing whitespace is allowed. Raise an error if we have illegal number syntax which makes it preferable over the naive C++ stringToInteger conversion function which just returns 0.

parse-options ((options Object) (legalOptions&Types Cons) (coercionError? boolean) (allowOtherKeys? boolean)) : Property-List
Parse options, check their validity according to legalOptions&Types and return the result as a PROPERTY-LIST. legalOptions&Types has to either be NULL or a flat list of legal <keyword> <coercionType> pairs. A type specification of @IDENTITY means don’t perform any coercion. If coercionError? is TRUE, raise an error if a coercion failed. If allowOtherKeys? is TRUE options other than those specified in legalOptions&Types are allowed but won’t be coerced since we don’t know their type. A special implicit :options keyword that does not need to be declared can be used to pass in an already parsed options list from a caller, which will then be analyzed and used instead.

parse-stella-name ((name String) (enableCaseConversion? boolean)) : String String Keyword
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** \((\text{minSize} \text{ integer})\) : \text{integer}  
[Function]  
Return a hash table prime of at least \text{minSize}.

**plist-to-kvlist** \((\text{self} \text{ property-list})\) : \(\text{(key-value-list of (like \text{any-key self}) (like \text{any-value self})})\)  
[Function]  
Convert \text{self} into a key-value list with identical and identically ordered keys and values.

**plus?** \((x)\) : \text{BOOLEAN}  
[Method on \text{INTEGER}]  
Return true if \text{x} is greater than 0.

**plus?** \((x)\) : \text{BOOLEAN}  
[Method on \text{LONG-INTEGER}]  
Return true if \text{x} is greater than 0.

**pop-load-path** () : \text{STRING}  
[Command]  
Remove the first element from the STELLA load path and return the removed element.

**primary-type** \((\text{self})\) : \text{TYPE}  
[Method on \text{OBJECT}]  
Returns the primary type of \text{self}. Gets defined automatically for every non-abstract subclass of \text{OBJECT}.

**primitive?** \((\text{self})\) : \text{BOOLEAN}  
[Method on \text{RELATION}]  
Return \text{true} if \text{self} is not a defined relation.

**print** \((\text{&body} \text{ body cons})\) : \text{OBJECT}  
[Macro]  
Print arguments to the standard output stream.

**print-configuration-properties**  
\((\text{configuration} \text{ configuration-table} \text{ stream output-stream})\) :  
[Function]  
Print all properties defined in \text{configuration} to \text{stream}.

**print-exception-context**  
\((\text{e} \text{ native-exception} \text{ stream output-stream})\) :  
[Function]  
Prints system dependent information about the context of the specified exception \text{e}. For example, in Java it prints a stack trace. In Lisp, it is vendor dependent.

**print-outline**  
\((\text{thing} \text{ object} \text{ stream output-stream})\)  
[Command]  
Print an outline of \text{thing} and its subparts on \text{stream}. If \text{depth} is non-negative, only \text{depth} levels will be printed. If \text{named?} is \text{TRUE}, then only named entities will be printed.

**print-properties** () :  
[Command]  
Print all current configuration property information to standard output.
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) [Function]
Iterate over all private storage-slots attached to class.

private? (self) : boolean [Method on relation]
Return true if self is not public.

probe-directory? ((fileName file-name)) : boolean [Function]
Return true if file fileName exists and is a directory. Note that this does not necessarily mean that the directory can also be read.

process-command-line-arguments ((count integer) (arguments (array () of string)) (unhandledOptionAction keyword)) : [Function]
Interpret any command line arguments for which handlers have been registered. Leave any remaining unprocessed arguments in *unprocessed-command-line-arguments*. If any unprocessed arguments use option syntax (that is they start with a -), proceed according to unhandledOptionAction which can be one of :ignore, :warn or :error. This ensures that at any point in the option processing, *unprocessed-command-line-arguments* accurately reflects the arguments which have been either skipped or not handled yet.

process-doctype ((doctype-declaration cons)) : xml-doctype [Function]
Takes an S-Expression representing a doctype and processes into a DOCTYPE object.
ptrans ((statement object)) :  
N-Command

Translate statement to Common-Lisp and print the result.

public-class-methods ((class class)) : (iterator of method-slot)  
Function

Iterate over all private methods attached to class.

public-class-storage-slots ((class class)) : (iterator of storage-slot)  
Function

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  
Method on SLOT

True if self or one it its ancestors is marked public.

push-load-path ((path string)) : (cons of string-wrapper)  
Command

Add the directories listed in the |-separated path to the front of the STELLA load path. Return the resulting load path.

pushf ((place cons) (value object)) : object  
Macro

Push value onto the cons list place.

put-slot-value ((self standard-object) (slot storage-slot) (value object)) : object  
Function

Set or insert value for the slot slot on self. CAUTION: Assumes that collection-valued slots are non-null.

qualified-stella-name? ((name string)) : BOOLEAN  
Function

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-global-variable-value ((variable global-variable)) :  
Function

Interpreted global variable reader which reads and returns its native value which will be wrapped if necessary. This will access the value at the top of the special stack.

read-line2 ((stream input-stream)) : string keyword  
Function

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.

read-native-variable-value ((var native-object-pointer) (type type)) : object  
Function

Interpreted variable reader which reads a native code variable represented by var of type type and returns its value which will be wrapped if necessary. If var is a special
variable, this will access the current dynamic value at the top of the special stack. This relies upon type being completely accurate and bad things will happen if it is not.

```lisp
read-slot-value ((self STANDARD-OBJECT) (slot STORAGE-SLOT)) : [Function]
  OBJECT
  Read and return a (possibly wrapped) value for the slot slot on self.
```

```lisp
read-xml-expressions ((filename STRING)) : CONS [Function]
  Read all of the top-level XML expressions from filename and return them in a list.
```

```lisp
reader (self) : SYMBOL [Method on STORAGE-SLOT]
  Name of a method called to read the value of the slot self.
```

```lisp
register-cmd-line-option (&rest (options OBJECT)) :
  Register a command line option. :key identifies the name of the option which will usually start with a dash such as --3 or --eval. :key2 and :key3 can be used to supply additional options (e.g., long option formats). To supply even more keys, a list can be supplied with the :keys option. If a :property is supplied, this option simply sets or adds to the values of the specified system configuration property. If a :handler name is specified, its function will be used to interpret the values of the option. :documentation can be used to supply a documentation string which will be printed by the help-option-handler (usually bound to -?). :value-type describes what type an option value should be coerced to before assigning it to the specified configuration :property. :n-arguments describes how many arguments this option takes. This will be 0 for simple switches and can be 1 or greater than one for option handlers that need one or more arguments. :default-value defines the value to use for zero-argument :property options. If :multi-valued? is true, values of multiple occurrences of the option will be added to the specified configuration :property. :error-action can be one of :ignore, :warn or :error to specify what to do in case an error is encountered during option processing.
```

```lisp
register-property-demon ((property STRING)
  (demonName SYMBOL)) :
  Register the function named demonName as the demon for property. Demons will be run as after demons on every configuration table update. Set the property stella.test.propertyDemon to see a test demon in action.
```

```lisp
regular-integer-valued? ((x LONG-INTEGER)) : BOOLEAN [Function]
  Return true if x can be represented by a regular integer.
```

```lisp
relative-unlogicalize-pathname ((pathName STRING)) : STRING [Function]
  If pathName is a logical pathname translate it, however, remove the :root-directory prefix of the logical host. Otherwise, return pathName as is.
```

```lisp
rem (x (\(y INTEGER)) : INTEGER [Method on 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
[Method on 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
[Function]
Remove value from property in configuration and return it. Use the global system configuration table if configuration is NULL.

remove-duplicates (self) : (LIKE SELF)
[Method on COLLECTION]
Return self with duplicates removed. Preserves the original order of the remaining members.

remove-duplicates-equal (self) : (LIKE SELF)
[Method on CONS]
remove-duplicates (which see) using an equal? test. IMPORTANT: since this uses hashing to speed things up, an equal-hash-code method needs to be defined for this to work.

remove-duplicates-equal (self) : (LIKE SELF)
[Method on LIST]
remove-duplicates (which see) using an equal? test.

replace-heap-root (self (value (LIKE (ANY-VALUE SELF)))) : [Method on HEAP]
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) : (LIST OF SYMBOL)
[Method on CLASS]
Returns a list of names of required slots for self.

required? (self) : BOOLEAN
[Method on STORAGE-SLOT]
True if a value must be assigned to this slot at creation time.

reset (self) :
[Method on CROSS-PRODUCT-ITERATOR]
Reset self to its initially allocated state. Note, that this is somewhat expensive, costing almost as much as allocating the iterator.

reset-stella-features () :
[Command]
Reset STELLA features to their default settings.

resize-buffer (self (size INTEGER)) :
[Method on BUFFERED-STRING]
Change the size of self to size. If size is smaller than the current size of self, it’s buffer will be truncated.

reverse (self) : (LIKE SELF)
[Method on VECTOR-SEQUENCE]
Reverse the order of elements in the active portion of self.

reverse-interval ((lowerbound INTEGER) (upperbound INTEGER)) :
[Function]
reverse-integer-interval-iterator
Create a reverse interval object.
Chapter 6: Library Functions

root-binary-directory () : FILE-NAME
Accessor to determine the currently active binary directory root. If it is defined as an option in the currently active system action, use that. If we have an active system, use its value, otherwise use the STELLA default value.

root-native-directory () : FILE-NAME
Accessor to determine the currently active native directory root. If it is defined as an option in the currently active system action, use that. If we have an active system, use its value, otherwise use the STELLA default value.

root-source-directory () : FILE-NAME
Accessor to determine the currently active sources directory root. If it is defined as an option in the currently active system action, use that. If we have an active system, use its value, otherwise use the STELLA default value.

run-hooks ((hooklist hook-list) (argument object)) :
Run all hook functions in hooklist, applying each one to argument.

running-as-lisp? () : BOOLEAN
Return true if the executable code is a Common Lisp application.

running-in-language () : KEYWORD
Returns the keyword for the language the current implementation is running in.

running-system-information () : STRING
Returns an information string about the current running system environment.

safe-compare-strings ((x string) (y string) (collation keyword)) : INTEGER
Variant of compare-strings that also tolerates NULL values. NULL sorts after everything else in any collation.

safe-equal-hash-code ((self object)) : INTEGER
Return a hash code for self. Just like equal-hash-code - which see, but also works for NULL. equal-hash-code methods that expect to handle NULL components should use this function for recursive calls.

safe-hash-code ((self object)) : INTEGER
Return a hash code for self. Just like hash-code - which see, but also works for NULL.

safe-lookup-slot ((class class) (slot-name symbol)) : SLOT
Alias for lookup-slot. Kept for backwards compatibility.

safety ((level integer-wrapper) (test object) &body (body cons)) :
Macro
OBJECT
Signal warning message, placing non-string arguments in quotes.

save-configuration-file ((table configuration-table) (file file-name) (title string)) :
Save table as a configuration file. Uses a Java-style property file syntax.
save-configuration-value ((stream output-stream) (value object)) : [Function]
Save value to stream as a properly formatted configuration value.

search-cons-tree-with-filter? ((tree object) (value object) (filter cons)) : BOOLEAN
Return true iff the value value is embedded within the cons tree tree. Uses an eql? test. Does not descend into any cons whose first element matches an element of filter.

search-for-object ((self object) (typeref object)) : object
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 based on the current time.

seed-random-number-generator2 () : [Function]
Seeds the random number generator based on the current time and /dev/random if it is available.

sequence ((collectiontype type) &rest (values object)) : (sequence of object)
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
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-load-path ((path string)) : (cons of string-wrapper) [Command]
Set the STELLA load path to the |separated directories listed in path. Return the resulting load path.

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*)) :  
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 or file 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)) :  
Set optimization levels for the qualities *safety*, *debug*, *speed*, and *space*.

**set-property** ((*property* name) (*value* object)) :  
Set *property* (a string or symbol) in the configuration table to *value*. Note that *property* is evaluated and will need to be quoted if supplied as a symbol. Symbols will also be upcased if this command is run in a non-case-sensitive module.

**set-stella-feature** (&rest (*features* keyword)) :  
Enable all listed STELLA features.

**set-translator-output-language** ((*new-language* keyword)) :  
Set output language to *new-language*. Return previous language.

**setq?** ((*variable* symbol) (*expression* cons)) : object  
Assign *variable* the result of evaluating *expression*, and return TRUE if *expression* is not NULL else return FALSE.

**shadowed-symbol?** (**symbol** generalized-symbol) : boolean  
Return true if *symbol* is shadowed in its home module.

**shift-right** (*arg* (*count* integer)) : 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 $2^{**count}$.

**shift-right** (*arg* (*count* integer)) : 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 $2^{**count}$.

**signal** (**type** symbol) &body (**body** cons) : object  
Signal error message, placing non-string arguments in quotes.

**signal-read-error** (&body (**body** cons)) : object  
Specialized version of **signal** that throws a READ-EXCEPTION.
sort (self (predicate function-code)) : (vector of (like (any-value self)))  
[Method on 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 GENERALIZED-SYMBOL, STRING, INTEGER, and FLOAT).

sort (self (predicate function-code)) : (heap of (like (any-value self)))  
[Method on 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.

sort-objects (self (slot storage-slot)  
(predicate function-code)) : (cons of (like (any-value self)))  
[Method on CONS]  
Just like sort but assumes each element of self has a slot whose value will be used for comparison. Elements must be descendants of STANDARD OBJECT. Note that while this will work with literal-valued slots, it will cause value wrapping everytime slot is read.

sort-objects (self (slot storage-slot)  
(predicate function-code)) : (vector of (like (any-value self)))  
[Method on VECTOR]  
Just like sort but assumes each element of self has a slot whose value will be used for comparison. Elements must be descendants of STANDARD OBJECT. Note that while this will work with literal-valued slots, it will cause value wrapping everytime slot is read.

sort-tuples (self (n integer) (predicate function-code))  
[Method on VECTOR]  
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.

split-string ((input string) (separator character)) : (cons of string-wrapper)  
[Function]  
Split input into separate strings based on the separator character.

start-function-call-logging ((fileName string)) :  
[Command]  
Start function call logging to fileName.

starts-with? ((string string) (prefix string) (start integer)) :  
[Function]  
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  
[Function]
Return true if self is a native collection.

stella-information () : STRING  
[Command]
Returns information about the current Stella implementation. Useful when reporting problems.

stella-integer-to-string-in-base ((integer long-integer) (base integer)) : STRING  
[Function]
STELLA version of integer-to-string-in-base which is faster than the C++ version we have but slower than the native Java version.

stella-object? ((self object)) : BOOLEAN  
[Function]
Return true if self is a member of the STELLA class OBJECT.

stella-version-string () : STRING  
[Function]
Return a string identifying the current version of STELLA.

stellafy ((thing lisp-code) (targetType type)) : OBJECT  
[Function]
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  
[Function]
Convert a Lisp object into a STELLA object.

stop-function-call-logging () :  
[Command]
Stop function call logging and close the current log file.

stream-position (self) : LONG-INTEGER  
[Method on OUTPUT-FILE-STREAM]
Return the current position of the file input cursor in self.

stream-position (self) : LONG-INTEGER  
[Method on INPUT-FILE-STREAM]
Return the current position of the file input cursor in self.

stream-position-setter (self (newpos long-integer)) : LONG-INTEGER  
[Method on OUTPUT-FILE-STREAM]
Set the current position of the file input cursor in self to newpos.

stream-position-setter (self (newpos long-integer)) : LONG-INTEGER  
[Method on INPUT-FILE-STREAM]
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  
[Method on INPUT-STREAM]
Read all of the input from stream and return it as a string.
**string-compare-case-normalized** ((x string) (y string)) : [Function] integer

Compare x and y and return -1, 0, or 1, depending on whether x is less than, equal, or greater than y relative to the :ascii-case-normalized collation. In this collation a < A < b which gives strings that only differ in case a definite order while otherwise behaving identically to :ascii-case-insensitive. This is similar to using a Java Collator for Locale.US with strength set to TERTIARY (which see).

**string-search-ignore-case** ((string string) (substring string)) (start integer)) : [Function] integer

Return start position of the left-most occurrence of substring in string, beginning from start. Return NULL if it is not a substring. The comparison ignores differences in letter case.

**string-to-calendar-date-with-default** ((input-date string) (default-date decoded-date-time)) : [Function] 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)) : [Function] 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-int** ((string string)) : [Function] integer

Convert a string representation of an integer into an integer. This is a convenience function that ensures a regular integer return value. If string represents a long integer, the behavior is undefined. Use parse-integer if the syntax of string needs to be checked for errors.

**string-to-surrogate** ((self string)) : surrogate [Function]

Return 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 [Function]

Parses and returns an 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.

**string-trim** ((string string)) : string [Function]

Remove any leading and trailing white space from string and return a copy of the trimmed substring (which might be empty if we had all white space). If no white space was removed, string is returned unmodified and uncopied.

**subclass-of?** ((subClass class) (superClass class)) : boolean [Function]

Return true if subClass is a subclass of superClass.
subsequence \((self (start integer) (end integer))\) [Method on BUFFERED-STRING]

\[\text{: STRING}\]

Return a substring of \text{string} beginning at position \text{start} and ending up to but not including position \text{end}, counting from zero. An \text{end} value of NULL stands for the rest of the string.

subsequence \((string (start integer) (end integer))\) [Method on MUTABLE-STRING]

\[\text{: STRING}\]

Return a substring of \text{string} beginning at position \text{start} and ending up to but not including position \text{end}, counting from zero. An \text{end} value of NULL stands for the rest of the string.

substitute-characters \((self (new-chars string) (old-chars string))\) [Method on STRING]

\[\text{: STRING}\]

Substitute all occurrences of a member of \text{old-chars} with the corresponding member of \text{new-chars} in the string \text{self}. Returns a new string.

substitute-characters \((self (new-chars string) (old-chars string))\) [Method on MUTABLE-STRING]

\[\text{: MUTABLE-STRING}\]

Substitute all occurrences of a member of \text{old-chars} with the corresponding member of \text{new-chars} in the string \text{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 [Function]

Return \text{true} iff the class named \text{sub-type} is a subclass of the class named \text{super-type}.

super-classes \((self)\) : (iterator of class) [Method on CLASS]

Returns an iterator that generates all proper super classes of \text{self}.

surrogate-name? \((\text{name string})\) : BOOLEAN [Function]

Return \text{TRUE} if name is prefixed by \text{SURROGATE-PREFIX-CHARACTER}.

surrogatify \((self)\) : SURROGATE [Method on SURROGATE]

Converts \text{self} into a surrogate.

surrogatify \((self)\) : SURROGATE [Method on SYMBOL]

Converts \text{self} into a surrogate (same semantics as \text{symbol-to-surrogate} which see).

surrogatify \((self)\) : SURROGATE [Method on STRING]

Converts \text{self} into a surrogate.

sweep \((self)\) : [Method on OBJECT]

Default method. Sweep up all \text{self}-type objects.

symbol-to-surrogate \(((self symbol))\) : SURROGATE [Function]

Return a surrogate with the same name as \text{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 \text{string-to-surrogate}, the search starts first from the home context of \text{self}, and if that fails, then it restarts in *module*.
symbol-to-type ((self symbol)) : surrogat [Function]
Convert self into a surrogate with the same name and module.

system-default-value (self) : object [Method on storage-slot]
Return a default value expression, or if self has dynamic storage, an initial value expression.

system-default-value (self) : object [Method on slot]
Return a default value expression, or if self has dynamic storage, an initial value expression.

system-loaded-or-started-up? ((name string)) : boolean [Function]
Return true if system name has either been loaded or initialized with its startup function.

system-loaded? ((name string)) : boolean [Function]
Return true if system name has been loaded.

terminate-program () : [Command]
Terminate and exit the program with normal exit code.

test-property-demon ((action keyword) (property string) (value object) (table configuration-table)) :
A test demon for the property demon machinery which simply prints arguments.

the-string (self) : string [Method on buffered-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.

time-duration-to-string (date) : string [Method on time-duration]
Returns a string representation of date

time-zone-format60 ((timezone float) (include-colon? boolean)) : string
Format zone as an hh:mm or hhmm string depending on include-colon?

toggle-output-language () : keyword [Function]
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++)" "()=<>+-; "', "'" "")
=>
Chapter 6: Library Functions

(top-level-stella-system-directory? ((directory file-name)) : BOOLEAN)
Return TRUE if directory is a top-level STELLA installation directory containing one or more STELLA systems. We currently determine this solely by looking for the existence of at least one relevant top-level native directory. This is needed to find system root directories and other relevant STELLA installation directories.

(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?.
:recursive? (default false): if true, perform translate-system with the provided options on systemName as well as all its required systems and so on. Required systems will be processed first. Note that even without this option, any required systems that have not yet been loaded or started up will also be processed, since that is assumed when loading systemName and supporting modules from a definition file.
:root-source-directory, :root-native-directory, :root-binary-directory: if specified these directories will be used to override the respective paths provided in system definitions or computed as defaults from a system’s home location.
translate-to-common-lisp? () : BOOLEAN [Function]
   Return true if current output language is Common-Lisp.

translate-to-cpp? () : BOOLEAN [Function]
   Return true if current output language is C++

translate-to-java? () : BOOLEAN [Function]
   Return true if current output language is Java

truncate ((n NUMBER)) : INTEGER [Function]
   Truncate n toward zero and return the result.

try-to-evaluate ((tree OBJECT)) : OBJECT [Function]
   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 [Function]
   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 [Function]
   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 [Method on SLOT]
   The type of a storage slot is its base type.

type-specifier (self) : TYPE-SPEC [Method on SLOT]
   If self has a complex type return its type specifier, otherwise, return type of self.

type-to-symbol ((type TYPE)) : SYMBOL [Function]
   Convert type into a symbol with the same name and module.

type-to-wrapped-type (self) : TYPE [Method on SURROGATE]
   Return the wrapped type for the type self, or self if it is not a bare literal type.

unbound-surrogates ((module MODULE) (local? BOOLEAN)) : [Function]
   (ITERATOR OF SURROGATE)
   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 () : [Command]
   Decrease the indentation level for subsequent log messages.

unescape-html-string ((input STRING)) : STRING [Function]
   Replaces HTML escape sequences such as & with their associated characters.
unescape-url-string ((input string)) : string  [Function]
Takes a string and replaces %-format URL escape sequences with their real character equivalent according to RFC 2396.

unprocessed-command-line-arguments () : (cons of string-wrapper)  [Command]
Return all command line arguments which have not yet been processed by (or been ignored by) process-command-line-arguments. If arguments have not yet been processed, this will return NULL.

unregister-all-cmd-line-options () :  [Function]
Unregister all currently registered command line options.

unregister-cmd-line-option ((key string)) :  [Function]
Unregister the command line option identified by key under all its keys.

unregister-property-demon ((property string)) :  [Function]
Unregister any demon for property.

unset-stella-feature (&rest (features keyword)) :  [Command]
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
Unwrap wrapper and return the result. Return NULL if wrapper is NULL.

value-setter (self)

(value (LIKE (ANY-VALUE SELF))) : (LIKE (ANY-VALUE SELF))
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
Signal warning message, placing non-string arguments in quotes.

with-network-stream ((binding cons) &body (body cons)) : OBJECT
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
Allocate permanent (as opposed to transient) objects within the scope of this declaration.

with-system-definition ((systemnameexpression object) &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
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
Return a literal object whose value is the BOOLEAN value.

wrap-function-code ((value FUNCTION-CODE)) :
FUNCTION-CODE-WRAPPER
Return a literal object whose value is the FUNCTION-CODE value.

wrap-integer-value ((value LONG-INTEGER)) : NUMBER-WRAPPER
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
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-global-variable-value ((variable GLOBAL-VARIABLE)
    (value OBJECT)) : [Function]
    Interpreted global variable writer which sets its native value to value unwrappig it if
necessary. This will change the current value at the top of the special stack.

write-html-escaping-url-special-characters
    ((stream NATIVE-OUTPUT-STREAM) (input STRING)) : [Function]
    Writes a string and replaces unallowed URL characters according to RFC 2396 with
%-format URL escape sequences.

write-native-variable-value ((var NATIVE-OBJECT-POINTER)
    (type TYPE) (value OBJECT)) : [Function]
    Interpreted variable writer which sets a native code variable represented by var of
type type to value which will be unwrapped if necessary. If var is a special variable,
this will change the current dynamic value at the top of the special stack. This relies
upon type being completely accurate and bad things will happen if it is not.

write-slot-value ((self STANDARD-OBJECT) (slot STORAGE-SLOT)
    (value OBJECT)) : [Function]
    Write a (possibly wrapped) value for the slot slot on self.

writer (self) : SYMBOL [Method on STORAGE-SLOT]
    Name of a method called to write the value of the slot self.

xml-base-attribute? ((item OBJECT)) : BOOLEAN [Function]
    Return true if item is an XML attribute object

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

xml-element-form? ((form OBJECT)) : BOOLEAN [Function]
    Return true if form is a CONS headed by an XML ELEMENT tag

xml-global-attribute? ((item OBJECT)) : BOOLEAN [Function]
    Return true if item is an XML attribute object

xml-local-attribute? ((item OBJECT)) : BOOLEAN [Function]
    Return true if item is an XML attribute object
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

```xml
<a a1=v1 a2='v2'> foo <b a3=v3/> bar </a>
```

becomes

```scheme
(<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

### A
- abs on FLOAT .................................................. 29
- abs on INTEGER .............................................. 29
- abs on LONG-INTEGER ................................. 75
- absolute-pathname? ...................................... 68
- acos .......................................................... 29
- activate-demon ............................................ 75
- activate? on POLYMORPHIC-RELATION ............ 75
- add-configuration-property ......................... 75
- add-current-date-substitution ..................... 75
- add-date-substitution .................................. 75
- add-hook ....................................................... 75
- add-load-path .............................................. 75
- add-property-value ....................................... 76
- add-trace ...................................................... 76
- advance on ITERATOR .................................. 59
- advance-past-whitespace .......................... 76
- all-classes ..................................................... 76
- all-contexts ..................................................... 76
- all-defined? .................................................... 76
- all-functions ............................................... 76
- all-included-modules .................................... 76
- all-lower-case-string? ................................... 32
- all-methods ..................................................... 76
- all-modules .................................................... 76
- all-public-functions .................................... 76
- all-public-methods ....................................... 76
- all-required-systems .................................... 77
- all-slots ......................................................... 77
- all-subcontexts ............................................. 77
- all-surrogates ............................................... 77
- all-symbols ..................................................... 77
- all-upper-case-string? ................................... 32
- all-variables ................................................... 77
- allocate-cross-product-iterator .................... 77
- allocate-iterator on ABSTRACT-ITERATOR .......... 77
- allocate-iterator on CONS ................................ 40
- allocate-iterator on KEY-VALUE-LIST ............. 49
- allocate-iterator on KEY-VALUE-MAP ............... 57
- allocate-iterator on LIST ............................. 44
- allocate-iterator on MEMOIZABLE-ITERATOR ....... 77
- allocate-iterator on PROPERTY-LIST ............... 47
- allocate-iterator on STELLA-HASH-TABLE ......... 54
- allocation on STORAGE-SLOT ...................... 77
- append .......................................................... 38
- append-character on BUFFERED-STRING .......... 77
- append-string on BUFFERED-STRING ............... 78
- apply ............................................................ 78
- apply-boolean-method ................................. 78
- apply-float-method ....................................... 78
- apply-integer-method .................................... 78
- apply-long-integer-method ......................... 78
- apply-method .................................................. 78

### 1
- 1+ ................................................................. 27
- 1- ................................................................. 27
- 1d-aref on DIMENSIONAL-ARRAY-MIXIN .............. 74
- 1d-aref-address on DIMENSIONAL-ARRAY-MIXIN ... 74
- 1d-aref-setter on DIMENSIONAL-ARRAY-MIXIN .... 74

### 2
- 2d-aref on 2-DIMENSIONAL-ARRAY-MIXIN ........... 74
- 2d-aref-address on 2-DIMENSIONAL-ARRAY-MIXIN ... 75
- 2d-aref-setter on 2-DIMENSIONAL-ARRAY-MIXIN ... 75
Function Index 134

consify on KEY-VALUE-LIST ....................... 49
consify on KEY-VALUE-MAP .......................... 57
consify on LIST ................................... 44
consify on OBJECT ................................. 84
consify on STELLA-HASH-TABLE .................... 54
consify on VECTOR ................................. 51
consify-command-line-arguments .................... 84
continuable-error ................................. 84
copy on CUSTOM-VECTOR-SEQUENCE ................. 84
copy on HASH-SET .................................. 58
copy on HEAP ....................................... 84
copy on KEY-VALUE-LIST ......................... 49
copy on KEY-VALUE-MAP ........................... 57
copy on LIST ...................................... 44
copy on MUTABLE-STRING ......................... 84
copy on PROPERTY-LIST ............................ 47
copy on STELLA-HASH-TABLE ....................... 54
copy on STRING ................................... 34
copy on VECTOR ................................... 51
copy on VECTOR-SEQUENCE .......................... 84
copy-cons-list .................................... 39
copy-cons-tree .................................... 39
copy-file ......................................... 67
copy-kv-cons-list .................................. 47
copy-stream-to-stream .............................. 84
cos ................................................... 29
cpp-find-native-slot-value-offset ................. 84
cpp-translate-system ............................... 84
cpptrans .......................................... 84
create-derived-list ................................ 84
create-directories ................................ 85
create-object ..................................... 85
create-world ..................................... 65
date-to-version-string .............................. 85
deactivate-demon .................................. 85
decode-calendar-date on CALENDAR-DATE .......... 69
decode-time-in-millis .............................. 70
decoded-date-time-to-is08601-string on 
   DECODED-DATE-TIME ................................ 85
decompose-namestring .............................. 85
decompose-namestring-full ......................... 86
default-cmd-line-option-handler .................. 86
default-form on STORAGE-SLOT ..................... 86
defdemon ........................................... 86
define-demon ...................................... 86
define-global-variable-object ...................... 86
define-logical-host-property ....................... 86
define-module ..................................... 86
define-stella-class ................................ 86
define-stella-global-variable-from-
   stringified-source ................................ 86
define-stella-method-slot ......................... 87
defined? on ARRAY ................................ 42
defined? on CHARACTER ........................... 26
defined? on CODE ................................. 26
defined? on FLOAT ................................. 27
defined? on INTEGER .............................. 27
defined? on LONG-INTEGER ......................... 87
defined? on MUTABLE-STRING ...................... 26
defined? on NATIVE-VECTOR ......................... 26
defined? on OBJECT ............................... 26
defined? on SECOND-CLASS-OBJECT ................ 26
defined? on STRING ............................... 26
defmain ........................................... 87
defmodule ......................................... 63
defsystem .......................................... 87
delete-file ....................................... 68
defiled on OBJECT .................................. 87
describe .......................................... 87
describe-object on OBJECT ......................... 88
destroy-class on CLASS ............................ 88
destroy-class-and-subclasses ...................... 88
destroy-context on CONTEXT ....................... 63
destroy-context on MODULE ....................... 65
destroy-context on STRING ......................... 63
destroy-context on WORLD ......................... 65
destroy-module .................................... 65
destructure-defmethod-tree ....................... 88
dictionary ......................................... 88
difference on CONS ................................ 41
difference on HASH-SET ............................ 59
difference on LIST ................................ 45
digit-character? .................................. 30
direct-super-classes on CLASS ..................... 88
directory-file-name ................................ 68
directory-parent-directory ....................... 68
directory-separator .................................. 69
directory-separator-string ....................... 69
disable-memoization .............................. 88
disabled-stella-feature? ......................... 88
disjoint-sets? on CONS ........................... 88
div ................................................... 28
div on INTEGER ................................... 88
div on LONG-INTEGER ............................. 88
drop-hook ......................................... 88
drop-load-path .................................... 88
drop-slot-value ................................... 89
drop-trace ........................................ 89
drop? on BUFFERED-STRING ......................... 89
drop? on CONS ..................................... 36
drop? on HEAP ..................................... 89
drop? on ITERATOR ................................ 59
drop? on KEY-VALUE-LIST ......................... 48
drop? on KEY-VALUE-MAP ......................... 57
drop? on LIST ..................................... 42
empty? on PROPERTY-LIST ......................... 46
empty? on STELLA-HASH-TABLE ...................... 54

d D

d e

eight-bit-character? .............................. 89
either .............................................. 89
empty? on BUFFERED-STRING ....................... 89
empty? on CONS ................................... 36
empty? on HEAP ................................... 89
empty? on ITERATOR .............................. 59
empty? on KEY-VALUE-LIST ....................... 48
empty? on KEY-VALUE-MAP ......................... 57
empty? on LIST ................................... 42
empty? on PROPERTY-LIST ......................... 46
empty? on STELLA-HASH-TABLE ...................... 54
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>help-find-matching-prefix-length</td>
<td>97</td>
</tr>
<tr>
<td>help-advance-past-whitespace</td>
<td>97</td>
</tr>
<tr>
<td>help-option-handler</td>
<td>97</td>
</tr>
<tr>
<td>help-print-outline</td>
<td>98</td>
</tr>
<tr>
<td>help-print-outline on CLASS</td>
<td>98</td>
</tr>
<tr>
<td>help-print-outline on CONTEXT</td>
<td>98</td>
</tr>
<tr>
<td>help-print-outline on MODULE</td>
<td>98</td>
</tr>
<tr>
<td>help-print-outline on OBJECT</td>
<td>98</td>
</tr>
<tr>
<td>help-print-outline on SLOT</td>
<td>98</td>
</tr>
<tr>
<td>hex-character-value</td>
<td>98</td>
</tr>
<tr>
<td>home-module on OBJECT</td>
<td>98</td>
</tr>
<tr>
<td>get-language-subdirectory-from-file-type</td>
<td>94</td>
</tr>
<tr>
<td>get-load-path</td>
<td>94</td>
</tr>
<tr>
<td>get-local-standard-time-zone</td>
<td>95</td>
</tr>
<tr>
<td>get-local-time-zone</td>
<td>69</td>
</tr>
<tr>
<td>get-local-time-zone-for-date</td>
<td>95</td>
</tr>
<tr>
<td>get-log-stream</td>
<td>95</td>
</tr>
<tr>
<td>get-logical-host-property</td>
<td>95</td>
</tr>
<tr>
<td>get-native-source-subdirectory-from-file-type</td>
<td>95</td>
</tr>
<tr>
<td>get-property</td>
<td>95</td>
</tr>
<tr>
<td>get-quoted-tree</td>
<td>95</td>
</tr>
<tr>
<td>get-root-directory-from-file-extension</td>
<td>95</td>
</tr>
<tr>
<td>get-root-directory-from-file-type</td>
<td>95</td>
</tr>
<tr>
<td>get-slot</td>
<td>96</td>
</tr>
<tr>
<td>get-stella-class on STRING</td>
<td>96</td>
</tr>
<tr>
<td>get-stella-class on SURROGATE</td>
<td>95</td>
</tr>
<tr>
<td>get-stella-class on SYMBOL</td>
<td>96</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-system-definition</td>
<td>96</td>
</tr>
<tr>
<td>get-system-last-modified-date</td>
<td>96</td>
</tr>
<tr>
<td>get-system-subdirectory-from-file-type</td>
<td>96</td>
</tr>
<tr>
<td>get-system-version-string</td>
<td>96</td>
</tr>
<tr>
<td>get-temp-directory</td>
<td>96</td>
</tr>
<tr>
<td>get-ticktock</td>
<td>71</td>
</tr>
<tr>
<td>get-time on CALENDAR-DATE</td>
<td>96</td>
</tr>
<tr>
<td>get-token-float</td>
<td>96</td>
</tr>
<tr>
<td>get-token-integer</td>
<td>96</td>
</tr>
<tr>
<td>get-token-long-integer</td>
<td>96</td>
</tr>
<tr>
<td>get-xml-attributes</td>
<td>72</td>
</tr>
<tr>
<td>get-xml-base-attribute-value</td>
<td>97</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-cdata-content</td>
<td>72</td>
</tr>
<tr>
<td>get-xml-tag</td>
<td>72</td>
</tr>
<tr>
<td>getenv</td>
<td>97</td>
</tr>
<tr>
<td>global-variable-type-spec</td>
<td>97</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>97</td>
</tr>
<tr>
<td>hashmod</td>
<td>55</td>
</tr>
<tr>
<td>heap-root on HEAP</td>
<td>97</td>
</tr>
<tr>
<td>heapify on HEAP</td>
<td>97</td>
</tr>
<tr>
<td>help-advance-past-whitespace</td>
<td>97</td>
</tr>
<tr>
<td>help-find-matching-prefix-length</td>
<td>97</td>
</tr>
<tr>
<td>help-get-stella-module</td>
<td>97</td>
</tr>
<tr>
<td>help-option-handler</td>
<td>97</td>
</tr>
<tr>
<td>help-print-outline on CLASS</td>
<td>98</td>
</tr>
<tr>
<td>help-print-outline on CONTEXT</td>
<td>98</td>
</tr>
<tr>
<td>help-print-outline on MODULE</td>
<td>98</td>
</tr>
<tr>
<td>help-print-outline on OBJECT</td>
<td>98</td>
</tr>
<tr>
<td>help-print-outline on SLOT</td>
<td>98</td>
</tr>
<tr>
<td>hex-character-value</td>
<td>98</td>
</tr>
<tr>
<td>home-module on OBJECT</td>
<td>98</td>
</tr>
</tbody>
</table>

**I**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>8</td>
</tr>
<tr>
<td>if-output-language</td>
<td>98</td>
</tr>
<tr>
<td>if-stella-feature</td>
<td>98</td>
</tr>
<tr>
<td>ignore</td>
<td>98</td>
</tr>
<tr>
<td>import-surrogate</td>
<td>61</td>
</tr>
<tr>
<td>import-symbol</td>
<td>60</td>
</tr>
<tr>
<td>in-module</td>
<td>65</td>
</tr>
<tr>
<td>incrementally-translate</td>
<td>98</td>
</tr>
<tr>
<td>indent-outline</td>
<td>98</td>
</tr>
<tr>
<td>inform</td>
<td>98</td>
</tr>
<tr>
<td>initial-value on CLASS</td>
<td>99</td>
</tr>
<tr>
<td>initial-value on STORAGE-SLOT</td>
<td>99</td>
</tr>
<tr>
<td>initialize-array</td>
<td>99</td>
</tr>
<tr>
<td>initialize-hash-table on STELLA-HASH-TABLE</td>
<td>99</td>
</tr>
<tr>
<td>initially on STORAGE-SLOT</td>
<td>99</td>
</tr>
<tr>
<td>insert on CUSTOM-VECTOR-SEQUENCE</td>
<td>99</td>
</tr>
<tr>
<td>insert on HASH-SET</td>
<td>58</td>
</tr>
<tr>
<td>insert on HEAP</td>
<td>99</td>
</tr>
<tr>
<td>insert on LIST</td>
<td>43</td>
</tr>
<tr>
<td>insert on SET</td>
<td>45</td>
</tr>
<tr>
<td>insert on VECTOR-SEQUENCE</td>
<td>51</td>
</tr>
<tr>
<td>insert-at on EXTENSIBLE-VECTOR</td>
<td>51</td>
</tr>
<tr>
<td>insert-at on FLOAT-HASH-TABLE</td>
<td>53</td>
</tr>
<tr>
<td>insert-at on HASH-TABLE</td>
<td>52</td>
</tr>
<tr>
<td>insert-at on INTEGER-HASH-TABLE</td>
<td>53</td>
</tr>
<tr>
<td>insert-at on KEY-VALUE-LIST</td>
<td>48</td>
</tr>
<tr>
<td>insert-at on KEY-VALUE-MAP</td>
<td>56</td>
</tr>
<tr>
<td>insert-at on PROPERTY-LIST</td>
<td>46</td>
</tr>
<tr>
<td>insert-at on STELLA-HASH-TABLE</td>
<td>53</td>
</tr>
<tr>
<td>insert-at on STRING-HASH-TABLE</td>
<td>52</td>
</tr>
<tr>
<td>insert-at on STRING-TO-INTEGER-HASH-TABLE</td>
<td>52</td>
</tr>
<tr>
<td>insert-at on VECTOR</td>
<td>50</td>
</tr>
<tr>
<td>insert-entry on KEY-VALUE-LIST</td>
<td>48</td>
</tr>
<tr>
<td>insert-if-better on HEAP</td>
<td>99</td>
</tr>
<tr>
<td>insert-last on LIST</td>
<td>43</td>
</tr>
<tr>
<td>insert-last on SET</td>
<td>45</td>
</tr>
<tr>
<td>insert-new on LIST</td>
<td>43</td>
</tr>
<tr>
<td>insert-string</td>
<td>35</td>
</tr>
<tr>
<td>instantiate-string-template</td>
<td>35</td>
</tr>
<tr>
<td>int-to-string</td>
<td>99</td>
</tr>
<tr>
<td>integer-length</td>
<td>99</td>
</tr>
<tr>
<td>integer-length10</td>
<td>99</td>
</tr>
<tr>
<td>integer-to-hex-string</td>
<td>100</td>
</tr>
</tbody>
</table>
integer-to-string ........................................ 30, 35
integer-to-string-in-base .................................. 100
integer-valued? .............................................. 100
intern-derived-surrogate ................................... 61
intern-derived-symbol ...................................... 60
intern-keyword ............................................. 62
intern-stella-name ........................................ 100
intern-surrogate ........................................... 61
intern-surrogate-in-module ................................ 61
intern-symbol ............................................... 60
intern-symbol-in-module .................................... 60
interpret-command-line-arguments .......................... 100
intersection on CONS ........................................ 41
intersection on HASH-SET .................................... 59
intersection on LIST .......................................... 45
isa? .......................................................... 100

J
java-translate-system ...................................... 100
jptrans ....................................................... 100
julian-day-to-modified-julian-day ....................... 70

K
keyword-name? ............................................... 100
kv-cons ....................................................... 47
kv-push on KEY-VALUE-LIST ................................ 49
kvlist-to-plist ............................................. 100

L
last on BUFFERED-STRING .................................. 100
last on CONS ................................................ 37
last on HEAP ............................................... 100
last on LIST ............................................... 42
last on VECTOR ............................................ 50
last on VECTOR-SEQUENCE .................................. 100
last-cons .................................................... 37
last-position on CONS ...................................... 38
last-position on LIST ....................................... 43
last-position on STRING .................................... 34
last-position on VECTOR .................................... 50
length on 2-DIMENSIONAL-ARRAY-MIXIN .................. 101
length on ABSTRACT-ITERATOR ............................. 59
length on CONS ............................................. 37
length on CONS-ITERATOR .................................. 100
length on DIMENSIONAL-ARRAY-MIXIN ..................... 101
length on HEAP ............................................. 101
length on KEY-VALUE-LIST ................................ 48
length on KEY-VALUE-MAP ................................ 57
length on LIST .............................................. 42
length on MUTABLE-STRING ................................ 33
length on PROPERTY-LIST .................................. 46
length on STELLA-HASH-TABLE ............................. 54
length on STRING ........................................... 33
length on VECTOR ........................................... 50
length on VECTOR-SEQUENCE ............................... 51
length-setter on BUFFERED-STRING ....................... 101
letter-character? .......................................... 31
lispify ....................................................... 101
lispify-boolean ............................................ 101
list ........................................................ 42
list* ........................................................ 39
list-directory-files ........................................ 101
list-directory-files-recursively ............................ 101
list-modules ............................................... 101
listify on CONS ............................................ 101
listify on ITERATOR ........................................ 102
listify on KEY-VALUE-LIST ................................ 102
listify on LIST ............................................. 101
listify on VECTOR ......................................... 102
load-configuration-file .................................... 102
load-file .................................................... 102
load-file-option-handler ................................... 102
load-path-option-handler .................................. 102
load-system ................................................ 102
local-gensym .............................................. 62
log .......................................................... 29
log-level<= ............................................... 103
log-message ............................................... 103
log10 ....................................................... 29
log2 ........................................................ 103
logical-host? ............................................... 69
logical-pathname? .......................................... 69
logmsg ....................................................... 103
lookup on HASH-TABLE ...................................... 52
lookup on INTEGER-HASH-TABLE ............................ 53
lookup on KEY-VALUE-LIST ................................ 48
lookup on KEY-VALUE-MAP ................................ 56
lookup on PROPERTY-LIST .................................. 46
lookup on STELLA-HASH-TABLE ............................. 53
lookup on STRING-HASH-TABLE ............................. 52
lookup on STRING-TO-INTEGER-HASH-TABLE ............... 52
lookup-class on STRING ..................................... 103
lookup-class on SYMBOL ................................... 103
lookup-class-by-qualified-name ............................. 103
lookup-class-cpp-type-by-name ............................. 103
lookup-command ............................................. 103
lookup-command-like-function .............................. 103
lookup-configuration-property ............................. 103
lookup-configuration-property-values ...................... 103
lookup-demon .............................................. 104
lookup-function ............................................ 104
lookup-function-by-name ................................... 104
lookup-function-by-qualified-name ....................... 104
lookup-global-variable on GENERALIZED-SYMBOL .......... 104
lookup-global-variable on STRING ......................... 104
lookup-global-variable on SURROGATE .................... 104
lookup-global-variable-by-name ........................... 104
lookup-global-variable-by-qualified-name ................ 104
lookup-keyword ............................................ 62
lookup-local-slot .......................................... 104
lookup-logging-parameter .................................. 104
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>lookup-macro</td>
<td>104</td>
</tr>
<tr>
<td>lookup-slot</td>
<td>104</td>
</tr>
<tr>
<td>lookup-slot-and-class-by-qualified-name</td>
<td>105</td>
</tr>
<tr>
<td>lookup-slot-by-qualified-name</td>
<td>105</td>
</tr>
<tr>
<td>lookup-stella-name</td>
<td>105</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>105</td>
</tr>
<tr>
<td>lookup-with-default on PROPERTY-LIST</td>
<td>105</td>
</tr>
<tr>
<td>lower-case-character?</td>
<td>31</td>
</tr>
<tr>
<td>lptrans</td>
<td>105</td>
</tr>
</tbody>
</table>

**M**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>make-current-date-time</td>
<td>69</td>
</tr>
<tr>
<td>make-date-time</td>
<td>69</td>
</tr>
<tr>
<td>make-file-name</td>
<td>105</td>
</tr>
<tr>
<td>make-matching-name</td>
<td>106</td>
</tr>
<tr>
<td>make-matching-name-full</td>
<td>106</td>
</tr>
<tr>
<td>make-mutable-string</td>
<td>32</td>
</tr>
<tr>
<td>make-raw-mutable-string</td>
<td>32</td>
</tr>
<tr>
<td>make-string</td>
<td>32</td>
</tr>
<tr>
<td>make-system</td>
<td>107</td>
</tr>
<tr>
<td>make-system-definition-file-name</td>
<td>107</td>
</tr>
<tr>
<td>make-temporary-file</td>
<td>107</td>
</tr>
<tr>
<td>make-temporary-file-name</td>
<td>68</td>
</tr>
<tr>
<td>make-xml-element</td>
<td>71</td>
</tr>
<tr>
<td>make-xml-global-attribute</td>
<td>72</td>
</tr>
<tr>
<td>make-xml-local-attribute</td>
<td>72</td>
</tr>
<tr>
<td>map-null-to-nil</td>
<td>40</td>
</tr>
<tr>
<td>map-null-to-nil-list</td>
<td>44</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>108</td>
</tr>
<tr>
<td>max on INTEGER</td>
<td>107</td>
</tr>
<tr>
<td>max on LONG-INTEGER</td>
<td>107</td>
</tr>
<tr>
<td>max on NUMBER-WRAPPER</td>
<td>108</td>
</tr>
<tr>
<td>memb? on CONS</td>
<td>38</td>
</tr>
<tr>
<td>memb? on LIST</td>
<td>43</td>
</tr>
<tr>
<td>member? on COLLECTION</td>
<td>108</td>
</tr>
<tr>
<td>member? on CONS</td>
<td>37</td>
</tr>
<tr>
<td>member? on CONS-ITERATOR</td>
<td>108</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>108</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>108</td>
</tr>
<tr>
<td>merge-file-names</td>
<td>108</td>
</tr>
<tr>
<td>merge-null-fields on DECODED-DATE-TIME</td>
<td>108</td>
</tr>
<tr>
<td>merge-superior-null-fields on DECODED-DATE-TIME</td>
<td>108</td>
</tr>
<tr>
<td>min</td>
<td>29</td>
</tr>
<tr>
<td>min on FLOAT</td>
<td>109</td>
</tr>
<tr>
<td>min on INTEGER</td>
<td>109</td>
</tr>
<tr>
<td>min on LONG-INTEGER</td>
<td>109</td>
</tr>
<tr>
<td>min on NUMBER-WRAPPER</td>
<td>109</td>
</tr>
<tr>
<td>mod</td>
<td>28</td>
</tr>
<tr>
<td>mod on INTEGER</td>
<td>109</td>
</tr>
<tr>
<td>mod on LONG-INTEGER</td>
<td>109</td>
</tr>
<tr>
<td>modified-julian-day-to-julian-day</td>
<td>70</td>
</tr>
<tr>
<td>multiple-parents? on CLASS</td>
<td>109</td>
</tr>
<tr>
<td>multiple-parents? on MODULE</td>
<td>109</td>
</tr>
<tr>
<td>multiple-parents? on WORLD</td>
<td>109</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>109</td>
</tr>
<tr>
<td>native-delete-file</td>
<td>109</td>
</tr>
<tr>
<td>native-file-length</td>
<td>109</td>
</tr>
<tr>
<td>native-file-write-date</td>
<td>109</td>
</tr>
<tr>
<td>native-probe-directory?</td>
<td>110</td>
</tr>
<tr>
<td>native-probe-file?</td>
<td>110</td>
</tr>
<tr>
<td>native-read-line</td>
<td>110</td>
</tr>
<tr>
<td>native-rename-file</td>
<td>110</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>110</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>110</td>
</tr>
<tr>
<td>non-empty? on BUFFERED-STRING</td>
<td>110</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>110</td>
</tr>
<tr>
<td>non-empty? on VECTOR</td>
<td>49</td>
</tr>
<tr>
<td>non-empty? on VECTOR-SEQUENCE</td>
<td>110</td>
</tr>
<tr>
<td>non-matching-position</td>
<td>110</td>
</tr>
<tr>
<td>non-matching-position-helper</td>
<td>110</td>
</tr>
<tr>
<td>nth on BUFFERED-STRING</td>
<td>110</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>110</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>nth-setter on BUFFERED-STRING</td>
<td>111</td>
</tr>
<tr>
<td>null-list?</td>
<td>42</td>
</tr>
<tr>
<td>null? on ARRAY</td>
<td>111</td>
</tr>
<tr>
<td>null? on CHARACTER</td>
<td>26</td>
</tr>
<tr>
<td>null? on CODE</td>
<td>26</td>
</tr>
</tbody>
</table>
null? on FLOAT ................................................. 26
null? on INTEGER .............................................. 26
null? on LONG-INTEGER ....................................... 111
null? on MUTABLE-STRING ...................................... 26
null? on NATIVE-VECTOR ......................................... 26
null? on OBJECT ................................................ 26
null? on SECOND-CLASS-OBJECT .................................. 26
null? on STRING ................................................... 26
number-less-than? ................................................. 111

O
object-equal? on CONS .............................................. 37
object-equal? on HASH-SET ........................................... 59
object-equal? on KEY-VALUE-LIST ................................. 48
object-equal? on KEY-VALUE-MAP .................................. 57
object-equal? on LIST ............................................... 42
object-equal? on OBJECT .......................................... 27
object-equal? on PROPERTY-LIST ................................. 46
object-equal? on SET ................................................. 46
object-equal? on STELLA-HASH-TABLE ............................ 54
object-equal? on VECTOR .............................................. 49
object-equal? on VECTOR-SEQUENCE ............................... 111
object-equal? on WRAPPER ......................................... 27
object-hash-code ...................................................... 54
odd? ................................................................. 28
odd? on INTEGER .................................................. 111
odd? on LONG-INTEGER .......................................... 111
only-if ............................................................... 111
open-input-file ...................................................... 66
open-network-stream ............................................... 111
open-output-file ...................................................... 67
operating-system .................................................... 74
ordered? on COLLECTION .......................................... 111
outline-depth-exceeded? ............................................ 111

P
parameters on CLASS .............................................. 111
parse-date-time ....................................................... 69
parse-date-time-in-time-zone ........................................ 111
parse-date-time-relative-to-base .................................. 112
parse-float ............................................................ 112
parse-integer ......................................................... 112
parse-long-integer .................................................. 112
parse-options ......................................................... 112
parse-stella-name .................................................... 112
pick-hash-table-size-prime .......................................... 113
plist-to-kvlist ....................................................... 113
plus? ................................................................. 28
plus? on INTEGER .................................................. 113
plus? on LONG-INTEGER .......................................... 113
pop on HASH-SET .................................................... 58
pop on ITERATOR ..................................................... 59
pop on KEY-VALUE-LIST ............................................ 49
pop on LIST ........................................................... 43
pop-load-path ........................................................ 113
pop-world ............................................................. 65
popq ................................................................. 39
position on CONS .................................................... 38
position on LIST ..................................................... 43
position on STRING .................................................. 33
position on VECTOR .................................................. 50
prepend on CONS ..................................................... 38
prepend on LIST ..................................................... 44
primary-type on OBJECT .......................................... 113
primitive? on RELATION .......................................... 113
print ................................................................. 113
print-configuration-properties ...................................... 113
print-exception-context ............................................. 113
print-outline ........................................................ 113
print-properties ...................................................... 113
print-recycle-lists .................................................. 114
print-spaces .......................................................... 114
print-stella-features ............................................... 114
print-unbound-surrogates .......................................... 114
print-undefined-methods ............................................ 114
print-undefined-super-classes ...................................... 114
print-xml-expression ............................................... 74
private-class-methods ............................................... 114
private-class-storage-slots ........................................ 114
private? on RELATION .............................................. 114
probe-directory? ..................................................... 114
probe-file? .......................................................... 67
process-command-line-arguments ................................... 114
process-doctype ..................................................... 114
ptrans ................................................................. 115
public-class-methods ............................................... 115
public-class-storage-slots .......................................... 115
public-slots on CLASS .............................................. 115
public-slots on OBJECT ............................................. 115
public? on SLOT ..................................................... 115
push on KEY-VALUE-LIST .......................................... 48
push on LIST ........................................................ 43
push on SET ........................................................ 45
push-load-path ....................................................... 115
push-world ........................................................... 65
pushf ................................................................. 115
pushq ................................................................. 39
pushq-new ........................................................... 39
put-slot-value ........................................................ 115

Q
qualified-stella-name ............................................. 115
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reverse</td>
<td>on VECTOR-SEQUENCE</td>
</tr>
<tr>
<td>reverse-interval</td>
<td></td>
</tr>
<tr>
<td>root-binary-directory</td>
<td></td>
</tr>
<tr>
<td>root-native-directory</td>
<td></td>
</tr>
<tr>
<td>root-source-directory</td>
<td></td>
</tr>
<tr>
<td>rotate-hash-code</td>
<td></td>
</tr>
<tr>
<td>round</td>
<td></td>
</tr>
<tr>
<td>run-hooks</td>
<td></td>
</tr>
<tr>
<td>running-as-lisp?</td>
<td></td>
</tr>
<tr>
<td>running-in-language</td>
<td></td>
</tr>
<tr>
<td>running-system-information</td>
<td></td>
</tr>
<tr>
<td>safe-compare-strings</td>
<td></td>
</tr>
<tr>
<td>safe-equal-hash-code</td>
<td></td>
</tr>
<tr>
<td>safe-hash-code</td>
<td></td>
</tr>
<tr>
<td>safe-import-surrogate</td>
<td></td>
</tr>
<tr>
<td>safe-import-symbol</td>
<td></td>
</tr>
<tr>
<td>safe-lookup-slot</td>
<td></td>
</tr>
<tr>
<td>safe-tree-size</td>
<td></td>
</tr>
<tr>
<td>safety</td>
<td></td>
</tr>
<tr>
<td>save-configuration-file</td>
<td></td>
</tr>
<tr>
<td>save-configuration-value</td>
<td></td>
</tr>
<tr>
<td>search-cons-tree-with-filter?</td>
<td></td>
</tr>
<tr>
<td>search-cons-tree?</td>
<td></td>
</tr>
<tr>
<td>search-for-object</td>
<td></td>
</tr>
<tr>
<td>second on CONS</td>
<td></td>
</tr>
<tr>
<td>second on LIST</td>
<td></td>
</tr>
<tr>
<td>second on MUTABLE-STRING</td>
<td></td>
</tr>
<tr>
<td>second on STRING</td>
<td></td>
</tr>
<tr>
<td>second on VEC</td>
<td></td>
</tr>
<tr>
<td>seed-random-number-generator</td>
<td></td>
</tr>
<tr>
<td>seed-random-number-generator2</td>
<td></td>
</tr>
<tr>
<td>sequence</td>
<td></td>
</tr>
<tr>
<td>set</td>
<td></td>
</tr>
<tr>
<td>set-call-log-break-point</td>
<td></td>
</tr>
<tr>
<td>set-configuration-property</td>
<td></td>
</tr>
<tr>
<td>set-current-date on DECODED-DATE-TIME</td>
<td></td>
</tr>
<tr>
<td>set-current-date-time on DECODED-DATE-TIME</td>
<td></td>
</tr>
<tr>
<td>set-current-time on DECODED-DATE-TIME</td>
<td></td>
</tr>
<tr>
<td>set-load-path</td>
<td></td>
</tr>
<tr>
<td>set-log-level</td>
<td></td>
</tr>
<tr>
<td>set-logging-parameters</td>
<td></td>
</tr>
<tr>
<td>set-optimization-levels</td>
<td></td>
</tr>
<tr>
<td>set-property</td>
<td></td>
</tr>
<tr>
<td>set-stella-feature</td>
<td></td>
</tr>
<tr>
<td>set-translator-output-language</td>
<td></td>
</tr>
<tr>
<td>setq</td>
<td></td>
</tr>
<tr>
<td>shadowed-symbol?</td>
<td></td>
</tr>
<tr>
<td>shift-right on INTEGER</td>
<td></td>
</tr>
<tr>
<td>shift-right on LONG-INTEGER</td>
<td></td>
</tr>
<tr>
<td>signal</td>
<td></td>
</tr>
<tr>
<td>signal-read-error</td>
<td></td>
</tr>
<tr>
<td>sin</td>
<td></td>
</tr>
<tr>
<td>sleep</td>
<td></td>
</tr>
<tr>
<td>sort on CONS</td>
<td></td>
</tr>
<tr>
<td>Function Index</td>
<td>141</td>
</tr>
<tr>
<td>---------------</td>
<td>-----</td>
</tr>
<tr>
<td>sort on HEAP</td>
<td>121</td>
</tr>
<tr>
<td>sort on LIST</td>
<td>44</td>
</tr>
<tr>
<td>sort on VECTOR</td>
<td>121</td>
</tr>
<tr>
<td>sort-objects on CONS</td>
<td>121</td>
</tr>
<tr>
<td>sort-objects on VECTOR</td>
<td>121</td>
</tr>
<tr>
<td>sort-tuples on CONS</td>
<td>40</td>
</tr>
<tr>
<td>sort-tuples on VECTOR</td>
<td>121</td>
</tr>
<tr>
<td>split-string</td>
<td>121</td>
</tr>
<tr>
<td>sqrt</td>
<td>20</td>
</tr>
<tr>
<td>start-function-call-logging</td>
<td>121</td>
</tr>
<tr>
<td>starts-with?</td>
<td>121</td>
</tr>
<tr>
<td>stella-collection?</td>
<td>122</td>
</tr>
<tr>
<td>stella-information</td>
<td>122</td>
</tr>
<tr>
<td>stella-integer-to-string-in-base</td>
<td>122</td>
</tr>
<tr>
<td>stella-object?</td>
<td>122</td>
</tr>
<tr>
<td>stella-version-string</td>
<td>122</td>
</tr>
<tr>
<td>stellify</td>
<td>122</td>
</tr>
<tr>
<td>stop-function-call-logging</td>
<td>122</td>
</tr>
<tr>
<td>stream-position on input-file-stream</td>
<td>122</td>
</tr>
<tr>
<td>stream-position on output-file-stream</td>
<td>122</td>
</tr>
<tr>
<td>stream-position-setter on input-file-stream</td>
<td>122</td>
</tr>
<tr>
<td>stream-position-setter on output-file-stream</td>
<td>122</td>
</tr>
<tr>
<td>string-to-string on input-stream</td>
<td>122</td>
</tr>
<tr>
<td>string-capitalize</td>
<td>34</td>
</tr>
<tr>
<td>string-compare</td>
<td>32</td>
</tr>
<tr>
<td>string-compare-case-normalized</td>
<td>123</td>
</tr>
<tr>
<td>string-downcase</td>
<td>34</td>
</tr>
<tr>
<td>string-eqv?</td>
<td>31</td>
</tr>
<tr>
<td>string-equal?</td>
<td>31</td>
</tr>
<tr>
<td>string-greater-equal?</td>
<td>32</td>
</tr>
<tr>
<td>string-greater?</td>
<td>32</td>
</tr>
<tr>
<td>string-less-equal?</td>
<td>32</td>
</tr>
<tr>
<td>string-less?</td>
<td>32</td>
</tr>
<tr>
<td>string-search</td>
<td>34</td>
</tr>
<tr>
<td>string-search-ignore-case</td>
<td>123</td>
</tr>
<tr>
<td>string-to-calendar-date</td>
<td>70</td>
</tr>
<tr>
<td>string-to-calendar-date-with-default</td>
<td>123</td>
</tr>
<tr>
<td>string-to-decoded-date-time</td>
<td>123</td>
</tr>
<tr>
<td>string-to-float</td>
<td>30, 35</td>
</tr>
<tr>
<td>string-to-int</td>
<td>123</td>
</tr>
<tr>
<td>string-to-integer</td>
<td>30, 35</td>
</tr>
<tr>
<td>string-to-mutable-string</td>
<td>35</td>
</tr>
<tr>
<td>string-to-surrogate</td>
<td>123</td>
</tr>
<tr>
<td>string-to-time-duration</td>
<td>123</td>
</tr>
<tr>
<td>string-trim</td>
<td>123</td>
</tr>
<tr>
<td>string-upcase</td>
<td>34</td>
</tr>
<tr>
<td>string&lt;</td>
<td>32</td>
</tr>
<tr>
<td>string&lt;=</td>
<td>32</td>
</tr>
<tr>
<td>string?&gt;</td>
<td>32</td>
</tr>
<tr>
<td>string&gt;=</td>
<td>32</td>
</tr>
<tr>
<td>stringify</td>
<td>36</td>
</tr>
<tr>
<td>stringify-in-module</td>
<td>36</td>
</tr>
<tr>
<td>subclass-of?</td>
<td>123</td>
</tr>
<tr>
<td>subsequence on buffered-string</td>
<td>124</td>
</tr>
<tr>
<td>subsequence on mutable-string</td>
<td>124</td>
</tr>
<tr>
<td>subsequence on STRING</td>
<td>34</td>
</tr>
<tr>
<td>subset? on CONS</td>
<td>41</td>
</tr>
<tr>
<td>subset? on hash-set</td>
<td>58</td>
</tr>
<tr>
<td>subset? on LIST</td>
<td>44</td>
</tr>
<tr>
<td>substitute on CONS</td>
<td>38</td>
</tr>
<tr>
<td>substitute on hash-set</td>
<td>58</td>
</tr>
<tr>
<td>substitute on LIST</td>
<td>44</td>
</tr>
<tr>
<td>substitute on mutable-string</td>
<td>34</td>
</tr>
<tr>
<td>substitute on set</td>
<td>45</td>
</tr>
<tr>
<td>substitute on STRING</td>
<td>34</td>
</tr>
<tr>
<td>substitute-characters on STRING</td>
<td>124</td>
</tr>
<tr>
<td>substitute-characters on STRING</td>
<td>124</td>
</tr>
<tr>
<td>substitute-cons-tree</td>
<td>39</td>
</tr>
<tr>
<td>subtract on CONS</td>
<td>41</td>
</tr>
<tr>
<td>subtract on hash-set</td>
<td>59</td>
</tr>
<tr>
<td>subtract on LIST</td>
<td>45</td>
</tr>
<tr>
<td>subtype-of?</td>
<td>124</td>
</tr>
<tr>
<td>super-classes on class</td>
<td>124</td>
</tr>
<tr>
<td>surrogate-name?</td>
<td>124</td>
</tr>
<tr>
<td>surrogatify on string</td>
<td>124</td>
</tr>
<tr>
<td>surrogatify on surrogate</td>
<td>124</td>
</tr>
<tr>
<td>surrogatify on symbol</td>
<td>124</td>
</tr>
<tr>
<td>sweep on object</td>
<td>124</td>
</tr>
<tr>
<td>symbol-plist</td>
<td>62</td>
</tr>
<tr>
<td>symbol-property</td>
<td>62</td>
</tr>
<tr>
<td>symbol-to-surrogate</td>
<td>124</td>
</tr>
<tr>
<td>symbol-to-type</td>
<td>125</td>
</tr>
<tr>
<td>symbol-value</td>
<td>62</td>
</tr>
<tr>
<td>symbolize</td>
<td>62</td>
</tr>
<tr>
<td>system-default-value on slot</td>
<td>125</td>
</tr>
<tr>
<td>system-default-value on storage-slot</td>
<td>125</td>
</tr>
<tr>
<td>system-loaded-or-started-up?</td>
<td>125</td>
</tr>
<tr>
<td>system-loaded?</td>
<td>125</td>
</tr>
<tr>
<td>tan</td>
<td>29</td>
</tr>
<tr>
<td>terminate-program</td>
<td>125</td>
</tr>
<tr>
<td>test-property-demon</td>
<td>125</td>
</tr>
<tr>
<td>the-string on buffered-string</td>
<td>125</td>
</tr>
<tr>
<td>third on cons</td>
<td>37</td>
</tr>
<tr>
<td>third on list</td>
<td>42</td>
</tr>
<tr>
<td>third on mutable-string</td>
<td>33</td>
</tr>
<tr>
<td>third on string</td>
<td>33</td>
</tr>
<tr>
<td>third on vector</td>
<td>50</td>
</tr>
<tr>
<td>ticktock-difference</td>
<td>71</td>
</tr>
<tr>
<td>ticktock-resolution</td>
<td>71</td>
</tr>
<tr>
<td>time-add</td>
<td>71</td>
</tr>
<tr>
<td>time-divide</td>
<td>71</td>
</tr>
<tr>
<td>time-duration-to-string on time-duration</td>
<td>125</td>
</tr>
<tr>
<td>time-multiply</td>
<td>71</td>
</tr>
<tr>
<td>time-subtract</td>
<td>71</td>
</tr>
<tr>
<td>time-zone-format60</td>
<td>125</td>
</tr>
<tr>
<td>toggle-output-language</td>
<td>125</td>
</tr>
<tr>
<td>tokenize-string</td>
<td>125</td>
</tr>
<tr>
<td>top-level-stella-system-directory?</td>
<td>126</td>
</tr>
</tbody>
</table>
Function Index 142

V
value-setter on
  ABSTRACT-DICTIONARY-ITERATOR 129
vector 50
visible-modules 65
visible-surrogate? 61
visible-symbol? 60

W
warn 129
when 8
white-space-character? 31
with-input-file 67
with-network-stream 129
with-output-file 67
with-permanent-objects 129
with-system-definition 129
with-transient-objects 129
within-context 62
within-module 65
within-world 65
write-base-value 129
write-boolean 129
write-character 31
write-float 30
write-function-code 129
write-integer 30
write-integer-value 129
write-long-integer 129
write-method-code 130
write-mutable-string 35
write-string 35
wrapped-type-to-type 130
wrapper-value-type 130
write-global-variable-value 130
write-html-escaping-url-special-characters 130
write-native-variable-value 130
write-slot-value 130
writer on STORAGE-SLOT 130

X
xml-attribute-match? on XML-ATTRIBUTE 72
xml-attribute-match? on XML-GLOBAL-ATTRIBUTE 73
xml-attribute-match? on XML-LOCAL-ATTRIBUTE 73
xml-attribute? 72
xml-base-attribute? 130
xml-cdata-form? 72
xml-cdata? 72
xml-declaration-form? 130
xml-declaration? 72
xml-document-type-form? 130
xml-element-form? 130
xml-element-match? on XML-ELEMENT 72
xml-element? 72
xml-expressions ....................... 74
xml-global-attribute? .................. 130
xml-local-attribute-match? .............. 73
xml-local-attribute? .................. 130
xml-lookup-attribute ................. 73
xml-processing-instruction-form? ...... 131
xml-processing-instruction? .......... 131
xml-tag-case .......................... 73
xml-token-list-to-s-expression ...... 131

Y
y-or-n? .................................. 66
yes-or-no? ............................... 66
yield-define-stella-class ............... 131

Z
zero-pad-integer ....................... 131
zero? .................................. 28
zero? on INTEGER ........................ 131
zero? on LONG-INTEGER ............... 131
Variable Index

*  
*printpretty?* ........................................... 40
*printprettycode?* .................................... 41
*printreadably?* ....................................... 40

A  
any-key of ............................................ 46, 47, 51, 52, 53
any-value of ...... 36, 41, 45, 46, 47, 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 .................................................. 41

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)