GBBopen

GBBopen Reference Manual

Version 1.5

Dan Corkill

The GBBopen Project
http://GBBopen.org

October 30, 2013
16:10 EDT

Please help improve this manual by reporting any errors, inaccuracies, bugs, misleading or confusing statements, missing or unhelpful index entries, and typographical errors that you find. E-mail bug reports, comments, and suggestions to bugs@GBBopen.org. Your help is greatly appreciated and will be acknowledged.

The GBBopen Project
181 Pondview Drive
Amherst, Massachusetts 01002

GBBopen@GBBopen.org
http://GBBopen.org

This manual was produced using LaTeX and PDFLaTeX.

Contents

Acknowledgments

Many people have contributed comments, suggestions, design ideas, questions (and answers), bug reports, and code to GBBopen, and we appreciate their time and effort. Acknowledgment of some of their contributions here does not necessarily imply that any individual or the organizations with which they are affiliated endorse GBBopen or this documentation. Disclaimers aside, GBBopen users thank each of you!

Douglas Crosher ported GBBopen to Scieneer CL. Gary King worked on the initial Digitool MCL and OpenMCL porting efforts. Christian Lynbech performed the initial CMUCL port. Sam Steingold initiated the CLISP port. Vladimir Tzankov provided Portable Threads support for CLISP/MT.

Questioners, bug reporters, capability demanders, contributors, and great idea suggesters include: Pascal Costanza, Matthew Danish, Michael Hannemann, Susan Lander, Attila Lendvai, Wendall Marvel, Clayton Morrison, Beryl Nelson, Eric O'Connor, Zack Rubinstein, Bill St. Clair, Earl Wagner, Paul Werkowski, and Huzaifa Zafar.

Organizations Franz Inc. and LispWorks Ltd. provided (and continue to provide) Common Lisp licenses and technical support to the Project.

Some early design work for GBBopen was supported by DARPA's Information Exploitation Office (IXO) under contract MDA-972-02-C-0028 to Information Extraction & Transport, Inc.

Other efforts using GBBopen that have indirectly led to contributed GBBopen improvements and enhancements include: research supported by the "Fusion Based Knowledge for the Future Force" ATO program and the "Advanced REsearch Solutions - Fused Intelligence with Speed and Trust" program at the U.S. Army RDECOM CERDEC Intelligence and Information Warfare Directorate, Fort Monmouth, NJ, under contract W15P7T-05-C-P621; research on "Command & Control and Data Fusion Architectures" supported by DND Canada under contract W7701-4-2118; work on "Knowledgable Dynamic-Process Modeling and Execution" supported by Boeing and Infosys Technologies Limited; research on "A Multi-Agent Approach for Heterogeneous Persistent Surveillance" supported by Raytheon Intelligence & Information Systems; work on "Massive-Scale Representation and Reasoning" supported by GHX; and research supported by the AFRL "Advanced Computing Architecture" program, under contract FA8750-05-1-0039.

Legacy Contributors GBBopen builds upon concepts and ideas that were explored and refined in the UMass Generic Blackboard system and the commercial GBB product. The following people made significant contributions to those systems:

UMass Generic Blackboard System	**GBB Product**
Dan Corkill	Tony Carrico
Kevin Gallagher	Dan Corkill
Philip Johnson	Raymond de Lacaze
Kelly Murray	Kevin Gallagher
	Susan Lander
	Zack Rubinstein
	Suzanne Tromara

The UMass Generic Blackboard Project received research support from The National Science Foundation, the Defense Advanced Research Projects Agency, the Office of Naval Research, and Texas Instruments, Inc.

Introduction

GBBopen is a modern, high-performance, open source blackboard-system development environment that is based on the concepts that were explored and refined in the UMass Generic Blackboard system and the commercial GBB product. GBBopen is not, however, a clone or updated version of either system. The GBBopen Project is applying the knowledge and experience gained with these earlier tools to create a new generation of blackboard-system capabilities and make them freely available to a wide audience.

GBBopen is structured for high-performance and scalability while maintaining flexibility and adaptability to changes in representation, knowledge-source (KS) components, and control strategies. Multi-dimensional abstraction of blackboard components ("space instances"), blackboard objects ("unit instances"), and proximity-based retrieval patterns is used to provide a semantically meaningful separation of blackboard-repository storage mechanisms from KS and control code. This separation allows storage and search strategies and optimizations to change dynamically as well as to be adapted to a broad range of application areas. GBBopen also provides highly efficient and extensible event primitives that form the foundation for fast, yet effective, opportunistic control reasoning.

At the implementation level, GBBopen is designed as a smooth extension of Common Lisp, providing all the advantages of a rich, dynamic, reflective, and extensible programming language to the blackboard-system architects and component writers. These capabilities are crucial in building complex blackboard-based applications where object representations, knowledge sources (KSs), and control mechanisms will change during development and over the operational lifetime of the blackboard application. GBBopen is tightly integrated with CLOS (the Common Lisp Object System) and provides additional blackboard-specific object mechanisms via the Metaobject Protocol.

The open-source licensing of GBBopen provides a number of important benefits:

- A modular, open-source reference implementation of blackboard-system infrastructure that serves as a basis for research and development activities.
- The availability of source code and the right to modify it enables unlimited improvement and enhancement of the software. It also makes it possible to port the code to new hardware and software, to adapt it to changing conditions, and to reach a detailed understanding of how GBBopen works. Source code availability also makes it much easier to isolate and fix bugs.
- The right to redistribute improvements and extensions to the GBBopen source code.
- The right to use the software.
- There is no single entity on which the future of the GBBopen software depends. This is particularly important given the highly specialized nature of blackboard-system software and the lack of multiple implementations.
- GBBopen supports alternative and additional GBBopen modules for use in research and experimentation.

1 Starting Up

GBBopen is packaged with its own module system (see page 21) that supports compiling and loading GBBopen components. The Module Manager Facility is designed for ease of use and for simplicity in porting to Common Lisp implementations. For example, to compile all GBBopen modules, you only need to evaluate the following forms within your Common Lisp environment. First load GBBopen's *<install-directory>*/initiate.lisp file:

```
cl-user> (load "<install-dir>/initiate.lisp")
;; Loading <install-dir>/initiate.lisp
;; GBBopen is installed in <install-dir>
;; Your "home" directory is <homedir>
;;     Loading <install-dir>/extended-repl.lisp
;;     Loading <install-dir>/commands.lisp
;;     Loading <install-dir>/gbbopen-modules-directory.lisp
;; No shared module command definitions were found in <install-dir>/gbbopen-modules/.
;; No personal module command definitions were found in <homedir>/gbbopen-modules/.
#P"<install-dir>/initiate.lisp"
cl-user>
```

Notice that some additional GBBopen initialization files have been loaded by this file as well as GBBopen's top level read-eval-print loop (REPL) command definitions.

> **Windows users**
>
> If you are running GBBopen on Windows, note that backslash is the escape character in the standard Common Lisp reader; it causes the next character to treated as a normal character rather than as having any special syntactic characteristics. So, each backslash in a file-name string must be entered as two backslash characters. For example:
>
> ```
> > (load "c:\\GBBopen\\initiate.lisp")
> ```
>
> Many Common Lisp implementations also support Unix-style regular slashes in Windows file names (e.g., `"c:/GBBopen/initiate.lisp"`), which can be particularly convenient when typing file names in the REPL.

Top-level REPL commands

The files loaded by *<install-directory>*/initiate.lisp add some handy REPL keyword commands when running Allegro CL, Clozure CL, CMUCL, ECL, LispWorks, SBCL, and Scieneer CL users. Some interaction interfaces, such as SLIME, use their own REPL rather than the top-level listener provided by the Common Lisp implementation and, therefore, may not support keyword REPL command processing. GBBopen provides a Swank extension to the SLIME REPL that supports REPL keyword commands.

Built-in and GBBopen REPL commands are defined in the file *<install-directory>*/commands.lisp. REPL commands, including those that have arguments, are specified using a non-list ("spread") representation. For example:

```
> :gbbopen-test :propagate :create-dirs
   ...
>
```

Some Common Lisp implementations (Clozure CL, LispWorks, and SBCL) and the SLIME REPL interface, also support REPL commands in non-spread (list) form in addition to the spread notation. For example:

```
> (:gbbopen-test :propagate :create-dirs)
   ...
>
```

Equivalent functions in the `:common-lisp-user` package are always defined for each REPL command, and these functions can be used in place of REPL keyword-command processing.

Compiling all GBBopen modules

Once *<install-dir>*`/initiate.lisp`, all GBBopen modules can be compiled by entering the following REPL command:

```
cl-user> :compile-gbbopen
;; Loading <install-dir>/startup.lisp
;; GBBopen is installed in <install-dir>
;; Your "home" directory is <homedir>
;;  Loading <install-dir>/source/module-manager/module-manager-loader.lisp
;; Loading <install-dir>/<platform-dir>/module-manager/module-manager.lisp
   ...
;; Loading <install-dir>/modules.lisp
;; No shared module definitions were found in <install-dir>/gbbopen-modules/.
;; No personal module definitions were found in <homedir>/gbbopen-modules/.
   ...
;;; GBBopen modules compilation completed.
[Common Lisp will exit]
```

GBBopen should compile all GBBopen modules and then exit Common Lisp without error.

As can be seen from the file-loading messages, the `startup.lisp` file that is loaded by the `:compile-gbbopen` command loads a bootstrap loader file for the Module Manager Facility (see page 21) (the file `source/module-manager/module-manager-loader.lisp`). This bootstrap file then loads the Module Manager files (`source/module-manager/module-manager.lisp` and `source/module-manager/module-manager-user.lisp`) followed by the module definitions for all GBBopen modules (contained in the file `source/modules.lisp`). Then the Module Manager is used to compile all GBBopen modules.

ASDF, clbuild, and Quicklisp users

GBBopen's Module Manager Facility provides an interface that allows ASDF (and therefore clbuild and Quicklisp) to play nice with Module Manager. If you installed GBBopen using clbuild or Quicklisp, ASDF has been informed of GBBopen's *<install-dir>*`/gbbopen.asd` system-definition file. Otherwise, to use ASDF to set up GBBopen, you must add the `gbbopen.asd` file to ASDF's Registry manually. Then, instead of loading the *<install-dir>*`/initiate.lisp` file, the Module Manager and GBBopen module definitions can be loaded using ASDF by entering:

```
cl-user> (asdf:operate 'asdf:load-op :gbbbopen)
;; Loading <install-dir>/initiate.lisp
;; GBBopen is installed in <install-dir>
;; Your "home" directory is <homedir>
   ...
;; Loading <install-dir>/startup.lisp
;; GBBopen is installed in <install-dir>
;; Your "home" directory is <homedir>
;;  Loading <install-dir>/source/module-manager/module-manager-loader.lisp
;; Loading <install-dir>/<platform-dir>/module-manager/module-manager.lisp
   ...
;; Loading <install-dir>/modules.lisp
;; No shared module definitions were found in
<install-dir>/gbbopen-modules/.
;; No personal module definitions were found in <homedir>/gbbopen-modules/.
   ...
;; Defining an ASDF defsystem for each Module Manager module...
cl-user>
```

or when ASDF is integrated with Common Lisp's **require**:

```
cl-user> (require :gbbopen)
;; Loading <install-dir>/initiate.lisp
;; GBBopen is installed in <install-dir>
;; Your "home" directory is <homedir>
   ...
;; Loading <install-dir>/startup.lisp
;; GBBopen is installed in <install-dir>
;; Your "home" directory is <homedir>
;;  Loading <install-dir>/source/module-manager/module-manager-loader.lisp
;; Loading <install-dir>/<platform-dir>/module-manager/module-manager.lisp
   ...
;; Loading <install-dir>/modules.lisp
;; No shared module definitions were found in
<install-dir>/gbbopen-modules/.
;; No personal module definitions were found in <homedir>/gbbopen-modules/.
   ...
;; Defining an ASDF defsystem for each Module Manager module...
cl-user>
```

Notice that loading the *<install-dir>*`/initiate.lisp` file loaded only GBBopen's REPL command processing extensions, global REPL command definitions, and module-directory processing into Common Lisp–the Module Manager is not loaded until it is needed (such as when we performed the `:gbbopen-user` REPL command). The ASDF `:gbbopen` "system" start up, on the other hand, must also load the Module Manager and module definitions, as they are required in order to define an ASDF system for each Module Manager module.

Personal `gbbopen-init.lisp` file

If a `gbbopen-init.lisp` file (source or compiled) is present in the user's "home" directory (as defined by **user-homedir-pathname**), it is loaded by the *<install-directory>*`/startup.lisp` file after the Module Manager Facility (see page 21) and definitions have been loaded. A personal *<homedir>*`/gbbopen-init.lisp` file is a handy mechanism for defining user-specific modules, application modules, GBBopen parameters, and other personalizations.

Here is a simple example of a personal *<homedir>*`/gbbopen-init.lisp` file that defines a root directory named `:my-app-root` and the module `:my-app`:

```
;;;; -*- Mode:Common-Lisp; Package:CL-USER -*-

(in-package :cl-user)

(module-manager:define-root-directory :my-app-root
    (make-pathname :directory "~/my-app"))

(module-manager:define-module :my-app
  (:requires :gbbopen-user)
  (:directory :my-app-root)
  (:files "my-file"))

;;; ============================================
;;;   End of File
;;; ============================================
```

For shared or more substantial application modules, we recommend a packaging convention that mirrors that of GBBopen. This will be discussed shortly in conjunction with using a personal `gbbopen-modules` directory (see page 7).

Personal `gbbopen-commands.lisp` file

You can create your own REPL commands by defining them in a personal `gbbopen-commands.lisp` file. If a `gbbopen-commands` file (source or compiled) is present in the user's "homedir" home directory (as defined by **user-homedir-pathname**), it is loaded automatically by GBBopen's `initiate.lisp` file. For example, the following personal `gbbopen-commands.lisp` file defines a REPL command (named `:my-app`) and an equivalent function (`cl-user::my-app`) for compiling and loading the `:my-app` module that was defined above:

```
;;;; -*- Mode:Common-Lisp; Package:CL-USER -*-

(in-package :cl-user)

(define-repl-command :my-app (&rest options)
  "Compile and load my GBBopen application module"
  (startup-module :my-app options :gbbopen-user))

;;; ============================================
;;;   End of File
;;; ============================================
```

Personal `gbbopen-modules` directory

Although a personal `gbbopen-init.lisp` file can be used to define the module `:my-app`, GBBopen provides an alternative mechanism that is even more convenient if you develop, share, or use a number of modules or applications.

If a `gbbopen-modules` directory is present in the user's home directory (as defined by `user-homedir-pathname`), it is expected to consist of directories each containing an individual GBBopen application or library. Although these application directories can be placed directly in the `gbbopen-modules` directory, it is generally more convenient to use a symbolic link (or a "pseudo symbolic-link file" on Windows) to point to the actual application directory, where ever it is located. For example, an application can be provided to a number of users by creating a symbolic link to the actual application directory in each user's `gbbopen-modules` directory.

Each application directory contains:

- a `modules.lisp` file that contains module definitions (loaded after the personal `gbbopen-init.lisp` file if there is one in the user's "homedir")
- a directory named `source` containing all the source files for the module or application
- an optional `commands.lisp` file that specifies REPL commands for the module (loaded after the personal `gbbopen-commands.lisp` file if there is one in the user's "homedir")
- any additional directories or files useful to the application

We highly recommend following this packaging convention, which mirrors that of GBBopen itself. It is very easy to create, use, and share modules defined in this way by placing symbolic links to the module directories in your personal `gbbopen-modules` directory. Windows, unfortunately, is the exception to this as Windows does not provide symbolic links. GBBopen users running on Windows must create a text file with the file extension `.sym` (that contains the target directory path as its sole line) as a stand-in for the symbolic link.

Installation-wide `shared-gbbopen-modules` directory

There is also an *<install-directory>*`/shared-gbbopen-modules` directory. As with a user's `gbbopen-modules` directory (discussed above), the `shared-gbbopen-modules` directory is assumed to contain symbolic links (or "pseudo-symbolic-link" files on Windows) to individual GBBopen module directory trees.

This is the recommended mechanism for installation-wide managing and sharing of modules and applications.

GBBopen Hyperdoc

Convenient access to a local copy of the GBBopen Hyperdoc manual from Common Lisp is available by using the **browse-hyperdoc** function; part of the `:os-interface` (see page 317) module. The browser used by **browse-hyperdoc** is specified by the value of `*preferred-browser*`. A different value can be specified in either your Common Lisp initialization file or, preferably, in your personal `gbbopen-init.lisp` file. Changing the default setting in GBBopen's `startup.lisp` is not recommended.

Emacs access to the GBBopen Hyperdoc is provided by *<install-directory>*`/browse-hyperdoc.el`. This file defines the interactive Emacs command `browse-hyperdoc` and binds it to `META-?`. To enable this command, load *<install-directory>*`/browse-hyperdoc.el` from your `.emacs` initialization file.

If you already use the `hyperspec.el` utility (included with SLIME and ILISP distributions, but

usable on its own), the Emacs `browse-hyperdoc` command will automatically defer to the Common Lisp HyperSpec when given a non-GBBopen entity. You can also download and install a local copy of the Common Lisp HyperSpec for use without a network connection. In this case, set the value of `common-lisp-hyperspec-root` in your `.emacs` initialization file to point to your local copy of the HyperSpec. For example:

```
(setf common-lisp-hyperspec-root "file:/usr/local/CLHS/")
```

Highly recommended!

Starting-up entities

Descriptions of entities related to installing and customizing GBBopen follow.

ignored-gbbopen-modules-directory-subdirectories [*Variable*]

Purpose
Specify subdirectory names to be ignored by GBBopen's `gbbopen-modules` directory processing.

Package
`:common-lisp-user`

Module
Defined in `gbbopen-modules-directory.lisp`

Value type
A list

Initial value
`(".svn")`

Description
Subdirectories named in `*ignored-gbbopen-modules-directory-subdirectories*` are ignored when processing shared and personal `gbbopen-modules` directories for command and module definitions.

Example
Add `"CVS"` to the list of subdirectories that are ignored by GBBopen's `gbbopen-modules` directory processing:

```
> (push "CVS" *ignored-gbbopen-modules-directory-subdirectories*)
("CVS" ".svn")
>
```

gbbopen-modules-directory-verbose [*Variable*]

Purpose
Controls whether the processing of individual modules is shown during `gbbopen-modules` directory processing.

Package
`:common-lisp-user`

Module
Defined in `gbbopen-modules-directory.lisp`

Value type
A generalized boolean

Initial value
`t`

preferred-browser [*Variable*]

Purpose
Specify the preferred browser program.

Package
`:common-lisp-user`

Module
Defined in `startup.lisp`

Value type
A string

Initial value
(see the first example below)

Description
To change the preferred browser, set the value of `*preferred-browser*` in either your Common Lisp initialization file or your personal `gbbopen-init.lisp` file. Because `startup.lisp` is under Subversion source control, changing `startup.lisp` directly is not recommended.

See also
browse-hyperdoc (page 318)

Examples
Here is the setting that is made in `startup.lisp`:

```
  (defvar *preferred-browser*
      ;; On Mac OSX we defer to the OS default browser:
      #+(or macosx darwin)
      "open"
      ;; LispWorks (non-Windows) and SBCL do not search PATH for programs, so
      ;; the path must be explicitly included in the preferred browser
setting:
      #+(or (and lispworks (not win32)) sbcl) "/usr/bin/firefox"
      #-(or macosx darwin (and lispworks (not win32)) sbcl) "firefox")
```

Specify a different browser (in a personal `gbbopen-init.lisp` file) on Linux machines:

```
  (in-package :common-lisp-user)

  #+linux
  (setf *preferred-browser*
    ;; Lispworks (non-Windows) and SBCL do not search PATH for programs, so
    ;; the path must be explicitly included in the preferred browser setting:
     #+(or lispworks sbcl) "/usr/bin/opera"
     #-(or lispworks sbcl) "opera")
```

Note
LispWorks (non-Windows platforms) and SBCL do not perform a `PATH` search for programs, so the browser-program path must be explicitly included in the preferred browser setting.

sym-file-verbose [*Variable*]

Purpose

Controls whether the mapping from `*.sym` pseudo-symbolic-link files to target module directories is shown during `gbbopen-modules` directory processing.

Package

`:common-lisp-user`

Module

Defined in `gbbopen-modules-directory.lisp`

Value type

A generalized boolean

Initial value

`nil`

Example

Show `*.sym` pseudo-symbolic-link file to target module-directory mapping `gbbopen-modules` directory processing on startup module loading (after *<install-directory>*`/initiate.lisp` has been loaded):

```
  > (setf *sym-file-verbose* 't)
  t
  > :startup
  ;; Loading <install-dir>/startup.lisp
  ;; GBBopen is installed in <install-dir>
  ;; Your "home" directory is <homedir>
  ;;  Loading <install-dir>/source/module-manager/module-manager-loader.lisp
  ;;   Loading <install-dir>/source/module-manager/module-manager.lisp
  ;;     Loading <install-dir>/source/module-manager/module-manager-user.lisp
  ;;  Loading <install-dir>/source/modules.lisp
  ;; No shared module definitions were found in
<install-dir>/shared-gbbopen-modules/.
  ;; Loading personal module definitions from <homedir>/gbbopen-modules/...
  ;; Pseudo (*.sym) link <homedir>/gbbopen-modules/an-app.sym -->
/usr/local/an-app/
  ;;  Loading /usr/local/an-app/modules.lisp
  T
  >
  >
```

define-repl-command *command-name-spec lambda-list* [*declaration** | *documentation*] *form** [*Macro*]

Purpose

Define a top-level REPL (read-eval-print loop) command.

Package

`:common-lisp-user` (also imported into and exported from `:module-manager`)

Module

Defined in `extended-repl.lisp`

Arguments

command-name-spec	A *command-name* or a list `(`*command-name option**`)`
lambda-list	A lambda-list
declaration	A declare expression (not evaluated)
documentation	A documentation string (not evaluated)
form	A form

Detailed syntax

option ::= `:add-to-native-help` | `:no-help` | `:no-cl-user-function`

Terms

command-name A keyword symbol naming the command

Description

The arguments to the command are not evaluated before the command is invoked; it is up to the command to perform argument evaluation if needed (see the example, below).

If the `:add-to-native-help` option is specified, then the command and its *documentation* string are added to the Common Lisp implementation's primary REPL help; otherwise the command is only added to the extended-REPL commands help that is displayed by the `:commands` REPL command. (Not all Common Lisp implementations distinguish primary and secondary command-help levels.)

If the `:no-help` option is specified, then the command is not added to either the primary or secondary help displays.

An equivalent function in the `:common-lisp-user` package is normally defined for the *command-name* keyword command, and this function can be used when REPL keyword-command processing is not fully supported. However, a `:common-lisp-user` functional version of the command is not defined if the `:no-cl-user-function` option is specified.

Documentation is a documentation string to be associated with the REPL command *command-name*.

See also

startup-module (page 16)

Examples

Define a REPL command named `:ds` to be a handy shortcut to the Common Lisp `describe` function:

```
(define-repl-command (:ds :add-to-native-help) (obj)
  "Describe object"
  (describe (eval obj)))
```

Define a REPL command named `:my-app` that compiles and loads the module `:my-app` and sets the current package to the `:gbbopen-user` package:

```
(define-repl-command :my-app (&rest options)
  "Compile and load my GBBopen application module"
  (startup-module :my-app options :gbbopen-user))
```

---

define-repl-command

funcall-in-package *symbol package-name* `&rest` *args* [*Function*]

Purpose

Apply a function to arguments when the package containing the function may not exist until execution time.

Package

`:common-lisp-user` (not exported)

Module

Defined in `initiate.lisp` (or in `startup.lisp`, if `initiate.lisp` is not used to initiate GBBopen)

Arguments

symbol	A symbol
package-name	A package name
args	Arguments to the function

Description

This function provides a convenient means of applying a function to arguments, when the package containing the function may not exist until execution time. The function to be applied is determined at runtime by using the name of *symbol* to find the symbol with the same name in *package* and using it as the function designator.

Example

Define a REPL command named `:my-app` that compiles and loads the module `:my-app`, sets the current package to the `:gbbopen-user` package, and then calls the function `startup` in the package `:my-app-package` that is defined when the `:my-app` module is loaded:

```
(define-repl-command :my-app (&rest options)
  "Compile and load my GBBopen application module"
  (startup-module :my-app options :gbbopen-user)
  ;; Call the startup function:
  (funcall-in-package '#:startup :my-app-package))
```

startup-module *module-name options* `&optional` *package-name* [*Function*]

Purpose

Compile and load a GBBopen module, even if the Module Manager Facility (see page 21) is not yet loaded, and optionally set the current package.

Package

`:common-lisp-user` (not exported)

Module

Defined in `initiate.lisp`

Arguments

module-name A keyword symbol naming a module

option Any of the following keywords:

`:create-dirs`	Creates any needed directories that are missing in the compiled-file tree
`:noautorun`	Binds ***autorun-modules*** to `nil` during compilation and loading
`:nopatches`	Do not compile or load any patches
`:nopropagate`	Cancels (overrides) a specified :propagate option
`:patches-only`	Do not compile or load/reload any non-patch files (binds ***patches-only*** to `t` during compilation and loading)
`:print`	Incrementally prints information during compilation and loading
`:propagate`	Applies the specified options to all required modules
`:recompile`	Compiles files even if the existing compiled file is newer than the source file
`:reload`	Loads files even if they are already loaded
`:source`	Loads from the source file even if the existing compiled file is newer than the source file (implies `:reload`)

package-name A package name (default is `nil`)

Errors

Module *module-name* has not been defined.
A relative directory specification contains a circularity.

Description

This function bootstraps GBBopen loading, if needed, before calling **compile-module** on *module-name* with *options*. Then, unless *package-name* is `nil`, the current package is set to the package named by *package-name*. The named package does not need to be defined before calling **startup-module**, but it must be defined at the conclusion of module compilation and loading.

Startup-module adds the option `:propagate` to the supplied *options*. If this propagation behavior is not desired, the `:nopropagate` option can be specified to override (cancel) propagation.

See also

compile-module (page 26)
define-module (page 30)
define-repl-command (page 13)

Example

Define a REPL command named `:my-app` that compiles and loads the module `:my-app` and sets the current package to the `:gbbopen-user` package:

```
(define-repl-command :my-app (&rest options)
  "Compile and load my GBBopen application module"
  (startup-module :my-app options :gbbopen-user))
```

---

startup-module

with-system-name (*system-name*) *form** ⇒ *result** [*Macro*]

Purpose

Associate *system-name* with REPL commands, directory definitions, and module definitions defined in *forms*.

Package

`:common-lisp-user` (also imported into and exported from `:module-manager`)

Module

Defined in `extended-repl.lisp`

Arguments

system-name	A keyword symbol identifying a system
form	A form
results	The values returned by evaluating the last *form*

Returns

The values returned by evaluating the last *form*.

See also

define-relative-directory (page 33)
define-root-directory (page 35)
define-module (page 30)
define-repl-command (page 13)

Examples

Define a REPL command named `:my-app` associated with system `:my-app` that compiles and loads the module `:my-app` and sets the current package to the `:gbbopen-user` package:

```
(with-system-name (:my-app)
  (define-repl-command :my-app (&rest options)
    "Compile and load my GBBopen application module"
    (startup-module :my-app options :gbbopen-user)))
```

Now, show the commands associated with system `:my-app`:

```
> :commands :my-app
Command           Description
-------           -----------
:my-app           Compile and load my GBBopen application module
>
```

Now, undefine all the commands, directories, and module definitions associated with system `:my-app`:

```
> :undefine-system :my-app
Really undefine commands, directories, & modules of :my-app?  y
;; System :my-app undefined.
> :commands :my-app
;; System :my-app was not found.
>
```

Notes

A developer will typically wrap the **define-repl-command** forms in an application's `commands.lisp` file with a single **with-system-name** macro and the definitions in the application's `modules.lisp` file with a second **with-system-name** macro.

---

with-system-name

2 Module Manager Facility

The Module Manager Facility provides a lightweight and easy to use mechanism for compiling and loading module files. The facility keeps track of the dependencies between modules and the modules that have been compiled and loaded. The Module Manager Facility was designed to be easy to use, yet it is powerful enough to manage substantial software projects, each operating on multiple hardware platforms and Common Lisp implementations. Other, more complex, **defsystem** packages, such as (ASDF), are available, but their complexity makes defining and maintaining correct system definitions difficult.

The `:module-manager` module is automatically loaded by the GBBopen *<install-directory>*`/startup.lisp` file (by the `module-manager-loader.lisp` file located in the `module-manager` subdirectory). If a `gbbopen-init.lisp` file (source or compiled) is present in the user's "home" directory, it is loaded immediately following the loading of the Module Manager Facility. A personal *<homedir>*`/gbbopen-init.lisp` file is very useful for defining GBBopen parameters and other personalizations (see Section 1).

Key concepts

The Module Manager Facility incorporates several important concepts:

- Dependencies are maintained among modules, rather than among individual files. Defining compilation and loading dependencies at the file level (and maintaining those dependencies as applications and libraries evolve) is both complex and error prone. Instead, each Module Manager module definition specifies a total ordering of any other modules that it requires (directly) as well as a total ordering of the individual files that comprise the module. (It is an error for a file to be associated with more than one module.)
- A total ordering of all the individual files that are required and used by a module is generated from the module definitions. The Module Manager Facility expands a module's specified required modules recursively to obtain a fully expanded sequence of all the modules that are required (directly or indirectly) by the module. The individual files associated with each module in this fully expanded sequence of required modules define the sequence of all the individual files that must be possibly compiled and/or loaded as part of compiling or loading the module.
- Recompilation and reloading dependencies are specified as options associated with individual files. For example, if the option `:forces-recompile` is specified for a file and that file needs to be recompiled, all files later in the total file sequence need to be recompiled and reloaded. This simple, "somewhat aggressive" recompilation mechanism replaces complex definitions of pairwise dependencies between individual files (and the risk of missed and no-longer-appropriate dependencies), but at the cost of some unneeded recompilation. It also simplifies handling dependencies that exist between some files in a requiring module and files in ones of that module's required modules when each module is supported by a different developer/maintainer. A developer only needs to consider whether a file includes forms (such as macro definitions and compiler transforms) that may effect the compilation and/or loading of downstream files (whether or not those files are known to the developer). With today's fast Common Lisp compilers, some unnecessary recompilation is a small price to pay for easier and more likely to be correct dependency specifications.
- The fully expanded required module sequences of all modules (and therefore the compilation and loading sequence of files for all modules) must be consistent. This requirement is checked by the Module Manager and ensures that the ordering of file compilation and loading will be maintained no matter what module (or modules) is being loaded. Ordering consistency is important in avoiding unnecessary recompilation of unchanged files if different modules were allowed to specify their sequence of overlapping required modules differently. Although

required-module ordering consistency is a very strong constraint—especially when independently development modules use common libraries as required modules—in practice, existing required-module orderings quickly eliminate most “otherwise arbitrary” required-module orderings. Where two required modules are not constrained as to their pairwise ordering, the convention of specifying the module that is earlier alphabetically first is recommended.

- Separate compiled-file directory trees are created for each Common Lisp implementation and platform. These compiled-file directories mirror the directory structure of the original source-file directory tree of a system. This separation makes packaging and distributing compiled files for one or more platforms (with or without the source tree) simple by including only the desired directory trees.
- Freezing a release (and providing patches later, if needed) are important phases of the release life cycle of some applications. Support for freezing and patches that are in keeping with the above concepts is provided explicitly by Module Manager operators.

Stand-alone usage

To bootstrap the Module Manager Facility stand-alone (separate from the GBBopen Project software tree), do the following:

- create a “root” directory to contain the Module Manager software tree (for example: `$ mkdir my-tree`)
- create the Module Manager portion of the source tree in the newly created “root” directory (for example: `$ cd my-tree ; mkdir -p source/module-manager`)
- copy the `module-manager-loader.lisp`, `module-manager.lisp`, `module-manager-user.lisp` files into the `source/module-manager` directory
- start your Common Lisp implementation and then load the `module-manager-loader.lisp` file
  ```
  > (load "my-tree/source/module-manager/module-manager-loader")
  ```
- compile the `:module-manager` and `:module-manager-user` modules:
  ```
  > (module-manager:compile-module :module-manager-user :create-dirs :propagate)
  ```

After performing the above steps, the Module Manager can be used stand-alone by loading `source/module-manager-loader.lisp` as part of your Common Lisp initialization.

Module Manager entities

Descriptions of Module Manager entities follow.

automatically-create-missing-directories [*Variable*]

Purpose

Controls whether any directories that are missing in the compiled-file tree should be created when needed.

Package

`:common-lisp-user` (also imported into and exported from `:module-manager`)

Module

`:module-manager`

Value type

A generalized boolean

Initial value

`t`

Description

By default, the Module Manager will create any needed directories in the compiled-file tree automatically during module compilation. For tighter control over new directory creation, ***automatically-create-missing-directories*** can be set to `nil`, causing **compile-module** to signal a continuable error during compilation if a needed directory is missing in the compiled-file tree. When ***automatically-create-missing-directories*** has been set to `nil`, the **compile-module** option `:create-dirs` provides a convenient way to bind ***automatically-create-missing-directories*** to true during compilation of a specified module.

See also

compile-module (page 26)

Example

Have the Module Manager generate a continuable error during module compilation when a needed directory in the compiled-file tree is missing (and the compile-module option `:create-dirs` was not specified when compiling the module needing that missing directory):

```
(setf common-lisp-user::*automatically-create-missing-directories* nil)
```

autorun-modules [*Variable*]

Purpose
Indicates whether a module should evaluate autorun forms when it is loaded.

Package
`:common-lisp-user` (also imported into and exported from `:module-manager`)

Module
`:module-manager`

Value type
A generalized boolean

Initial value
True

Description
The value of ***autorun-modules*** can be used to conditionally evaluate forms when module files are loaded. By default, the value of ***autorun-modules*** is set to true, but it can be set or bound to `nil` to disable conditional autorun forms. The compile-module or load-module option `:noautorun` can also be specified to bind ***autorun-modules*** to `nil` during module compilation and loading.

See also
compile-module (page 26)
load-module (page 47)
load-module-file (page 49)

Example
Conditionally evaluate the `http-test` function when the file `http-test.lisp` that includes the following form is loaded:

```
(when common-lisp-user::*autorun-modules*
  (http-test "GBBopen.org" 80))
```

patches-only [*Variable*]

Purpose

Controls whether regular files (non-patch files) in modules are compiled and loaded by the Module Manager.

Package

`:common-lisp-user` (also imported into and exported from `:module-manager`)

Module

`:module-manager`

Value type

A generalized boolean

Initial value

`nil`

Description

By default, ***patches-only*** is set to `nil`, but it can be set to true to instruct **compile-module** and **load-module** to bypass compiling and loading regular files (non-patch files) in modules. By setting ***patches-only*** to true after loading an application, the image is "frozen" so that only patches will be applied by subsequent uses of **compile-module** or **load-module**.

See also

compile-module (page 26)
load-module (page 47)

Example

Compile, load, and freeze the GBBopen application defined by module `:my-app`:

```
> (progn (compile-module :my-app :propagate)
         (setf *patches-only* 't))
t
>
```

compile-module *module-name* `&rest` *options* [*Function*]

Purpose
Compiles and loads the files in a module.

Package `:module-manager`

Module `:module-manager`

Arguments
module-name A keyword symbol naming a module

option Any of the following keywords:

`:create-dirs`	Creates any needed directories that are missing in the compiled-file tree
`:noautorun`	Binds ***autorun-modules*** to `nil` during compilation and loading
`:nopatches`	Do not compile or load any patches
`:nopropagate`	Cancels (overrides) a specified :propagate option
`:patches-only`	Do not compile or load/reload any non-patch files (binds ***patches-only*** to `t` during compilation and loading)
`:print`	Incrementally prints information during compilation and loading
`:propagate`	Applies the specified options to all required modules
`:recompile`	Compiles files even if the existing compiled file is newer than the source file
`:reload`	Loads files even if they are already loaded
`:source`	Loads from the source file even if the existing compiled file is newer than the source file (implies `:reload`)

Errors
Module *module-name* has not been defined.
A relative directory specification contains a circularity.
Module *module-name* or one of its required modules has not been loaded and ***patches-only*** is true.

Description
These file options, when specified for individual files in the module definition (see **define-module**), have the following effects (overriding the behavior of *options* supplied to **compile-module**):

`:developing`	If the patch file has changed, recompile and reload it (patch files only)
`:forces-recompile`	If the file has changed, recompile and reload all subsequent files and modules
`:noload`	Compile, but do not load the file
`:recompile`	Always recompile the file
`:reload`	Always reload the file
`:skip-recompile`	Skip recompiling the file if it has been loaded already (takes precedence over other recompilation decisions)
`:source`	Do not compile the file (load the source instead)

See also

:disable-compiler-macros (page 62)
:full-safety (page 63)
automatically-create-missing-directories (page 23)
autorun-modules (page 24)
patches-only (page 25)
define-module (page 30)
load-module (page 47)
load-module-file (page 49)
startup-module (page 16)

Example

Compile and load the GBBopen User module and all its required modules, creating new compilation directories if they do not exist already:

```
(compile-module :gbbopen-user :propagate :create-dirs)
```

REPL Note

Compile-module can be invoked using the REPL command:

```
:cm [module-name [option*]]
```

which remembers the last specified *module-name* and *options* as default values for the command.

---

compile-module

continue-patch *form** [*Macro*]

Purpose
Continue the definition of a multiple top-level form patch to a module.

Package `:module-manager`

Module `:module-manager`

Arguments
form A form

Errors
A patch has not been started with **start-patch**.

Description
The **continue-patch** macro, along with **start-patch** and **finish-patch**, can be used to define a patch as multiple top-level forms in the same patch file if a patch cannot be defined as a single **patch** form.

See also

allow-redefinition (page 65)
define-module (page 30)
describe-module (page 37)
describe-patches (page 39)
finish-patch (page 40)
get-patch-description (page 44)
parse-date (page 181)
patch (page 53)
patch-loaded-p (page 55)
start-patch (page 57)
undefmethod (page 128)

Example
Define a more complex patch (in a file named `my-app-p002.lisp` in the `patches` subdirectory of the module):

```
(start-patch (2 "06-23-08"
                :author "Corkill"
                :description "A more complex patch example")
    (printv "More complex example patch started!"))

(eval-when (:compile-toplevel)
  (continue-patch
   (printv "Defining compile-time-only-macro-for-patch...")
   (defmacro compile-time-only-macro-for-patch (x)
     `',x)))

(eval-when (:compile-toplevel :load-toplevel :execute)
```

```
    (continue-patch
     (printv "Defining macro-for-patch at compile & load time...")
     (defmacro macro-for-patch (x)
       `',x)))

  (continue-patch
   (printv "Using macro-for-patch at load time...")
   (macro-for-patch abc))

  (eval-when (:compile-toplevel :load-toplevel :execute)
    (continue-patch
     (printv "Using macro-for-patch at compile & load time...")
     (macro-for-patch xyz)))

  (eval-when (:compile-toplevel)
    (continue-patch
     (printv "Using compile-time-only-macro-for-patch...")
     (compile-time-only-macro-for-patch abc)))

  (finish-patch
   (printv "More complex example patch finished!"))
```

---

continue-patch

define-module *module-name [documentation] module-option** [*Macro*]

Purpose

Defines a module to the Module Manager Facility.

Package :module-manager

Module :module-manager

Arguments

module-name A keyword symbol naming a module
documentation A documentation string (not evaluated)
module-options See below

Errors

The :requires module option specifies a fully expanded required-module order that contains a circularity.
The :requires module option specifies a fully expanded required-module order that conflicts with the fully expanded required-module order in a previously defined module.

Detailed syntax

module-option ::= (:requires *module-name**) |
(:directory *directory-specifier*) |
(:files *file-specifier**) |
(:patches *file-specifier**)
directory-specifier ::= *root-or-relative-directory subdirectory**
file-specifier ::= *file-name* |
(*file-name file-option**)
file-option ::= :recompile | :reload | :source | :forces-recompile | :noload

Terms

root-or-relative-directory A keyword naming a root or relative directory, a keyword naming another module, or nil, indicating that the module is rooted at the *load-truename* value in effect when the module definition is loaded
subdirectory A string naming a subdirectory
file-name A string naming a file
module-name A keyword symbol naming a module

Description

The *module-options* :requires, :directory, :files, and :patches can be specified in any order, but at most one of each is allowed.

If a :directory module option is not specified, an implicit root directory for the module (at the *load-truename* of the file containing the **define-module** form) is used. If the :directory module option specifies a keyword naming another module, the directory specification of the named module is used as the base (root or relative) directory for the module that is being defined. If both a module and a root or relative directory definition have the same name, the directory definition takes precedence over the module directory specification.

The :requires module option specifies, in order, the modules that must be loaded before this module. The fully expanded required-module order determined from the specified :requires module option must be consistent with all previously defined modules.

The :files module option specifies, in order, the files that are compiled (when appropriate) and loaded for this module. Similarly, the :patches module option specifies, in order, the patch files that are compiled (when appropriate) and loaded for this module. The combination of the :requires, :files, and :patches module options specifies a total ordering of all the files that must be compiled and loaded for this module (the files associated with each of the required modules followed by the files specified for this module followed by the patch files associated with each of the required modules followed by the patch files specified for this module).In order to maintain an invariant patch ordering, patches should be added sequentially to the main (last loaded) module of an application. The source files for patches should be placed in a subdirectory named patches in the module's source-file directory.

File-options have the following effects:

:developing	If the patch file has changed, recompile and reload it (patch files only)
:forces-recompile	If the file has changed, recompile and reload all subsequent files and modules
:noload	Compile, but do not load the file
:recompile	Always recompile the file
:reload	Always reload the file
:skip-recompile	Skip recompiling the file if it has been loaded already (takes precedence over other recompilation decisions)
:source	Do not compile the file (load the source instead)

The *documentation* string associated with a module can be accessed and set by using Common Lisp's documentation generic function with the module's *module-name* and the *doc-type* module.

See also

define-relative-directory (page 33)
define-root-directory (page 35)
describe-module (page 37)
describe-patches (page 39)
compile-module (page 26)
continue-patch (page 28)
get-patch-description (page 44)
finish-patch (page 40)
load-module (page 47)
load-module-file (page 49)
patch (page 53)
patch-loaded-p (page 55)
start-patch (page 57)
with-system-name (page 18)
with-module-redefinitions (page 59)

Examples

Define a root directory and module for :my-app:

```
(define-root-directory :my-app-root
  (make-pathname :directory "~/my-app"))
```

```
(define-module :my-app
  "My simple application module"
  (:requires :gbbopen-user)
  (:directory :my-app-root)
  (:files "preamble"
          ("macros" :forces-recompile)
          "classes"
          "my-app"
          "epilogue"))
```

Define a module for `:my-app` rooted relative to the file containing the **define-module** form:

```
(define-module :my-app
  "My simple application module"
  (:requires :gbbopen-user)
  (:files "preamble"
          ("macros" :forces-recompile)
          "classes"
          "my-app"
          "epilogue"))
```

Retrieve and then modify the documentation string of module `:my-app`:

```
> (documentation ':my-app 'module)
"My simple application module"
> (setf (documentation ':my-app 'module) "My way cool application module")
"My way cool application module"
> (documentation ':my-app 'module)
"My way cool application module"
>
```

---

define-module

define-relative-directory *name directory* `&rest` *subdirectories* [*Function*]

Purpose

Defines a directory relative to a root or relative directory or to the directory specification of a module definition.

Package `:module-manager`

Module `:module-manager`

Arguments

name	A keyword symbol naming the relative directory
directory	The keyword symbol name of a root or relative directory or the name of a module
subdirectories	One or more strings specifying, in order, subdirectories from *directory* to the relative directory. (The keyword `:up` or `:back` can also be supplied in addition to any of these strings, indicating to go upward one semantic or syntactic level of directory structure, respectively.)

Description

Root and relative directory definitions are used to isolate file-system details from module definitions. A relative directory is defined in relation to another directory definition, that is either a root directory or another relative directory (which itself is eventually associated with a root directory). If this root directory location changes, every relative directory associated with it is adjusted automatically.

The *documentation* string associated with a relative directory can be accessed and set by using Common Lisp's `documentation` generic function with the directory's *name* and the *doc-type* `directory`.

See also

define-root-directory (page 35)
get-directory (page 42)
show-defined-directories (page 56)
with-system-name (page 18)

Examples

Define a relative directory below `:my-app-root` named `:my-tests`:

```
(define-relative-directory :my-tests
  "The tests directory for My App"
  :my-app-root "tests")
```

Define another relative directory named `:my-performance-tests` below `:my-tests`:

```
(define-relative-directory :my-performance-tests
"The performance-tests subdirectory for My App"
:my-tests "performance")
```

Define a data directory sibling to the `source` directory of `:my-app` module:

```
(define-relative-directory :my-app-data
  "The data directory for My App"
  :my-app :up "data")
```

Retrieve and then modify the documentation string of the relative directory named `:my-tests`:

```
> (documentation ':my-app-data 'directory)
"The data directory for My App"
> (setf (documentation ':my-app-data 'directory)
"The data directory for my way cool application")
"The data directory for my way cool application"
> (documentation ':my-app-data 'directory)
"The data directory for my way cool application"
>
```

---

define-relative-directory

define-root-directory *name directory-specification* `&rest` *subdirectories* [*Function*]

Purpose

Define a root directory.

Package `:module-manager`

Module `:module-manager`

Arguments

name	A keyword symbol naming the root directory or a list containing the keyword-symbol name and, optionally, an application-version-identifier string
directory-specification	One of the following: • A pathname specifying the root directory • A keyword naming a previously defined root directory • A string specifying the root directory (only if a *documentation* string is supplied; otherwise, convert the root directory string to a pathname) • A symbol whose value is one of the above
subdirectories	One or more strings specifying, in order, subdirectories from *directory-specification* to the root directory. (The keyword `:up` or `:back` can also be supplied in addition to any of these strings, indicating to go upward one semantic or syntactic level of directory structure, respectively.)

Description

Root and relative directory definitions are used to isolate file-system details from module definitions. Root directories specify a fixed anchor directory for a tree of relative directory definitions. If the root directory is redefined to a new location, all relative directories beneath it are updated automatically.

Note: When a root directory is used as the *directory-specification* for a new root directory, the new root-directory location will not be changed if the location of the source root directory is changed.

When an application-version-identifier string is supplied in *name*, it is concatenated to the standard *<platform-dir>* compiled-directory name for all compiled module files that are defined relative to this root-directory.

The *documentation* string associated with a root directory can be accessed and set by using Common Lisp's `documentation` generic function with the directory's *name* and the *doc-type* `directory`.

See also

define-relative-directory (page 33)
get-directory (page 42)
get-root-directory (page 46)
show-defined-directories (page 56)
with-system-name (page 18)

Examples

Define a root directory named `:my-app-root`:

```
(define-root-directory :my-app-root
  "The installation directory for My App"
  (make-pathname :directory "~/my-app"))
```

Define a root directory named `:my-app-root` to be the directory containing the file containing the **define-root-directory** form:

```
(define-root-directory :my-app-root
  "The installation directory for My App"
  *load-truename*)
```

Define a root directory named `:my-app-root` to be the directory containing the file containing the **define-root-directory** form, but if the `:my-app-preprelease` feature is present, place the compiled files in a separate "beta" compile tree:

```
(define-root-directory '(:my-app-root #+my-app-prerelease "beta")
  "The installation directory for My App"
  *load-truename*)
```

Retrieve and then modify the documentation string of the root directory named `:my-app-root`:

```
> (documentation ':my-app-root 'directory)
"The installation directory for My App"
> (setf (documentation ':my-app-root 'directory)
   "The installation directory for my way cool application")
"The installation directory for my way cool application"
> (documentation ':my-app-root 'directory)
"The installation directory for my way cool application"
>
```

---

define-root-directory

describe-module *module-name* [*Function*]

Purpose
Print information about a module.

Package :module-manager

Module :module-manager

Arguments
module-name A keyword symbol naming a module

Errors
Module *module-name* has not been defined.
A relative directory specification contains a circularity.

Description
The description is printed to the ***standard-output*** stream.

Files and patch files shown with an asterisk following their load time indicate that the source file has been modified since that file was loaded.

See also
define-module (page 30)

Examples
Describe the :gbbopen-test module before it has been loaded:

```
> (describe-module :gbbopen-test)
Module :gbbopen-test (not loaded)
  The GBBopen Test module performs basic regression (trip) tests
    on the GBBopen Core entities.
  Requires: (:gbbopen-user)
  Fully expanded requires: (:module-manager :module-manager-user
                            :portable-threads :gbbopen-tools :gbbopen-core
                            :os-interface :gbbopen-user)
  Source directory: <install-dir>/source/gbbopen/test/
  Compiled directory: <install-dir>/<platform-dir>/gbbopen/test/
  Forces recompile date: None
  Files:                 basic-tests (:reload)
  Patches:               basic-tests-p001
                         basic-tests-p002 (:developing)
>
```

Describe the :gbbopen-test module after it has been loaded:

```
> (describe-module :gbbopen-test)
Module :gbbopen-test (loaded)
  The GBBopen Test module performs basic regression (trip) tests
    on the GBBopen Core entities.
```

```
    Requires: (:gbbopen-user)
    Fully expanded requires: (:module-manager :module-manager-user
                              :portable-threads :gbbopen-tools :gbbopen-core
                              :os-interface :gbbopen-user)
    Source directory: <install-dir>/source/gbbopen/test/
    Compiled directory: <install-dir>/<platform-dir>/gbbopen/test/
    Forces recompile date: None
    Files:   Jun 27 05:22  basic-tests (:reload)
    Patches: Jun 27 05:22  basic-tests-p001
             Jun 27 05:22* basic-tests-p002 (:developing)
  >
```

---

describe-module

describe-patches *module-name* [*Function*]

Purpose
Print information about patch *id* to a *module*.

Package :module-manager

Module :module-manager

Arguments
module-name A keyword symbol naming a module

Errors
Module *module-name* has not been defined.

See also

continue-patch (page 28)
define-module (page 30)
describe-module (page 37)
finish-patch (page 40)
get-patch-description (page 44)
parse-date (page 181)
patch (page 53)
patch-loaded-p (page 55)
start-patch (page 57)

Examples
Display the patches to module :my-app:

```
> (describe-patches ':my-app)
;; 1      Jun 22, 2008 Corkill (loaded Jun 24 13:08)
;;           A simple example patch
;; 2      Jun 23, 2008 Corkill (loaded Jun 24 13:08)
;;           A more complex patch example
>
```

Display the patches to module :gbbopen-test (when that module has not been loaded):

```
> (describe-patches ':gbbopen-test)
;; Module :gbbopen-test is not loaded
>
```

finish-patch *form** [*Macro*]

Purpose
Finish the definition of a multiple top-level form patch to a module.

Package `:module-manager`

Module `:module-manager`

Arguments
form A form

Errors
A patch has not been started with **start-patch**.

Description
The **start-patch** macro, along with **continue-patch** and **finish-patch**, can be used to define a patch as multiple top-level forms in the same patch file if a patch cannot be defined as a single **patch** form.

See also

allow-redefinition (page 65)
continue-patch (page 28)
define-module (page 30)
describe-module (page 37)
describe-patches (page 39)
get-patch-description (page 44)
parse-date (page 181)
patch (page 53)
patch-loaded-p (page 55)
start-patch (page 57)
undefmethod (page 128)

Example
Define a more complex patch (in a file named `my-app-p002.lisp` in the `patches` subdirectory of the module):

```
(start-patch (2 "06-23-08"
                :author "Corkill"
                :description "A more complex patch example")
    (printv "More complex example patch started!"))

(eval-when (:compile-toplevel)
  (continue-patch
   (printv "Defining compile-time-only-macro-for-patch...")
   (defmacro compile-time-only-macro-for-patch (x)
     `',x)))

(eval-when (:compile-toplevel :load-toplevel :execute)
```

```
    (continue-patch
     (printv "Defining macro-for-patch at compile & load time...")
     (defmacro macro-for-patch (x)
       `',x)))

  (continue-patch
   (printv "Using macro-for-patch at load time...")
   (macro-for-patch abc))

  (eval-when (:compile-toplevel :load-toplevel :execute)
    (continue-patch
     (printv "Using macro-for-patch at compile & load time...")
     (macro-for-patch xyz)))

  (eval-when (:compile-toplevel)
    (continue-patch
     (printv "Using compile-time-only-macro-for-patch...")
     (compile-time-only-macro-for-patch abc)))

  (finish-patch
   (printv "More complex example patch finished!"))
```

---

finish-patch

get-directory *name* `&rest` *subdirectories* ⇒ *pathname* [*Function*]

Purpose
Return the pathname based on a root directory, a relative directory, or a module.

Package `:module-manager`

Module `:module-manager`

Arguments
name	A keyword symbol naming a root directory, a relative directory, or a module.
subdirectories	One or more strings specifying, in order, subdirectories to be appended to the root directory, relative directory, or module directory. (The keyword `:up` or `:back` can also be supplied in addition to any of these strings, indicating to go upward one semantic or syntactic level of directory structure, respectively.)
pathname	A pathname

Returns
The pathname.

Errors
A relative directory specification contains a circularity.

Description
Root and relative-directory definitions are searched first. If no *name* directory definition is found, module definitions are searched. Root directories specify a fixed anchor directory for a tree of relative directory definitions. The pathname returned for relative directories is based on the source pathname.

See also
define-relative-directory (page 33)
define-root-directory (page 35)
get-root-directory (page 46)
show-defined-directories (page 56)

Examples
Return the pathname of the `:my-app-root` root directory:

```
> (get-directory :my-app-root)
#P"~/my-app/"
>
```

Return the pathname of the `:gbbopen-tools` relative directory:

```
> (get-directory :gbbopen-tools)
#P"<install-dir>/source/tools/"
>
```

Return this same pathname, this time computed from the `:gbbopen-root` root directory:

```
> (get-directory :gbbopen-root "tools")
#P"<install-dir>/source/tools/"
>
```

Return the pathname of the `:gbbopen-user` module:

```
> (get-directory :gbbopen-user)
#P"<install-dir>/source/gbbopen/"
>
```

---

get-directory

get-patch-description *id module-name* ⇒ *id, date, author, description, date-loaded, file-name* or `nil` [*Function*]

Purpose
Return the information describing patch *id* to a *module*.

Package `:module-manager`

Module `:module-manager`

Arguments
id	An object
module-name	A keyword symbol naming a module
date	A Universal Time
author	A string
description	A string or a proper list of strings or `nil`
date-loaded	A Universal Time
file-name	A string

Returns
Six values: *id*, *date*, *author*, *description*, *date-loaded*, and *file-name*, or `nil` if the patch has not been loaded.

Errors
Module *module-name* has not been defined.

See also

continue-patch (page 28)
define-module (page 30)
describe-module (page 37)
describe-patches (page 39)
finish-patch (page 40)
parse-date (page 181)
patch (page 53)
patch-loaded-p (page 55)
start-patch (page 57)

Examples
Get the description values for patch `1` to module `:my-app`:

```
> (get-patch-description 1 ':my-app)
1
3423139200
"Corkill"
"A simple example patch"
3423316130
"my-app-p001"
>
```

and for non-existent patch `3`:

```
> (get-patch-description 3 ':my-app)
nil
>
```

and for patch `1` in module `:gbbopen-test` (when that module has not been loaded):

```
> (get-patch-description 1 ':gbbopen-test)
nil
>
```

---

get-patch-description

get-root-directory *name* ⇒ *pathname* [*Function*]

Purpose
Return the pathname of the root directory of a root directory, a relative directory, or a module.

Package `:module-manager`

Module `:module-manager`

Arguments
name A keyword symbol naming a root directory, a relative directory, or a module.
pathname A pathname

Returns
The root-directory pathname of the defined directory or module.

Description
Root and relative-directory definitions are searched first. If no *name* directory definition is found, module definitions are searched. Root directories specify a fixed anchor directory for a tree of relative directory definitions.

See also
define-relative-directory (page 33)
define-root-directory (page 35)
get-directory (page 42)
show-defined-directories (page 56)

Examples
Return the root-directory pathname of the `:gbbopen-root` root directory:

```
> (get-root-directory :gbbopen-root)
#P"<install-dir>/"
>
```

Return the root-directory pathname of the `:gbbopen-tools` relative directory:

```
> (get-root-directory :gbbopen-tools)
#P"<install-dir>/"
>
```

Return the root-directory pathname of the `:gbbopen-user` module:

```
> (get-root-directory :gbbopen-user)
#P"<install-dir>/"
>
```

load-module *module-name* `&rest` *options* [*Function*]

Purpose
Load the files in a module.

Package `:module-manager`

Module `:module-manager`

Arguments
module-name A keyword symbol naming a module

option Any of the following keywords:

`:noautorun`	Binds ***autorun-modules**** to `nil` during loading
`:nopatches`	Do not load any patches
`:nopropagate`	Cancels (overrides) a specified :propagate option
`:patches-only`	Do not load/reload any non-patch files (binds ***patches-only**** to `t` during loading)
`:print`	Incrementally prints information during loading
`:propagate`	Applies the specified options to all required modules
`:reload`	Loads files even if they are already loaded
`:source`	Loads from the source file even if the existing compiled file is newer than the source file (implies `:reload`)

Errors
Module *module-name* has not been defined.
A relative directory specification contains a circularity.
Module *module-name* or one of its required modules has not been loaded and ***patches-only**** is true.

Description
These file options, when specified for individual files in the module definition (see **define-module**), have the following effects (overriding the behavior of *options* supplied to **load-module**):

`:developing`	If the compiled patch file has changed, reload it (patch files only)
`:forces-recompile`	If the file has changed, recompile and reload all subsequent files and modules
`:noautorun`	Binds ***autorun-modules**** to `nil` during compilation and loading
`:print`	Incrementally prints information during compilation and loading
`:reload`	Always reload the file
`:source`	Do not compile the file (load the source instead)

See also

autorun-modules* (page 24)
patches-only* (page 25)
compile-module (page 26)
define-module (page 30)
load-module-file (page 49)

Example

Load the GBBopen User module and all its required modules:

```
(load-module :gbbopen-user)
```

REPL Note

Load-module can be invoked using the REPL command:

```
:lm [module-name [option*]]
```

which remembers the last specified *module-name* and *options* as default values for the command.

---

load-module

load-module-file *module-name file-name* `&rest` *options* ⇒ *pathname* [*Function*]

Purpose
Load a single file from a module.

Package `:module-manager`

Module `:module-manager`

Arguments
module-name A keyword symbol naming a module

file-name A string naming a file in the module

option Any of the following keywords:

`:noautorun`	Binds ***autorun-modules*** to `nil` during loading
`:print`	Incrementally prints information during loading
`:source`	Loads from the source file even if the existing compiled file is newer than the source file

pathname A pathname

Returns
The pathname of the loaded file.

Errors
Module *module-name* has not been defined.
File *file-name* is not associated with the module.

A relative directory specification contains a circularity.

Description
File options specified for individual files in the module definition (see **define-module**) are ignored by **load-module-file**.

See also

autorun-modules (page 24)

define-module (page 30)

load-module (page 47)

Example
Load the source of the file `tools` from the GBBopen Tools module:

```
> (load-module-file :gbbopen-tools "tools" :source)
;; Loading <install-dir>/source/tools/tools.lisp
#P"<install-dir>/source/tools/tools.lisp"
>
```

REPL Note

Load-module-file can be invoked using the REPL command:

```
:lmf [module-name file-name [option*]]
```

which remembers the last specified *module-name*, *file-name*, and *options* as default values for the command.

---

load-module-file

module-directories *module-name* ⇒ *source-directory, compiled-directory* [*Function*]

Purpose
Return the source and compiled directories of a module.

Package :module-manager

Module :module-manager

Arguments
module-name A keyword symbol naming a module
source-directory A pathname
compiled-directory A pathname

Returns
Two values: the source directory pathname and the compiled directory pathname.

Errors
Module *module-name* has not been defined.
A relative directory specification contains a circularity.

See also
define-module (page 30)
load-module-file (page 49)

Example
Return the source and compiled directories of the Agenda Shell module:

```
> (module-directories :agenda-shell)
#P"<install-dir>/source/gbbopen/control-shells"
#P"<install-dir>/<platform-dir>/gbbopen/control-shells"
>
```

module-loaded-p *module-name* ⇒ *boolean* [*Function*]

Purpose
Determine if a module has been fully loaded into Common Lisp.

Package `:module-manager`

Module `:module-manager`

Arguments
module-name A keyword symbol naming a module
boolean A generalized boolean

Returns
True if the module has been fully loaded; `nil` otherwise.

Errors
Module *module-name* has not been defined.

See also
define-module (page 30)
load-module (page 47)

Example
Check if the Agenda Shell module has been loaded:

```
> (module-loaded-p :agenda-shell)
t
>
```

patch (*id date* `&key` *author description*) *form** [*Macro*]

Purpose
Define a patch to a module.

Package :module-manager

Module :module-manager

Arguments
id	An object
date	A simple string representing a date (parsed by **parse-date**)
author	A string (default is `"Anonymous"`)
description	A string or a proper list of strings (default is `nil`)
form	A form

Description
The source file for a patch should be placed in a subdirectory named `patches` in the module's source-files directory.

If a patch cannot be defined as a single **patch** form, the more primitive **start-patch**, **continue-patch**, and **finish-patch** macros can be used to separate the patch into multiple top-level forms in the same patch file.

See also

allow-redefinition (page 65)
continue-patch (page 28)
define-module (page 30)
describe-module (page 37)
describe-patches (page 39)
finish-patch (page 40)
get-patch-description (page 44)
parse-date (page 181)
patch-loaded-p (page 55)
start-patch (page 57)
undefmethod (page 128)

Example
Define a simple patch (in a file named `my-app-p001.lisp` in the `patches` subdirectory of the module):

```
(patch (1 "06-22-08"
          :author "Corkill"
          :description "A simple example patch")
    (printv "Simple example patch loaded!"))
```

and add the patch file to the `:my-app` module:

```
(define-module :my-app
  (:requires :gbbopen-user)
  (:files "preamble"
          ("macros" :forces-recompile)
          "classes"
          "my-app"
          "epilogue")
  (:patches "my-app-p001"))
```

---

patch

patch-loaded-p *id module-name* ⇒ *boolean* [*Function*]

Purpose
Determine if patch *id* to *module* has been loaded.

Package :module-manager

Module :module-manager

Arguments
id	An object
module-name	A keyword symbol naming a module
boolean	A generalized boolean

Returns
True if the patch has been loaded; `nil` otherwise.

Errors
Module *module-name* has not been defined.

See also

continue-patch (page 28)
define-module (page 30)
describe-module (page 37)
describe-patches (page 39)
finish-patch (page 40)
get-patch-description (page 44)
parse-date (page 181)
patch (page 53)
start-patch (page 57)

Examples
Check if patch `1` to module `:my-app` has been loaded:

```
> (patch-loaded-p 1 ':my-app)
t
>
```

Check for non-existent patch `3`:

```
> (patch-loaded-p 3 ':my-app)
nil
>
```

Check for patch `1` in module `:gbbopen-test` (when that module has not been loaded):

```
> (patch-loaded-p 1 ':gbbopen-test)
nil
>
```

show-defined-directories <no arguments> [*Function*]

Purpose

Show all root and relative-directory definitions.

Package

`:module-manager`

Module

`:module-manager`

See also

define-relative-directory (page 33)
define-root-directory (page 35)
get-directory (page 42)
get-root-directory (page 46)

Example

List the currently defined root and relative directories:

```
> (show-defined-directories)
:gbbopen
    Relative to :gbbopen-root
    Subdirectories: ("gbbopen")
:gbbopen-root
    Root: <install-dir>
:gbbopen-tools
    Relative to :gbbopen-root
    Subdirectories: ("tools")
:module-manager-root
    Root: <install-dir>
>
```

start-patch (*id date* `&key` *author description*) *form** [*Macro*]

Purpose
Start the definition of a multiple top-level form patch to a module.

Package `:module-manager`

Module `:module-manager`

Arguments
id	An object
date	A simple string representing a date (parsed by **parse-date**)
author	A string (default is `"Anonymous"`)
description	A string or a proper list of strings (default is `nil`)
form	A form

Description
The source file for a patch should be placed in a subdirectory named `patches` in the module's source-files directory.

The **start-patch** macro, along with **continue-patch** and **finish-patch**, can be used to define a patch as multiple top-level forms in the same patch file if a patch cannot be defined as a single **patch** form.

See also

allow-redefinition (page 65)
continue-patch (page 28)
define-module (page 30)
describe-module (page 37)
describe-patches (page 39)
finish-patch (page 40)
get-patch-description (page 44)
parse-date (page 181)
patch (page 53)
patch-loaded-p (page 55)
undefmethod (page 128)

Example
Define a more complex patch (in a file named `my-app-p002.lisp` in the `patches` subdirectory of the module):

```
(start-patch (2 "06-23-08"
                :author "Corkill"
                :description "A more complex patch example")
    (printv "More complex example patch started!"))

(eval-when (:compile-toplevel)
  (continue-patch
   (printv "Defining compile-time-only-macro-for-patch..."))
```

```
    (defmacro compile-time-only-macro-for-patch (x)
      `',x)))

  (eval-when (:compile-toplevel :load-toplevel :execute)
    (continue-patch
     (printv "Defining macro-for-patch at compile & load time...")
     (defmacro macro-for-patch (x)
       `',x)))

  (continue-patch
   (printv "Using macro-for-patch at load time...")
   (macro-for-patch abc))

  (eval-when (:compile-toplevel :load-toplevel :execute)
    (continue-patch
     (printv "Using macro-for-patch at compile & load time...")
     (macro-for-patch xyz)))

  (eval-when (:compile-toplevel)
    (continue-patch
     (printv "Using compile-time-only-macro-for-patch...")
     (compile-time-only-macro-for-patch abc)))

  (finish-patch
   (printv "More complex example patch finished!"))
```

---

start-patch

with-module-redefinitions *form** [*Macro*]

Purpose
Defer required-module order checking of module definitions in *forms* until all *forms* have been processed.

Package :module-manager

Module :module-manager

Arguments
form A form

Errors
The fully expanded required-module order of a module redefined in *forms* conflicts with the fully expanded required-module order of another module.

Description
Normally, the fully expanded required-module order specified by the `:requires` option is checked immediately for conflicts with previously defined modules by **define-module**. Wrapping several **define-module** forms with **with-module-redefinitions** allows the definitions to be conflicting temporarily, until all the *forms* have been processed.

See also
define-module (page 30)

Example
Redefine several modules with deferred conflict checking:

```
(with-module-redefinitions
  (define-module :alpha ...)
  (define-module :beta ...))
          ...
  (define-module :omega ...)))
```

3 GBBopen Tools

The GBBopen Tools module, `:gbbopen-tools`, contains useful Common Lisp additions and utilities. Convenient (shorthand operators (see page 143) for declared `fixnum`, `short-float`, `single-float`, `double-float`, and `long-float` numeric operators are provided, as are efficient pseudo probability (see page 149) operators. Uniform access to commonly used CLOS and MOP symbols (see page 141) is provided by the module's `:gbbopen-tools` package. Date and time parsing and formatting entities (see page 157), `fixnum`-based offset-universal-time functions (see page 197), and left-leaning red-black (LLRB) trees (see page 214) are also included in GBBopen Tools.

Additional GBBopen Tools entities that are not loaded as part of the `:gbbopen-tools` module are documented in the Additional GBBopen Tools section (see page 223) of this Reference. These additional-tool entities are grouped into separate modules that can be loaded as appropriate.

Several entities in the `:gbbopen-tools` module that are extended by additional methods in the `:gbbopen-core` module are documented in the GBBopen Core section (see page 323) of this Reference. These entities are:

```
*print-object-for-sending*
*save/send-references-only*
initialize-saved/sent-instance
make-duplicate-instance
make-duplicate-instance-changing-class
omitted-slots-for-saving/sending
print-object-for-saving/sending
print-slot-for-saving/sending
with-reading-saved/sent-objects-block
with-saving/sending-block
```

GBBopen Tools entities

Descriptions of GBBopen Tools entities follow.

:disable-compiler-macros [*Feature*]

Purpose

Disable GBBopen compiler macros.

See also

:full-safety (page 63)
defcm (page 82)

Example

Recompile a GBBopen application (and GBBopen itself) with all GBBopen compiler macros disabled:

```
(pushnew :disable-compiler-macros *features*)
(compile-module :my-app :recompile :propagate)
```

:full-safety [*Feature*]

Purpose

Disable GBBopen compile-time optimizations.

Description

The **:full-safety** feature disables all compile-time optimizations, including the use of compiler macros. Compiler macros can be disabled without disabling other compile-time optimizations with the feature **:disable-compiler-macros**.

See also

defcm (page 82)
:disable-compiler-macros (page 62)

Example

Recompile a GBBopen application (and GBBopen itself) with all GBBopen optimizations disabled:

```
(pushnew :full-safety *features*)
(compile-module :my-app :recompile :propagate)
```

disable-with-error-handling [*Variable*]

Purpose
Controls whether **with-error-handling** is in effect.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Value type A generalized boolean

Initial value `nil`

Description
Setting ***disable-with-error-handling*** to true causes **with-error-handling** to evaluate its *forms* without any error handling in place. Any *handler-form* or *error-form* in the **with-error-handling** is ignored.

Setting ***disable-with-error-handling*** to true is very handy for debugging an unexpected error in a **with-error-handling** *form*.

See also
with-error-handling (page 132)

Example
Use ***disable-with-error-handling*** to debug a handled form:

```
> (with-error-handling (/ 10 0) (printv (error-message)) nil)
;;  (error-message) => "Attempt to divide 10 by zero."
nil
> (setf *disable-with-error-handling* 't)
t
> (with-error-handling (/ 10 0) (printv (error-message)) nil)
Error: Attempt to divide 10 by zero.
>>
```

allow-redefinition *form** ⇒ *result** [*Macro*]

Purpose
Suppress redefinition warnings associated with processing *forms* when possible.

Package
`:gbbopen-tools` (home package is `:module-manager`)

Module
`:module-manager`

Arguments
form A form
results The values returned by evaluating the last *form*

Returns
The values returned by evaluating the last *form*.

See also
continue-patch (page 28)
finish-patch (page 40)
patch (page 53)
start-patch (page 57)

Example
```
> (defun redefined-function () "Silly.")
silly
> (defclass also-silly () ())
also-silly
> (allow-redefinition
     (defun redefined-function "Even more silliness.")
     (define-unit-class also-silly () ()))
also-silly
>
```

Note
This macro is defined in the `:module-manager` module in order to make it available as early as possible.

At present, redefinition warnings are not suppressed on CMUCL, ECL, SBCL, and Scieneer CL.

assq *item alist* ⇒ *entry* [*Function*]

Purpose
Search for *entry* in *alist* using `eq` as the comparison function.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
item An object
alist An association list
entry A cons that is an element of *alist* or `nil`

Returns
The first cons in *alist* whose car is `eq` to *item*, or nil if no such cons is found.

Description
Assq is a convenient shorthand for:

```
(assoc item (the list alist) :test #'eq)
```

See also
memq (page 102)

Examples
```
> (assq 'b '((a . 1) (b . 2) (c . 3) (b . 4)))
(b . 2)
> (assq 'x '((a . 1) (b . 2) (c . 3) (b . 4)))
nil
>
```

bounded-value *min number max* ⇒ *bounded-number* [*Function*]

Purpose
Return a numeric value that is bounded between a minimum and maximum value.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
min	A number (the minimum bound)
number	A number
max	A number (the maximum bound)
bounded-number	A number

Returns
One of the following values:

- *number* if it between *min* and *max*, inclusive
- *min* if *number* is less than *min*
- *max* if *number* is greater than *max*

Examples
```
> (bounded-value 3 pi 4)
3.141592653589793d0
> (bounded-value 3.5 pi 4)
3.5
> (bounded-value 2 pi 3)
3
>
```

Note
Declared numeric (see page 143) and pseudo probability (see page 149) versions of **bounded-value** are also provided: **bounded-value&**, **bounded-value$&**, **bounded-value$**, **bounded-value$$**, **bounded-value$$$**, and **bounded-value%**.

case-using *test keyform* {*normal-clause*}* [*otherwise-clause*] ⇒ *result** or `nil` [*Macro*]

Purpose

Conditionally execute the forms in a clause that is selected by matching the result of evaluating *keyform* according to *test*.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

test A symbol designating a comparison predicate (not evaluated)

keyform A form; evaluated to produce a *test-key* (see below)

results The values returned by evaluating the last *form* in the selected clause or `nil`

Returns

The values returned by the last form in the matching *normal-clause*; otherwise the values returned by the last form in the *otherwise-clause*, if specified; otherwise `nil`.

Detailed syntax

normal-clause ::= `(`*keys form**`)`
otherwise-clause ::= `({otherwise | t}` *form**`)`

Terms

test-key An object produced by evaluating *keyform*

keys An object or a proper list of objects. To refer to the symbols `t` and `otherwise` by themselves as the sole key object for a *normal-clause*, `(t)` and `(otherwise)`, respectively, must be specified as the *keys* for the clause.

form A form

Description

The specified *test* symbol is not evaluated; however the comparison predicate that it designates must be available during expansion of the **case-using** form.

The *keyform* is first evaluated to produce the *test-key*.

Each of the *normal-clauses* is then considered in turn. If the *test-key* matches that clause according to *test*, then the forms in that clause are evaluated as an implicit `progn`, and the values it returns are returned as the value of the **case-using** form.

Case-using is a generalization of Common Lisp's `case` macro.

See also

ccase-using (page 71)

ecase-using (page 90)

Examples

```
> (case-using string= "a"
    ("a" 1)
    (("b" "c" "d") 2))
1
> (case-using string= "d"
    ("a" 1)
    (("b" "c" "d") 2))
2
> (case-using string= "C"
    ("a" 1)
    (("b" "c" "d") 2))
nil
> (case-using string= "C"
    ("a" 1)
    (("b" "c" "d") 2)
    (otherwise -1))
-1
> (case-using equalp "C"
    ("a" 1)
    (("b" "c" "d") 2)
    (otherwise -1))
2
>
```

---

case-using

case-using-failure [*Condition*]

Package :gbbopen-tools

Module :gbbopen-tools

Description

The condition **case-using-failure** is a subclass of `type-error`.

See also

case-using (page 68)
ccase-using (page 71)
ecase-using (page 90)

ccase-using *test keyplace* {*clause*}* ⇒ *result** [*Macro*]

Purpose

Conditionally execute the forms in a clause that is selected by matching the result of evaluating *keyplace* according to *test*, generating a correctable error if no clause is selected.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

test A symbol designating a comparison predicate (not evaluated)

keyplace A form; evaluated to produce a *test-key* and possibly also used later as a place if no *keys* match (see below)

results The values returned by evaluating the last *form* in the selected *clause*

Returns

The values returned by the last form in the selected *clause*.

Errors

No *clause* was selected.

Detailed syntax

clause ::= `(`*keys form**`)`

Terms

test-key An object produced by evaluating *keyplace*

keys An object or a proper list of objects.

form A form

Description

The specified *test* symbol is not evaluated; however the comparison predicate that it designates must be available during expansion of the **case-using** form.

The *keyplace* is first evaluated to produce the *test-key*.

Each of the *clauses* is then considered in turn. If the *test-key* matches that clause according to *test*, then the forms in that clause are evaluated as an implicit `progn`, and the values it returns are returned as the value of the **ccase-using** form.

If no *clause* is selected, a correctable error of type `case-using-failure` (a subclass of `type-error`) is signaled. The offending datum is the *test-key* and the expected type is type equivalent to `(member (union` *keys* `:test` *test*`)`. Common Lisp's `store-value` restart can be used to correct the error.

Ccase-using is a generalization of Common Lisp's `ccase` macro.

See also

case-using-failure (page 70)
case-using (page 68)
ecase-using (page 90)

Examples

```
> (defparameter *x* "a")
*x*
> (ccase-using string= *x*
    ("a" 1)
    (("b" "c" "d") 2))
1
> (setf *x* "d")
"d"
> (ccase-using string= *x*
    ("a" 1)
    (("b" "c" "d") 2))
2
> (setf *x* "C")
"C"
> (ccase-using string= *x*
    ("a" 1)
    (("b" "c" "d") 2))
Error: "C" fell through an ecase-using string= form;
       the valid keys are "a", "b", "c", and "d".

Restart actions (select using :c n):
  0: Supply a new value for *x*.
>> :c 0
Enter a form to evaluate as the new value for *x*: "a"
*x* is now "a"
1
>
```

---

ccase-using

compiler-macroexpand *form* `&optional` *env* ⇒ *expansion, expanded-p* [*Function*]

Purpose
Expand *form* until it is no longer a compiler-macro form.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
form A form
env An environment object (default is `nil`)
expansion A form
expanded-p A generalized boolean

Returns
Two values:

- if *form* is a compiler-macro form, then return the repeatedly expanded compiler-macro form and true
- otherwise, return the given *form* and `nil`

Description
Form is expanded repeatedly by calling **compiler-macroexpand-1** until it is no longer a compiler-macro form.

See also
compiler-macroexpand-1 (page 74)

Examples
```
> (compiler-macroexpand '(ensure-list x))
(let ((#:g7105 x)) (if (listp #:g7105) #:g7105 (list #:g7105)))
t
> (compiler-macroexpand '(cons x nil))
(cons x nil)
nil
>
```

compiler-macroexpand-1 *form* `&optional` *env* ⇒ *expansion, expanded-p* [*Function*]

Purpose
Expand *form* if it is a compiler-macro form.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
form A form
env An environment object (default is `nil`)
expansion A form
expanded-p A generalized boolean

Returns
Two values:

- if *form* is a compiler-macro form, then return the compiler-macro expansion of *form* and true
- otherwise, return the given *form* and `nil`

Description
If *form* is a compiler-macro form, then **compiler-macroexpand-1** expands the compiler-macro-form call once.

See also
compiler-macroexpand (page 73)

Examples
```
> (compiler-macroexpand-1 '(ensure-list x))
(let ((#:g7105 x)) (if (listp #:g7105) #:g7105 (list #:g7105)))
t
> (compiler-macroexpand-1 '(cons x nil))
(cons x nil)
nil
>
```

counted-delete *item sequence* `&key` *from-end test test-not start end count key* [*Function*]
⇒ *result-sequence, count*

Purpose
A version of `delete` that returns the number of items that were deleted as a second value.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
item An object
sequence A proper sequence
from-end A generalized boolean (default is `nil`)
test A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'eql`)
test-not A function designator specifying a function object of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)
start Starting index into *sequence* (default is `0`)
end Ending index into *sequence* (default is `nil`, meaning end of *sequence*)
count An integer or `nil` (default is `nil`)
key A function designator specifying a function object of one argument, or `nil` (default is `nil`)
result-sequence A sequence

Returns
Two values:

- the *sequence* from which the elements that satisfy the *test* have been removed
- the number of items that have been removed, or `nil`

Description
Returns a sequence from which elements that satisfy the *test* have been deleted. The supplied *sequence* may be modified in constructing the result; however, modification of the supplied *sequence* itself is not guaranteed.

Specifying a *from-end* value of true matters only when the *count* is provided, and in that case only the rightmost *count* elements satisfying the *test* are deleted.

See also
atomic-delete (page 230)
delq (page 80)
delq-one (page 81)

Examples
```
> (counted-delete 'a '(a b c a b c))
(b c b c)
2
> (counted-delete #\a "abcabc")
```

```
"bcbc"
2
> (counted-delete 'z '(a b c a b c))
(a b c a b c)
0
> (counted-delete #\a "abcabc" :from-end 't :count 1)
"abcbc"
1
>
```

Note

This is what `delete` should have been (and was on the Lisp Machines).

---

counted-delete

decf-after *place* `&optional` *decrement* ⇒ *original-value* [*Macro*]

Purpose
Decrement the value of *place*, returning the original value of *place*.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
place	A form which is suitable for use as a generalized reference
decrement	A number (default is `1`)
original-value	A number

Returns
The original value of *place*.

See also
incf-after (page 94)

Examples
```
> (defparameter *x* 4)
*x*
> (decf-after *x*)
4
> *x*
3
> (decf-after *x* 3)
3
> *x*
0
>
```

Note
Declared numeric (see page 143) and pseudo probability (see page 149) versions of **decf-after** are also provided: **decf-after&**, **decf-after$&**, **decf-after$**, **decf-after$$**, **decf-after$$$**, and **decf-after%**.

decf/delete-acons *item decrement place* `&key` *key test test-not* ⇒ *new-place-value* [*Macro*]

Purpose
Decrement by *decrement* the value associated with *item* in an association list stored in *place*, deleting the cons associated with *item* if the value becomes zero.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
item	An object
decrement	A number
place	A form which is suitable for use as a generalized reference
key	A function designator specifying a function object of one argument, or `nil` (default is `nil`)
test	A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'eql`)
test-not	A function designator specifying a function object of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)
new-place-value	An association list

Returns
An association list (the new value of *place*).

Errors
Item *item* is not present in the association list stored in *place*.

Description
This is the inverse of **pushnew/incf-acons**.

See also
pushnew/incf-acons (page 113)

Examples
```
> (setf alist '((x . 2)(y . 2))
((x . 2)(y . 2))
> (decf/delete-acons 'x 1 alist)
((x . 1)(y . 2))
> (decf/delete-acons 'x 1 alist)
((y . 2))
> (decf/delete-acons 'y 2 alist)
nil
>
```

Note

Declared numeric (see page 143) and pseudo probability (see page 149) versions of **decf/delete-acons** are also provided: **decf&/delete-acons**, **decf$&/delete-acons**, **decf$/delete-acons**, **decf$$/delete-acons**, **decf$$$/delete-acons**, and **decf%/delete-acons**,

---

decf/delete-acons

delq *item list* ⇒ *list* [*Function*]

Purpose
Destructively delete all *item* elements from *list* using `eq` as the comparison function.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
item An object
list A proper list

Returns
A list from which all *item* elements have been deleted.

Description
Delq is a convenient shorthand for:

```
(delete item (the list list) :test #'eq)
```

As is the case with `delete`, **delq** may modify the top-level structure of *list* in constructing the *result-list*.

See also
atomic-delete (page 230)
counted-delete (page 75)
delq-one (page 81)

Examples
```
> (delq 'b '(a b c b))
(a c)
> (delq 'x '(a b c b))
(a b c b)
>
```

delq-one *item list* ⇒ *list* [*Function*]

Purpose
Destructively delete the first occurrence of *item* from *list* using `eq` as the comparison function.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
item An object
list A proper list

Returns
A list from which the first *item* element has been deleted.

Description
Delq-one is a convenient and efficient shorthand for:

```
(delete item (the list list) :test #'eq :count 1)
```

As is the case with `delete`, **delq-one** may modify the top-level structure of *list* in constructing the *result-list*.

See also
atomic-delete (page 230)
counted-delete (page 75)
delq (page 80)

Examples
```
> (delq-one 'b '(a b c b))
(a c b)
> (delq-one 'x '(a b c b))
(a b c b)
>
```

defcm *name lambda-list* [[*declaration** | *documentation*]] *form** ⇒ *name* [*Macro*]

Purpose
Define a read-time conditional compiler macro.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
name	A non-`nil`, non-keyword symbol that names a function or macro
lambda-list	A lambda-list
declaration	A declare expression (not evaluated)
documentation	A documentation string (not evaluated)
form	A form

Returns
The supplied *name*.

Description
Defcm is a read-time conditional version of Common Lisp's `define-compiler-macro`. As with `define-compiler-macro`:

- The expander function is installed as a compiler macro function for *name*.
- The `&whole` argument is bound to the form argument that is passed to **defcm**. The remaining *lambda-list* parameters are specified as if this form contained the function name in the car and the actual arguments in the cdr. However, if the car of the actual form is the symbol `funcall`, then the destructuring of the arguments is actually performed using its cddr instead.
- Documentation is attached as a documentation string to name (as kind `compiler-macro`) and to the compiler macro function.
- A compiler macro can decline to provide an expansion merely by returning the original form (which can be obtained by using `&whole`).

A compiler macro is not defined if the feature **:disable-compiler-macros** or the feature **:full-safety** is present when the expander function is executed on a form.

See also
:disable-compiler-macros (page 62)
:full-safety (page 63)

Example
Here is the compiler macro defined for **ensure-list**:

```
> (defcm ensure-list (x)
    (with-once-only-bindings (x)
      `(if (listp ,x) ,x (list ,x))))
ensure-list
>
```

define-class *class-name* (*{superclass-name}**) (*{slot-specifier}**) *{class-option}** [*Macro*]
⇒ *new-class*

Purpose

Extended macro for defining or redefining a class.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

class-name	A non-`nil`, non-keyword symbol that names the class
superclass-name	A non-`nil`, non-keyword symbol that specifies a direct superclass of the class *class-name*
slot-specifiers	See below
class-options	See below
new-class	A new or modified class object

Returns

The newly defined or modified class object.

Detailed syntax

slot-specifier ::= *slot-name* |
(*slot-name* [[*slot-option*]])

slot-option ::= {`:accessor` *reader-function-name*}* |
{`:allocation` *allocation-type*} |
{`:documentation` *string*} |
{`:initarg` *initarg-name*}* |
{`:initform` *form*} |
{`:reader` *reader-function-name*}* |
{`:type` *type-specifier*} |
{`:writer` *writer-function-name*}*

class-option ::= (`:default-initargs` . *initarg-list*) |
(`:documentation` *string*) |
(`:export-accessors` *boolean*) |
(`:export-class-name` *boolean*) |
(`:export-slot-names` *slots-specifier*) |
(`:generate-accessors` *slots-specifier*) |
(`:generate-accessors-format` {`:prefix` | `:suffix`} |
(`:generate-accessors-prefix` {*string* | *symbol*}) |
(`:generate-accessors-suffix` {*string* | *symbol*}) |
(`:generate-initargs` *slots-specifier*) |
(`:metaclass` *class-name*)

slots-specifier ::= `nil` | `t` | *included-slot-name** |
{`t :exclude` *excluded-slot-name**}

Terms

class-name A non-`nil`, non-keyword symbol that names a class

documentation A documentation string
initarg-list An initialization argument list
slot-name A non-`nil`, non-keyword symbol

Description

Each *superclass-name* argument specifies a direct superclass of the new class. If the superclass list is empty, then the direct superclass defaults to the single class `standard-object`.

The `:metaclass` class option, if specified, must be a subclass of `standard-class`. The default metaclass value is `standard-class`.

See also

define-unit-class (page 330)
make-instance (page 364)
with-generate-accessors-format (page 136)

Examples

Define a class, `rectangle`, generating "`-of`" slot accessors:

```
> (define-class rectangle (point)
    (length width))
#<standard-class rectangle>
>
```

Define a class, `foo`, generating "*class-name* . *slot-name*" slot accessors:

```
> (define-class foo ()
    ((slot :initform ':uninitialized))
    (:generate-accessors-format :prefix))
#<standard-class foo>
>
```

---

define-class

do-until *form test-form* [*Macro*]

Purpose

Evaluates *form* and then *test-form* repeatedly, as long as *test-form* evaluates to `nil`.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

form A form
test-form A form

See also

do-while (page 86)
until (page 130)
while (page 131)

Examples

```
> (let ((i 0))
    (do-until (print i)
        (> (incf i) 3)))
1
2
3
nil
> (let ((i 10))
    (do-until (print i)
        (> (incf i) 3)))
10
nil
>
```

do-while *form test-form* [*Macro*]

Purpose

Evaluates *form* and then *test-form* repeatedly, until *test-form* evaluates to `nil`.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

form A form
test-form A form

See also

do-until (page 85)
until (page 130)
while (page 131)

Examples

```
> (let ((i 0))
    (do-while (print i)
        (<= (incf i) 3)))
1
2
3
nil
> (let ((i 10))
    (do-while (print i)
        (<= (incf i) 3)))
10
nil
>
```

dosequence (*var sequence-form* [*result-form*]) *declaration** {*tag* | *form*}* ⇒ *result**or `nil` [*Macro*]

Purpose
A generalized `dolist`-style iterator for any sequence.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
var	A variable symbol
sequence-form	A form that evaluates to a sequence
result-form	A form
declarations	A declare expression (not evaluated)
tag	A `go` tag (not evaluated)
form	A form
results	The values returned by evaluating the last *form*

Returns
If a `return` or `return-from` form is executed, then the values passed from that form are returned; otherwise, the values returned by evaluating the *result-form* are returned, or `nil` if there is no *result-form*.

Description
The body of **dosequence** is like a `tagbody`. **Dosequence** evaluates *sequence-form*, which should produce a sequence. It then executes the body once for each element in the sequence, with *var* bound to the element.

The scope of the binding of *var* does not include the *sequence-form*, but it does include the *result-form*.

See also
dosublists (page 88)

Examples
```
> (dosequence (elt #(1 2 3)) (print elt))
1
2
3
nil
> (dosequence (char "abc") (print char))
#\a
#\b
#\c
nil
>
```

dosublists (*var list-form* [*result-form*]) *declaration** {*tag* | *form*}* ⇒ *result**or `nil` [*Macro*]

Purpose
A `dolist`-style iterator for successive sublists of a list.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
var	A variable symbol
list-form	A form that evaluates to a proper list
result-form	A form
declarations	A declare expression (not evaluated)
tag	A `go` tag (not evaluated)
form	A form
results	The values returned by evaluating the last *form*

Returns
If a `return` or `return-from` form is executed, then the values passed from that form are returned; otherwise, the values returned by evaluating the *result-form* are returned, or `nil` if there is no *result-form*.

Description
The body of **dosublists** is like a `tagbody`. **Dosublists** evaluates *list-form*, which should produce a proper list. It then executes the body once for each successive sublist in the list, with *var* bound to the sublist.

The scope of the binding of *var* does not include the *list-form*, but it does include the *result-form*.

See also
dosequence (page 87)

Examples
```
> (dosublists (sublist '(1 2 3)) (print sublist))
(1 2 3)
(2 3)
(3)
nil
> (dosublists (sublist '(1 2 3) (print "Done")) (print sublist))
(1 2 3)
(2 3)
(3)
"Done"
>
```

dotted-length *list* ⇒ *n* [*Function*]

Purpose
Return the length of a proper list or dotted list.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
list A proper list or a dotted list
n An integer

Returns
The length of *list*.

Examples
```
> (dotted-length '(a b))
2
> (dotted-length '(a b . c))
2
>
```

Note
This function will not work on a circular list.

ecase-using *test keyform* {*clause*}* ⇒ *result** [*Macro*]

Purpose

Conditionally execute the forms in a clause that is selected by matching the result of evaluating *keyform* according to *test*, generating an error if no clause is selected.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

test	A symbol designating a comparison predicate (not evaluated)
keyform	A form; evaluated to produce a *test-key* (see below)
results	The values returned by evaluating the last *form* in the selected *clause*

Returns

The values returned by the last form in the selected *clause*.

Errors

No *clause* was selected.

Detailed syntax

clause ::= `(`*keys form**`)`

Terms

test-key	An object produced by evaluating *keyform*
keys	An object or a proper list of objects.
form	A form

Description

The specified *test* symbol is not evaluated; however the comparison predicate that it designates must be available during expansion of the **case-using** form.

The *keyform* is first evaluated to produce the *test-key*.

Each of the *clauses* is then considered in turn. If the *test-key* matches that clause according to *test*, then the forms in that clause are evaluated as an implicit `progn`, and the values it returns are returned as the value of the **ecase-using** form.

If no *clause* is selected, a non-correctable error of type `case-using-failure` (a subclass of `type-error`) is signaled. The offending datum is the *test-key* and the expected type is type equivalent to `(member (union` *keys* `:test` *test*`)`.

Ecase-using is a generalization of Common Lisp's `ecase` macro.

See also

case-using-failure (page 70)
case-using (page 68)
ccase-using (page 71)

Examples

```
> (ecase-using string= "a"
    ("a" 1)
    (("b" "c" "d") 2))
1
> (ecase-using string= "d"
    ("a" 1)
    (("b" "c" "d") 2))
2
> (ecase-using string= "C"
    ("a" 1)
    (("b" "c" "d") 2))
Error: "C" fell through an ecase-using string= form;
       the valid keys are "a", "b", "c", and "d".
>>
```

---

ecase-using

ensure-finalized-class *class* ⇒ *finalized-class* [*Function*]

Purpose
Finalizes *class* if it is not already finalized.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
class A class designator
finalized-class A class object

Returns
The finalized *class*.

Example
```
> (ensure-finalized-class (find-class 'hyp))
#<standard-unit-class hyp>
>
```

Note
This function is compiled in-line for top performance.

ensure-list *object* ⇒ *list* [*Function*]

Purpose
Construct a list containing an object, if the object is not already a list.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
object An object
list A list

Returns
The *object* if it is a list or, if *object* is an atom, a newly consed list containing *object* as its sole element.

Examples
```
> (ensure-list 'x)
(x)
> (ensure-list '(x))
(x)
> (ensure-list nil)
nil
>
```

Note
This function is compiled in-line for top performance.

incf-after *place* `&optional` *increment* ⇒ *original-value* [*Macro*]

Purpose
Increment the value of *place*, returning the original value of *place*.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
place	A form which is suitable for use as a generalized reference
increment	A number (default is `1`)
original-value	A number

Returns
The original value of *place*.

See also
decf-after (page 77)

Examples
```
> (defparameter *x* 0)
*x*
> (incf-after *x*)
0
> *x*
1
> (incf-after *x* 3)
1
> *x*
4
>
```

Note
Declared numeric (see page 143) and pseudo probability (see page 149) versions of **incf-after** are also provided: **incf-after&**, **incf-after$&**, **incf-after$**, **incf-after$$**, **incf-after$$$**, and **incf-after%**.

list-length-1-p *list* ⇒ *boolean* [*Function*]

Purpose
Fast length = 1 test of a list.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
list A proper list or a dotted list
boolean A generalized boolean

Returns
True if *list* has length 1; `nil` otherwise.

See also
list-length> (page 97)
list-length>1 (page 98)
list-length>2 (page 99)
list-length-2-p (page 96)

Examples
```
> (list-length-1-p '(a))
t
> (list-length-1-p '(a b))
nil
> (list-length-1-p nil)
nil
> (list-length-1-p '(a . b))
nil
>
```

list-length-2-p *list* ⇒ *boolean* [*Function*]

Purpose
Fast length = 2 test of a list.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
list A proper list or a dotted list
boolean A generalized boolean

Returns
True if *list* has length 2; `nil` otherwise.

See also
list-length-1-p (page 95)
list-length> (page 97)
list-length>1 (page 98)
list-length>2 (page 99)

Examples
```
> (list-length-2-p '(a b))
t
> (list-length-2-p '(a b c))
nil
> (list-length-2-p '(a))
nil
> (list-length-2-p '(a b . c))
nil
>
```

list-length> *n list* ⇒ *boolean* [*Function*]

Purpose
Fast length $> n$ test of a list.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
n A non-negative `fixnum`
list A proper list or a dotted list
boolean A generalized boolean

Returns
True if *list* has length $> n$; `nil` otherwise.

See also
list-length> (page 97)
list-length-1-p (page 95)
list-length-2-p (page 96)

Examples
```
> (list-length> 1 '(a b))
t
> (list-length> 2 '(a b))
nil
> (list-length> 1 '(a))
nil
> (list-length> 1 '(a . b))
nil
> (list-length> 1 '(a b . c))
t
>
```

list-length>1 *list* ⇒ *boolean* [*Function*]

Purpose
Fast length > 1 test of a list.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
list A proper list or a dotted list
boolean A generalized boolean

Returns
True if *list* has length > 1; `nil` otherwise.

See also
list-length> (page 97)
list-length-1-p (page 95)
list-length-2-p (page 96)

Examples
```
> (list-length>1 '(a b))
t
> (list-length>1 '(a))
nil
> (list-length>1 '(a . b))
nil
> (list-length>1 '(a b . c))
t
>
```

list-length>2 *list* ⇒ *boolean* [*Function*]

Purpose
Fast length > 2 test of a list.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
list A proper list or a dotted list
boolean A generalized boolean

Returns
True if *list* has length > 2; `nil` otherwise.

See also
list-length> (page 97)
list-length>1 (page 98)
list-length-1-p (page 95)
list-length-2-p (page 96)

Examples
```
> (list-length>2 '(a b c))
t
> (list-length>2 '(a b))
nil
> (list-length>2 '(a b . c))
nil
> (list-length>2 '(a b c . d))
t
>
```

make-hash-values-vector *hash-table* ⇒ *vector* [*Function*]

Purpose
Return a vector containing all values in *hash-table*.

Package
`:gbbopen-tools`

Module
`:gbbopen-tools`

Arguments
hash-table A hash table
vector A vector

Returns
The newly allocated *vector*.

Examples
```
> (defparameter *ht* (make-hash-table))
*ht*
> (make-hash-values-vector *ht*)
#()
> (setf (gethash 'a *ht*) 1)
1
> (setf (gethash 'b *ht*) 2)
2
> (setf (gethash 'c *ht*) 3)
3
> (make-hash-values-vector *ht*)
#(3 1 2)
>
```

make-keyword *x* ⇒ *keyword* [*Function*]

Purpose
Return a keyword symbol as specified by *x*.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
x A string, symbol, or character
keyword A keyword symbol

Returns
The keyword symbol.

Examples
```
> (make-keyword 'gbbopen)
:gbbopen
> (make-keyword "GBBOPEN")
:gbbopen
> (make-keyword #\X)
:x
>
```

Note
This function is compiled in-line for top performance.

memq *item list* ⇒ *tail* [*Function*]

Purpose
Search for *item* in *list* using `eq` as the comparison function.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
item An object
list A proper list
tail A proper list

Returns
The tail of *list* beginning with *item* if *item* is present; `nil` otherwise.

Description
Memq is a convenient shorthand for:

```
(member item (the list list) :test #'eq)
```

See also
counted-delete (page 75)
assq (page 66)
delq (page 80)
delq-one (page 81)

Examples
```
> (memq 'b '(a b c b))
(b c b)
> (memq 'x '(a b c b))
nil
>
```

multiple-value-setf (*place**) *form* ⇒ *primary-value* [*Macro*]

Purpose
Assign values to *places*.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
place A form which is suitable for use as a generalized reference or `nil`
form A form
primary-value The first value returned by evaluating *form*

Returns
The primary value returned by *form*.

Description
The *form* is evaluated, and each *place* is assigned to the corresponding value returned by *form*. If `nil` is used one or more *places*, no assignment is made to the `nil` "place," but the corresponding value is consumed (ignored). If there are more *places* than values returned, `nil` is assigned to the extra *places*. If there are more values than *places*, the extra values are discarded.

Examples
```
> (setf *x* (cons 0 0))
(0 . 0)
> (multiple-value-setf ((car *x*) (cdr *x*)) (values 1 2 3))
1
> *x*
(1 . 2)
> (multiple-value-setf (nil nil (cdr *x*)) (values -1 -2 -3))
-1
> *x*
(1 . -3)
> (multiple-value-setf ((car *x*) (cdr *x*)) 100)
100
> *x*
(100)
> (multiple-value-setf ((car *x*) (cdr *x*)) nil)
nil
> *x*
(nil)
>
```

nsorted-insert *item list* `&optional` *predicate key* ⇒ *result-list* [*Function*]

Purpose
Positionally insert *item* in *list* based on *predicate* and *key* functions.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
item	An object
list	A proper list
predicate	A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'<`)
key	A function designator specifying a function object of one argument, or `nil` (default is `nil`)
result-list	A proper list

Returns
A list into which *item* has been inserted.

Description
The supplied *list* may be modified in constructing the result; however, modification of the supplied *list* itself is not guaranteed.

Example
```
> (nsorted-insert 5 '(2 4 6 8))
(2 4 5 6 8)
>
```

object-address *object* `&optional` *hex-string-p* ⇒ *address* [*Function*]

Purpose
Return the internal address of *object*.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
object An object
hex-string-p A generalized boolean (default is `nil`)
address An integer or a string

Returns
The integer internal address of *object* or a string containing the hexadecimal representation of the address if *hex-string-p* is true.

Description
Object-address can be useful with **printv**.

See also
printv (page 108)

Examples
```
> (object-address *package*)
1907568474
> (object-address *package* 't)
"71B32F5A"
>
```

print-instance-slot-value *instance slot-name stream* `&key` *function no-space* [*Generic Function*]

Purpose

Print the value of the *slot-name* slot in *instance* or `[Unbound]`, if the slot is unbound.

Method signatures

print-instance-slot-value (*instance* `standard-gbbopen-instance`) slot-name stream

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

instance A standard-gbbopen-instance object
slot-name A non-`nil`, non-keyword symbol
stream A stream
function A function designator specifying a function object of one argument
no-space A generalized boolean (default is `nil`)

Description

Unless *no-space* is true, a space character is printed to *stream*. Then the slot value in *instance* specified by *slot-name* is printed to *stream*, unless the slot is unbound, in which case `[Unbound]` is printed to *stream*.

If the slot is bound and *function* is specifed, *function* is called with the slot value and the result is printed to *stream* rather than the slot value.

See also

print-instance-slots (page 107)
standard-gbbopen-instance (page 126)

Example

Extend the `print-object` printing for `hyp` instances to include `location` and `belief` slot values:

```
(defmethod print-instance-slots ((obj hyp) stream)
  (call-next-method)
  (print-instance-slot-value obj 'location stream)
  (print-instance-slot-value obj 'belief stream))
```

print-instance-slots *instance stream* [*Generic Function*]

Purpose

Extend standard-gbbopen-instance printing performed by `print-object` to include additional slot-value information.

Method signatures

print-instance-slots (*instance* `ksa`) stream
print-instance-slots (*instance* `link/nonlink-slot-event`) stream
print-instance-slots (*instance* `multiple-instances-event`) stream
print-instance-slots (*instance* `single-instance-event`) stream
print-instance-slots (*instance* `space-instance-event`) stream
print-instance-slots (*instance* `standard-event-instance`) stream
print-instance-slots (*instance* `standard-gbbopen-instance`) stream
print-instance-slots (*instance* `standard-unit-instance`) stream
print-instance-slots :`after` (*instance* `standard-unit-instance`) stream

Package

:`gbbopen-tools`

Module

:`gbbopen-tools`

Arguments

instance A standard-gbbopen-instance object
stream A stream

See also

print-instance-slot-value (page 106)
standard-gbbopen-instance (page 126)

Examples

Extend the `print-object` printing for `hyp` instances to include `location` and `belief` slot values:

```
(defmethod print-instance-slots ((obj hyp) stream)
  (call-next-method)
  (when (and (slot-boundp obj 'location)
             (slot-boundp obj 'belief))
    (format stream " ~s ~s"
            (slot-value obj 'location)
            (slot-value obj 'belief))))
```

or if displaying `[Unbound]` for unbound slots is desired:

```
(defmethod print-instance-slots ((obj hyp) stream)
  (call-next-method)
  (print-instance-slot-value obj 'location stream)
  (print-instance-slot-value obj 'belief stream))
```

printv *form** ⇒ *result** [*Macro*]

Purpose
Assist debugging by printing forms and the results of evaluating them to `*trace-output*`.

Package
`:gbbopen-tools` (home package is `:module-manager`)

Module
`:module-manager`

Arguments
forms An implicit `progn` of forms to be evaluated and printed
results The values returned by evaluating the last *form* that is not the keyword symbol `:hr`

Returns
The values returned by evaluating the last *form* that is not the keyword symbol `:hr`

Description
The following is performed for each form in *forms*:

- if the form is the keyword symbol `:hr`, a dashed separator line printed to `*trace-output*`
- if the form is a string (before evaluation), it is treated as a label and printed to `*trace-output*` without enclosing double-quote characters
- if the form is a self-evaluating object, it is printed to `*trace-output*`
- otherwise, the form is printed to `*trace-output*`, then the form is evaluated and the result values are printed to `*trace-output*`.

See also
printvot (page 201)

Examples
```
> (printv :hr "PRINTV example" (list 1 2) (list 3 4) '"A quoted string"
          "Return no values:" (values)
          "Return multiple values:" (values 5 6) :hr)
;; -------------------------------------------------------------
;; PRINTV example
;;   (list 1 2) => (1 2)
;;   (list 3 4) => (3 4)
;; Return no values:
;;   (values) => [returned 0 values]
;;   '"A quoted string" => "A quoted string"
;; Return multiple values:
;;   (values 5 6) => 5; 6
;; -------------------------------------------------------------
5
6
> (printv *package* (object-address *package* 't))
;;  *package* => #<The GBBOPEN-USER package>
;;  (object-address *package* 't) => "71b32f5a"
```

```
"71b32f5a"
>
```

Note

This macro is defined in the `:module-manager` module in order to make it available as early as possible.

---

printv

push-acons *item value place* ⇒ *new-place-value* [*Macro*]

Purpose
Add a new *item value* cons to an association list stored in *place*.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
item	An object
value	An object
place	A form which is suitable for use as a generalized reference
new-place-value	An association list

Returns
An association list (the new value of *place*).

See also
pushnew-acons (page 111)
pushnew/incf-acons (page 113)

Examples
```
> (setf alist nil)
nil
> (push-acons 'x 1 alist)
((x . 1))
> (push-acons 'y 2 alist)
((y . 2) (x . 1))
> alist
((y . 2) (x . 1))
>
```

pushnew-acons *item value place* `&key` *key test test-not* ⇒ *new-place-value* [*Macro*]

Purpose

Replace the *value* associated with *item* in an association list stored in *place* or add a new `(`*item* `.` *value*`)` cons to the association list if there is no existing association for *item*.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

item	An object
value	An object
place	A form which is suitable for use as a generalized reference
key	A function designator specifying a function object of one argument, or `nil` (default is `nil`)
test	A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'eql`)
test-not	A function designator specifying a function object of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)
new-place-value	An association list

Returns

An association list (the new value of *place*).

See also

push-acons (page 110)
pushnew/incf-acons (page 113)

Examples

```
> (setf alist nil)
nil
> (pushnew-acons 'x 1 alist)
((x . 1))
> (pushnew-acons 'y 2 alist)
((y . 2) (x . 1))
> (pushnew-acons 'x -1 alist)
((y . 2) (x . -1))
> alist
((y . 2) (x . -1))
>
```

pushnew-elements *list place* `&key` *key test test-not* ⇒ *new-place-value* [*Macro*]

Purpose

Pushes new elements in *list* onto the list stored in *place*.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

list	A proper list
place	A form which is suitable for use as a generalized reference
key	A function designator specifying a function object of one argument, or `nil` (default is `nil`)
test	A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'eql`)
test-not	A function designator specifying a function object of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)
new-place-value	An association list

Returns

The new value of *place*.

Description

Each element in *list* is checked to see if it is already present in the proper list stored in *place*. If the element is not already present, it is prepended to the list stored in *place*.

Examples

```
> (setf x '(1 3 5))
(1 3 5)
> (pushnew-elements '(1 2 3) x)
(2 1 3 5)
> (pushnew-elements '(3 4 5) x)
(4 2 1 3 5)
>
```

pushnew/incf-acons *item increment place* `&key` *key test test-not* ⇒ *new-place-value* [*Macro*]

Purpose

Increment by *increment* the value associated with *item* in an association list stored in *place* or add a new `(item . increment)` cons to the association list if there is no existing association for *item*.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

item	An object
increment	A number
place	A form which is suitable for use as a generalized reference
key	A function designator specifying a function object of one argument, or `nil` (default is `nil`)
test	A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'eql`)
test-not	A function designator specifying a function object of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)
new-place-value	An association list

Returns

An association list (the new value of *place*).

See also

push-acons (page 110)
pushnew-acons (page 111)
decf/delete-acons (page 78)

Examples

```
> (setf alist nil)
nil
> (pushnew/incf-acons 'x 1 alist)
((x . 1))
> (pushnew/incf-acons 'x 1 alist)
((x . 2))
> (pushnew/incf-acons 'y 2 alist)
((y . 2) (x . 2))
> (pushnew/incf-acons 'x -1 alist)
((y . 2) (x . 1))
> alist
((y . 2) (x . 1))
>
```

Note

Declared numeric (see page 143) and pseudo probability (see page 149) versions of **pushnew/incf-acons** are also provided: **pushnew/incf&-acons**, **pushnew/incf$&-acons**, **pushnew/incf$-acons**, **pushnew/incf$$-acons**, **pushnew/incf$$$-acons**, and **pushnew/incf%-acons**.

---

pushnew/incf-acons

remove-properties *plist indicators* ⇒ *new-plist* [*Function*]

Purpose
Non-destructively removes properties from a property list.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
plist A property list
indicators A list of objects
new-plist A property list

Returns
The new property list.

Description
All instances of each specified property in the property list are removed.

See also
remove-property (page 116)

Examples
```
> (remove-properties '(:x 1 :y 2 :z 3) '(:y))
(:x 1 :z 3)
> (remove-properties '(:x 1 :y 2 :x 11 :y 12 :z -1) '(:z :y))
(:x 1 :x 11)
> (remove-properties '(:x 1 :y 2 :z 3) '(:missing))
(:x 1 :y 2 :z 3)
>
```

remove-property *plist indicator* ⇒ *new-plist* [*Function*]

Purpose
Non-destructively remove a property from a property list.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
plist A property list
indicator An object
new-plist A property list

Returns
The new property list.

Description
If there is more than one instance of property in the property list, only the first one is removed.

See also
remove-properties (page 115)

Examples
```
> (remove-property '(:x 1 :y 2 :z 3) :y)
(:x 1 :z 3)
> (remove-property '(:x 1 :y 2 :x 11 :y 12) :y)
(:x 1 :x 11 :y 12)
> (remove-property '(:x 1 :y 2 :z 3) :missing)
(:x 1 :y 2 :z 3)
>
```

set-equal *list-1 list-2* `&key` *test test-not key* ⇒ *boolean* [*Function*]

Purpose
Determine if all elements in two lists are present in both lists.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
list-1 A proper list
list-2 A proper list
test A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'eql`)
test-not A function designator specifying a function object of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)
key A function designator specifying a function object of one argument, or `nil` (default is `nil`)
boolean A generalized boolean

Returns
True if all elements in *list-1* are also in *list-2* and vice versa; `nil` otherwise.

Description
Duplicate elements in either list are permitted, so the lengths of *list-1* and *list-2* can differ and still return true.

Examples
```
> (set-equal '(1 2 3) '(3 2 1))
t
> (set-equal '(1 2) '(3 2 1))
nil
> (set-equal '(1 2 3) '(3 1))
nil
> (set-equal '(1 2 3) '(3 3 3 2 1))
t
> (set-equal '(1 2 3) '(4 5 6 7) :test #'/=)
t
>
```

sets-overlap-p *list-1 list-2* `&key` *test test-not key* ⇒ *boolean* [*Function*]

Purpose
Determine if any element in *list-1* appears in *list-2*.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
list-1 A proper list
list-2 A proper list
test A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'eql`)
test-not A function designator specifying a function object of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)
key A function designator specifying a function object of one argument, or `nil` (default is `nil`)
boolean A generalized boolean

Returns
True if any element in *list-1* is also in *list-2*; `nil` otherwise.

Description
Duplicate elements in either list are permitted.

Examples
```
> (sets-overlap-p '(1 2 3) '(3 4 5))
t
> (sets-overlap-p '(1 2) '(3 4 5))
nil
> (sets-overlap-p '(1 3 7) '(3 4 5 6) :test #'/=)
t
>
```

shuffle-list *list* ⇒ *shuffled-list* [*Function*]

Purpose
Return a copy of a list, with the elements in random order.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
list A proper list
shuffled-list A proper list

Returns
The shuffled copy of *list*.

Examples
```
> (shuffle-list '(a b c d))
(b a c d)
> (shuffle-list '(a b c d))
(c a d b)
>
```

shrink-vector *vector length* ⇒ *vector* [*Function*]

Purpose
Shorten a one-dimensional simple array destructively, if possible.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
vector A “simple” vector that is a one-dimensional simple array
length A non-negative integer, not greater than the length of *vector*

Returns
The truncated *vector*, which may not be identical (`eq`) to the original *vector* argument on some Common Lisp implementations.

Description
This function provides access to the Common Lisp implementation’s low-level truncation operation.

Example
```
> (shrink-vector "abcdefghijklmnopqrstuvwxyz" 3)
"abc"
>
```

Note
This function is compiled in-line for top performance.

sole-element *list* ⇒ *element* or `nil` [*Function*]

Purpose
Return the first element of a list containing, at most, one element.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
list A proper list
element An object

Returns
The sole element of *list* or `nil`.

Errors
List contains more than one element.

Description
If *list* is a cons, **sole-element** returns the car of that cons. If *list* is `nil`, **sole-element** returns `nil`. If *list* is a cons and the cdr of that cons is not `nil`, a continuable error is signaled. If you continue from the error, the first element is returned.

This function is preferable to **car** when you expect a list of, at most, one element. For example, this function is often used on the results of calling **find-instances** or **filter-instances** when only a single unit instance is expected in the result list.

Examples
```
> (sole-element '(a))
a
> (sole-element nil)
nil
> (sole-element '(a b))
Error: The list (a b) contains more than 1 element.
       If continued - Ignore the remaining elements.
>>
```

Note
This function is compiled in-line for top performance.

splitting-butlast *list* `&optional` *n* ⇒ *result-list, tail* [*Function*]

Purpose

Return all but the last *n* elements of *list* and, as a second value, the tail containing those last *n* elements.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

list	A proper list or a dotted list
n	A non-negative integer (default is `1`)
result-list	A proper list
tail	A proper list or a dotted list

Returns

Two values:

- a copy of *list* up to, but not including, the last *n* conses
- the unused tail of *list*

Examples

```
> (splitting-butlast '(a b c d e))
(a b c d)
(e)
> (splitting-butlast '(a b c d e) 3)
(a b)
(c d e)
> (splitting-butlast '(a b . c))
(a)
(b . c)
>
```

sorted-maphash *function hash-table predicate* &key *key* [*Function*]

Purpose

Apply a function once to each of the entries in a hash table according to the entry-key order determined by the *predicate* function

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

function	A function designator specifying a function object of two arguments, the key and value of the hash-table entry
hash-table	A hash table
predicate	A function designator specifying a function object of two arguments that returns a generalized boolean
key	A function designator specifying a function object of one argument, or `nil` (default is `nil`)

Description

For each entry in the hash table, the *function* is called with two arguments–the key and the value of that entry.

The first argument to the *predicate* function is the key of one entry in the hash table (or part of that key extracted by the *key* function, if supplied); the second argument is the key of another entry in the hash table (or part of that key extracted by the *key* function, if supplied). The *predicate* function should return true if and only if the first argument is strictly less than the second; otherwise the predicate should return false.

Example

Print a list of entries in `hash-table`, in ascending order of their keys (which are strings):

```
(sorted-maphash
   #'(lambda (key value)
       (format t "~&;; key: ~s value: ~s~%" key value)
   hash-table
   #'string<)
```

sortf *place predicate* `&key` *key* [*Macro*]

Purpose

Sort the sequence value of *place*, and change the value of *place* to the sorted result.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments

place A form which is suitable for use as a generalized reference

predicate A function designator specifying a function object of two arguments that returns a generalized boolean

key A function designator specifying a function object of one argument, or `nil` (default is `nil`)

Description

The first argument to the *predicate* function is one element of the sequence value in *place* (or part of that element extracted by the key function, if supplied); the second argument another element of the sequence (or part of that element extracted by the *key* function, if supplied). The *predicate* function should return true if and only if the first argument is strictly less than the second; otherwise the predicate should return false.

The sorting operation can be destructive, and it is not guaranteed stable. Elements considered equal by *predicate* might not stay in their original order.

See also

stable-sortf (page 125)

Example

```
> (defparameter *x* '(1 5 3 2 4))
*x*
> (sortf *x* '<)
(1 2 3 4 5)
> *x*
(1 2 3 4 5)
>
```

stable-sortf *place predicate* `&key` *key* [*Macro*]

Purpose

Stably sort the sequence value of *place*, and change the value of *place* to the sorted result.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

place	A form which is suitable for use as a generalized reference
predicate	A function designator specifying a function object of two arguments that returns a generalized boolean
key	A function designator specifying a function object of one argument, or `nil` (default is `nil`)

Description

The first argument to the *predicate* function is one element of the sequence value in *place* (or part of that element extracted by the key function, if supplied); the second argument another element of the sequence (or part of that element extracted by the *key* function, if supplied). The *predicate* function should return true if and only if the first argument is strictly less than the second; otherwise the predicate should return false.

The sorting operation can be destructive. Elements considered equal by *predicate* stay in their original order.

See also

sortf (page 124)

Example

```
> (defparameter *x* '(1 2 3 6 5 4))
*x*
> (stable-sortf *x* #'(lambda (x y) (and (oddp x) (evenp y))))
(1 3 5 2 6 4)
> *x*
(1 3 5 2 6 4)
>
```

standard-gbbopen-instance [*Class*]

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Description

The class **standard-gbbopen-instance** is a subclass of `standard-object`. It is a superclass of **deleted-unit-instance**, **standard-event-instance**, **standard-link-pointer**, and **standard-unit-instance**.

See also

deleted-unit-instance (page 337)
print-instance-slots (page 107)
standard-event-instance (page 411)
standard-link-pointer (page 388)
standard-unit-instance (page 370)

trimmed-substring *character-bag string* `&optional` *start end* ⇒ *trimmed-substring* [*Function*]

Purpose
Extract and trim a substring from *string*.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
character-bag A sequence containing characters
string A string designator
start Starting index into *string* (default is `0`)
end Ending index into *string* (default is `nil`, meaning end of *string*)
trimmed-substring A string

Returns
The extracted, trimmed string

Examples
```
> (trimmed-substring '(#\space #\tab) "xxx   abc   yyy" 3 12)
"abc"
> (trimmed-substring " " "xxx   abc   yyy" 3 12)
"abc"
> (trimmed-substring " " "xxx   abc   yyy" 3)
"abc   yyy"
> (trimmed-substring " " "xxx   abc   yyy" 3 13)
"abc   y"
> (trimmed-substring " " "xxx   abc   yyy" 2 12)
"x   abc"
> (trimmed-substring " " "   abc   ")
"abc"
>
```

Note
The function **trimmed-substring** is semantically equivalent to

```
(string-trim character-bag (subseq string start end))
```

but avoids allocating an intermediate substring.

undefmethod *function-name* {*method-qualifier*}* *specialized-lambda-list* [*Macro*]
[[*declaration** | *documentation*]] *form**

Purpose
Locate and undefine a method.

Package
`:gbbopen-tools`

Module
`:gbbopen-tools`

Arguments
function-name	Either a symbol or `(setf symbol)`
method-qualifier	A non-list method qualifier object (such as `:before`, `:after`, or `:around`)
specialized-lambda-list	A specialized lambda list (as per `defmethod`)
declarations	A declare expression (not evaluated)
documentation	A documentation string (not evaluated)
forms	Zero or more forms

See also
continue-patch (page 28)
finish-patch (page 40)
patch (page 53)
start-patch (page 57)

Example
After creating an undesired method, use **undefmethod** to remove it:

```
> (defmethod instance-name-of :before ((instance standard-unit-instance))
    (print "Oops"))
#<standard-method instance-name-of :before (standard-unit-instance)>
> (instance-name-of (find-instance-by-name 112 'hyp))
"Oops"
112
> (undefmethod instance-name-of :before ((instance standard-unit-instance)))
#<standard-generic-function instance-name-of>
> (instance-name-of (find-instance-by-name 112 'hyp))
112
>
```

Note
This macro may not be able to locate and undefine some methods with environment-specific eql specializers.

unbound-value-indicator [*Constant*]

Purpose

Represent an unbound value.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Value type

A keyword symbol

Value

`:---unbound---`

See also

define-unit-class (page 330)

Example

Define a slot-reader function that returns the value of `my-slot` or **unbound-value-indicator** if the slot is unbound:

```
(defun safe-my-slot-of (instance)
  (if (slot-bound-p instance 'my-slot)
      (slot-value instance 'my-slot)
      unbound-value-indicator))
```

until *test-form declaration* form** [*Macro*]

Purpose

Evaluates *test-form* and each *form* repeatedly, as long as *test-form* evaluates to `nil`.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

test-form A form
declaration A declare expression (not evaluated)
forms An implicit **progn** of forms to be evaluated

See also

do-until (page 85)
do-while (page 86)
while (page 131)

Examples

```
> (let ((i 0))
    (until (> (incf i) 3)
       (print i)))
1
2
3
nil
> (until 't (print "No"))
nil
>
```

while *test-form declaration* form** [*Macro*]

Purpose

Evaluates *test-form* and each *form* repeatedly, until *test-form* evaluates to `nil`.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

test-form A form
declaration A declare expression (not evaluated)
forms An implicit **progn** of forms to be evaluated

See also

do-until (page 85)
do-while (page 86)
until (page 130)

Examples

```
> (let ((i 0))
    (while (<= (incf i) 3)
       (print i)))
1
2
3
nil
> (while nil (print "No"))
nil
>
```

with-error-handling [*form* | (*form* [(:conditions *type*)] *handler-form**) *error-form** [*Macro*]
⇒ *result**

Purpose

Evaluate each *handler-form* and each *error-form* if an error occurs while evaluating *form*.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments

form	A form
type	A type specifier (default is `(and error (not interrupt-signal))` on Allegro CL; otherwise `error`)
handler-forms	Zero or more forms
error-forms	Zero or more forms
results	The values returned by evaluating *form*, the values returned by evaluating the last *handler-form* form or the last *error-form* form, or no values

Returns

The values returned by evaluating *form* unless an error occurs during that evaluation in which case:

- the values of evaluating the last *error-form* form, if specified, are returned
- otherwise, the values of evaluating the last *handler-form* form, if specified, are returned
- otherwise, no values are returned

Description

If an error occurs while evaluating *form*, each *handler-form* is evaluated in the dynamic context of the error, then the dynamic context is unwound to that in which *form* was evaluated and each *error-form* is evaluated.

A lexical function, **error-message**, is available for use within each *handler-form* and *error-form*. This lexical function accepts no arguments and returns a string describing the error that occurred during the evaluation of *form*.

Another lexical function, **error-condition**, is also available for use within each *handler-form* and nobr*error-form*. This lexical function accepts no arguments and returns the condition object that signaled the error.

The conditions that are handled can be changed by using the `(:conditions` *type*`)` option. Unlike other Common Lisp implementations, Allegro CL includes interrupt signals (typically generated by the user typing `control-C` characters in the REPL) as `error` conditions. Interrupt signals are excluded by default on Allegro CL.

See also

disable-with-error-handling (page 64)

Examples

```
> (with-error-handling (values 1 2 3) ':error-occurred)
1
2
3
> (with-error-handling (values 1 2 (error "Bad")) ':error-occurred)
:error-occurred
> (with-error-handling (values 1 2 (/ 10 0)) (printv (error-message)) nil)
;;  (error-message) => "Attempt to divide 10 by zero."
nil
> (defparameter *x* 0)
*x*
> (with-error-handling
      ((let ((*x* 1))
         (error "A silly error has occurred."))
       (printv "Handler form" (error-message) *x*)
       (values :c :b :a))
    (printv "Error form" (error-message) *x*)
    (values :a :b :c))
;; Handler form
;;  (error-message) => "A silly error has occurred."
;;  *x* => 1
;; Error form
;;  (error-message) => "A silly error has occurred."
;;  *x* => 0
:a
:b
:c
> (with-error-handling
      ((let ((*x* 1))
         (error "A silly error has occurred."))
       (printv "Handler form" (error-message) *x*)
       (values :c :b :a)))          ; No error forms
;; Handler form
;;  (error-message) => "A silly error has occurred."
;;  *x* => 1
:c
:b
:a
> (with-error-handling
      ;; No handler-forms:
      ((let ((*x* 1))
         (error "A silly error has occurred.")))
    (printv "Error form" (error-message) *x*)
    (values :a :b :c))
;; Error form
;;  (error-message) => "A silly error has occurred."
;;  *x* => 0
:a
:b
:c
> (with-error-handling (warn "Not too bad") ':error-occurred)
```

```
;; Warning: Not too bad
nil
> (with-error-handling ((warn "Not too bad")
                        (:conditions (or (and error
                                              #+allegro
                                              (not interrupt-signal))
                                         warning)))
    :error-occurred)
:error-occurred
>
```

---

with-error-handling

with-full-optimization (*option**) *declaration** *form** ⇒ *result** [*Macro*]

Purpose

Compile *forms* with `(speed 3)`, `(safety 0)`, `(debug 0)`, and `(compilation-speed 0)` optimization settings.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

option No options are currently supported

declaration A declare expression (not evaluated)

forms An implicit **progn** of forms to be evaluated

results The values returned by evaluating the last *form*

Returns

The values returned by evaluating *form*.

Description

This macro provides a convenient means of declaring small code fragments for fastest (and least safe) compiler optimizations. If the feature **:full-safety** is present at compile time, this macro has no effect on optimization settings.

Examples

Declare a function definition, including argument checking, to be fully optimized for the fastest (and least safe) execution:

```
(with-full-optimization ()
  (defun extent-> (value)
    `(,value ,infinity)))
```

Optimize the same function definition, but this time without invocation and argument-checking optimizations:

```
(defun extent-> (value)
  (with-full-optimization ()
    `(,value ,infinity)))
```

with-generate-accessors-format (*format* [*prefix/suffix-name*]) *form** ⇒ *result** [*Macro*]

Purpose

Change the default for accessor names generated by **define-class**, **define-event-class**, **define-space-class**, and **define-unit-class** definitions appearing in *forms*.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

format Either the keyword `:prefix` or `:suffix`

prefix/suffix-name One of the following (evaluated):

- A string
- A symbol
- A function designator specifying a function object accepting two arguments, class-name and slot-name, that returns the complete string to be used for the accessor name

forms An implicit **progn** of forms

results The values returned by evaluating the last *form*

Returns

The values returned by evaluating the last *form*.

Description

If a function object *prefix/suffix-name* is specified, it is called to produce the complete accessor-name string, no matter which *format* value is provided. Otherwise, if `:prefix` is specified as the *format* value, a string or symbol *prefix/suffix-name* is concatenated in front of the slot name to generate the slot-accessor name. If `:suffix` is specified as the *format* value, a string or symbol *prefix/suffix-name* is concatenated after the slot name.

The default *prefix/suffix-name* for `:prefix` is a function that generates historical GBB-style "*class-name* . *slot-name*" slot accessors; the default for `:suffix` is `'#:-of`.

See also

define-class (page 83)
define-event-class (page 394)
define-space-class (page 438)
define-unit-class (page 330)

Examples

Define three classes, `point`, `circle`, and `rectangle`, generating GBB-style "*class-name* . *slot-name*" slot accessors:

```
> (with-generate-accessors-format (:prefix)
    (define-class point ()
      (x y))
    (define-class circle (point)
```

```
      (radius))
    (define-class rectangle (point)
      (length width)))
#<standard-class rectangle>
>
```

Re-define the classes, generating "*slot-name*"-only slot accessors:

```
> (with-generate-accessors-format (:suffix "")
    (define-class point ()
      (x y))
    (define-class circle (point)
      (radius))
    (define-class rectangle (point)
      (length width)))
#<standard-class rectangle>
>
```

Re-define the classes, generating "*slot-name*`-of-`*class-name*" slot accessors (note that the `strange-name-string` name-generation function must be available at compile time):

```
> (eval-when (:compile-toplevel :load-toplevel :execute)
    (defun strange-name-string (class-name slot-name)
      (concatenate 'simple-string
        (symbol-name class-name) "-"
        (symbol-name '#:of) "-"
        (symbol-name slot-name))))
strange-name-string
> (with-generate-accessors-format (:prefix (symbol-function
'strange-name-string))
    (define-class point ()
      (x y))
    (define-class circle (point)
      (radius))
    (define-class rectangle (point)
      (length width)))
#<standard-class rectangle>
>
```

with-generate-accessors-format

with-gensyms (*symbol**) *declaration** *form** $\Rightarrow$ *result** [*Macro*]

Purpose

Bind each *symbol* to a gensym symbol.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments

symbols	Zero or more symbols to be bound to gensyms
declaration	A declare expression (not evaluated)
forms	An implicit **progn** of forms to be evaluated
results	The values returned by evaluating the last *form*

Returns

The values returned by evaluating the last *form*.

Examples

```
> (pprint (macroexpand '(with-gensyms (a b) (form))))
(let ((a (gensym))
      (b (gensym)))
  (form))
>
```

with-once-only-bindings (*symbol**) *declaration** *form** $\Rightarrow$ *result** [*Macro*]

Purpose

Evaluate, in order, each *symbol* and then make every reference to *symbol* inside each *form* refer to that value.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

symbols	Zero or more symbols to be evaluated once only
declaration	A declare expression (not evaluated)
forms	An implicit **progn** of forms to be evaluated
results	The values returned by evaluating the last *form*

Returns

The values returned by evaluating the last *form*.

Description

This is GBBopen's version of the "once-only" macro-writing macro that ensures that the forms associated with macro arguments are only evaluated once and in the specified order.

Example

```
(define-compiler-macro ensure-list (x)
  (with-once-only-bindings (x)
    `(if (listp ,x) ,x (list ,x))))
```

xor `&rest` *args* ⇒ *boolean* [*Function*]

Purpose
Return the exclusive or (XOR) of zero or more arguments.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
arg A generalized boolean (an object)
boolean A generalized boolean

Returns
True if there are an even odd number of true arguments; `nil` otherwise.

Examples
```
> (xor)
nil
> (xor nil nil)
nil
> (xor 't 't)
nil
> (xor nil 't)
t
> (xor 't nil)
t
> (xor nil 't nil 't 't nil)
t
>
```

3.1 CLOS and MOP

The GBBopen Tools module imports commonly used CLOS and MOP symbols into the `:gbbopen-tools` package and then exports them, making it easy to use CLOS and MOP entities (without worrying about each Common Lisp implementation's package structure) by simply using the `:gbbopen-tools` package. The following CLOS and MOP symbols are exported by `:gbbopen-tools`:

```
accessor-method-slot-definition
add-dependent
add-direct-method
add-direct-subclass
class-default-initargs
class-direct-default-initargs
class-direct-slots
class-direct-subclasses
class-direct-superclasses
class-finalized-p
class-precedence-list
class-prototype
class-slots
compute-applicable-methods-using-classes
compute-class-precedence-list
compute-default-initargs
compute-discriminating-function
compute-effective-method
compute-effective-slot-definition
compute-slots
direct-slot-definition
direct-slot-definition-class
effective-slot-definition
effective-slot-definition-class
ensure-class
ensure-class-using-class
ensure-generic-function-using-class
eql-specializer
eql-specializer-object
extract-lambda-list
extract-specializer-names
finalize-inheritance
find-method-combination
forward-referenced-class
funcallable-standard-class
funcallable-standard-instance-access
funcallable-standard-object
generic-function-argument-precedence-order
generic-function-declarations
generic-function-lambda-list
generic-function-method-class
generic-function-method-combination
generic-function-methods
generic-function-name
intern-eql-specializer
```

```
make-method-lambda
map-dependents
metaobject
method-function
method-generic-function
method-lambda-list
method-specializers
reader-method-class
remove-dependent
remove-direct-method
remove-direct-subclass
set-funcallable-instance-function
slot-boundp-using-class
slot-definition
slot-definition-allocation
slot-definition-initargs
slot-definition-initform
slot-definition-initfunction
slot-definition-location
slot-definition-name
slot-definition-readers
slot-definition-type
slot-definition-writers
slot-makunbound-using-class
slot-value-using-class
specializer
specializer-direct-generic-functions
specializer-direct-methods
standard-accessor-method
standard-direct-slot-definition
standard-effective-slot-definition
standard-instance-access
standard-reader-method
standard-slot-definition
standard-writer-method
update-dependent
validate-superclass
writer-method-class
```

Load-time warning messages are displayed if the Common Lisp implementation does not have one of these symbols in its CLOS or MOP implementation.

3.2 Declared Numerics

The :gbbopen-tools module contains a set of declared-numeric and pseudo probability (see page 149) operators that provide convenient shorthands for declaring fixnum, short-float, single-float, double-float, long-float, and pseudo-probability numeric operations. If the feature **:full-safety** is present at compile time, these operators do not make their fixnum, single-float, double-float, and long-float declarations.

The names of the declared-numeric and pseudo-probability operators include these "type-indicator" characters:

Characters	Declared Type
&	fixnum
$&	short-float
$	single-float
$$	double-float
$$$	long-float
%	pseudo-probability

Thus, the declared-numeric + operators are:

- +& fixnum addition
- +$& short-float addition
- +$ single-float addition
- +$$ double-float addition
- +$$$ long-float addition
- +% pseudo-probability addition

Most Common Lisp implementations map double-float numbers to the 64-bit IEEE 754 double format and single-float numbers to the 32-bit IEEE 754 single format. The four types of floating-point declared numeric operators (short-float, single-float, double-float, and long-float) are always defined, even if the Common Lisp implementation provides fewer distinct internal float representations. The features :has-short-float, :has-single-float, :has-double-float, and :has-long-float are added when the implementation provides that distinct internal float representation. (The feature :has-single-float is always defined.)

The following two features related to fixnum capabilities are defined when appropriate: :fixnum-size-below-29 and :fixnum-size-supports-unsigned-byte-32.

Some examples of declared-numeric operators include:

```
(& x) ⟹ (the fixnum x)
(+& x y) ⟹ (the fixnum (+ (the fixnum x) (the fixnum y))
(>$ a b) ⟹ (> (the single-float a) (the single-float b))
(minusp$$ value) ⟹ (minusp (the double-float value))
(truncate& x y)) ⟹ (the (values fixnum fixnum) (truncate (& x) (& y)))
```

The complete set of operators for each declared numeric type are listed in the following sections. (Pseudo probability operators are listed in the Pseudo Probabilities section (see page 149) of the manual.)

Fixnum operators

These `fixnum` declared-numeric operators are defined in the `:gbbopen-tools` module:

Operator	Operation	Example
`&`	`the fixnum`	`(& x)`
`+&`	`+`	`(+& x y z)`
`-&`	`-`	`(-& x y z)`
`1+&`	`1+`	`(1+& x)`
`1-&`	`1-`	`(1-& x)`
`*&`	`*`	`(*& x y z)`
`/&`	`/`	`(/& x y z)`
`=&`	`=`	`(=& x y z)`
`/=&`	`/=`	`(/=& x y z)`
`<&`	`<`	`(<& x y z)`
`<=&`	`<=`	`(<=& x y z)`
`>&`	`>`	`(>& x y z)`
`>=&`	`>=`	`(>=& x y z)`
`abs&`	`abs`	`(abs& x)`
`bounded-value&`	`bounded-value`	`(bounded-value& x y z)`
`ceiling&`	`ceiling`	`(ceiling& x divisor)`
`decf&`	`decf`	`(decf& x delta)`
`decf-&after`	`decf-after`	`(decf&-after x delta)`
`decf/delete&-acons`	`decf/delete-acons`	`(decf/delete&-acons x delta alist)`
`evenp&`	`evenp`	`(evenp& x)`
`fceiling&`	`fceiling`	`(fceiling& x divisor)`
`floor&`	`floor`	`(floor& x divisor)`
`ffloor&`	`ffloor`	`(ffloor& x divisor)`
`fround&`	`fround`	`(fround& x divisor)`
`ftruncate&`	`ftruncate`	`(ftruncate& x divisor)`
`incf&`	`incf`	`(incf& x delta)`
`incf&-after`	`incf-after`	`(incf&-after x delta)`
`max&`	`max`	`(max& x y z)`
`min&`	`min`	`(min& x y z)`
`minusp&`	`minusp`	`(minusp& x)`
`mod&`	`mod`	`(mod& x divisor)`
`oddp&`	`oddp`	`(oddp& x)`
`plusp&`	`plusp`	`(plusp& x)`
`pushnew/incf&-acons`	`pushnew/incf-acons`	`(pushnew/incf&-acons 'x delta alist)`
`round&`	`round`	`(round& x divisor)`
`truncate&`	`truncate`	`(truncate& x divisor)`
`zerop&`	`zerop`	`(zerop& x)`

The one-argument function **coerce&** provides convenient `fixnum` coercion:

```
(setf x (coerce& x))
```

Although `(coerce x 'fixnum)` is not permitted in Common Lisp, `(coerce& x)` is equivalent to `(truncate x)` when the remainder is zero and the returned quotient is a `fixnum`. Otherwise, `(coerce& x)` signals an error.

Short-float operators

These `short-float` declared-numeric operators are defined in the `:gbbopen-tools` module:

Operator	Operation	Example
`$&`	`the short-float`	`($& x)`
`+$&`	`+`	`(+$& x y z)`
`-$&`	`-`	`(-$& x y z)`
`1+$&`	`1+`	`(1+$& x)`
`1-$&`	`1-`	`(1-$& x)`
`*$&`	`*`	`(*$& x y z)`
`/$&`	`/`	`(/$& x y z)`
`=$&`	`=`	`(=$& x y z)`
`/=$&`	`/=`	`(/=$& x y z)`
`<$&`	`<`	`(<$& x y z)`
`<=$&`	`<=`	`(<=$& x y z)`
`>$&`	`>`	`(>$& x y z)`
`>=$&`	`>=`	`(>=$& x y z)`
`abs$&`	`abs`	`(abs$& x)`
`bounded-value$&`	`bounded-value`	`(bounded-value$& x y z)`
`ceiling$&`	`ceiling`	`(ceiling$& x divisor)`
`decf$&`	`decf`	`(decf$& x delta)`
`decf-$&after`	`decf-after`	`(decf$&-after x delta)`
`decf/delete$&-acons`	`decf/delete-acons`	`(decf/delete$&-acons x delta alist)`
`evenp$&`	`evenp`	`(evenp$& x)`
`fceiling$&`	`fceiling`	`(fceiling$& x divisor)`
`floor$&`	`floor`	`(floor$& x divisor)`
`ffloor$&`	`ffloor`	`(ffloor$& x divisor)`
`fround$&`	`fround`	`(fround$& x divisor)`
`ftruncate$&`	`ftruncate`	`(ftruncate$& x divisor)`
`incf$&`	`incf`	`(incf$& x delta)`
`incf$&-after`	`incf-after`	`(incf$&-after x delta)`
`max$&`	`max`	`(max$& x y z)`
`min$&`	`min`	`(min$& x y z)`
`minusp$&`	`minusp`	`(minusp$& x)`
`mod$&`	`mod`	`(mod$& x divisor)`
`oddp$&`	`oddp`	`(oddp$& x)`
`plusp$&`	`plusp`	`(plusp$& x)`
`pushnew/incf$&-acons`	`pushnew/incf-acons`	`(pushnew/incf$&-acons 'x delta alist)`
`round$&`	`round`	`(round$& x divisor)`
`truncate$&`	`truncate`	`(truncate$& x divisor)`
`zerop$&`	`zerop`	`(zerop$& x)`

The one-argument function **coerce$&** provides convenient `short-float` coercion:

```
(setf x (coerce$& x))
```

Single-float operators

These `single-float` declared-numeric operators are defined in the `:gbbopen-tools` module:

Operator	Operation	Example
`$`	`the single-float`	`($ x)`
`+$`	`+`	`(+$ x y z)`
`-$`	`-`	`(-$ x y z)`
`1+$`	`1+`	`(1+$ x)`
`1-$`	`1-`	`(1-$ x)`
`*$`	`*`	`(*$ x y z)`
`/$`	`/`	`(/$ x y z)`
`=$`	`=`	`(=$ x y z)`
`/=$`	`/=`	`(/=$ x y z)`
`<$`	`<`	`(<$ x y z)`
`<=$`	`<=`	`(<=$ x y z)`
`>$`	`>`	`(>$ x y z)`
`>=$`	`>=`	`(>=$ x y z)`
`abs$`	`abs`	`(abs$ x)`
`bounded-value$`	`bounded-value`	`(bounded-value$ x y z)`
`ceiling$`	`ceiling`	`(ceiling$ x divisor)`
`decf$`	`decf`	`(decf$ x delta)`
`decf-$after`	`decf-after`	`(decf$-after x delta)`
`decf/delete$-acons`	`decf/delete-acons`	`(decf/delete$-acons x delta alist)`
`evenp$`	`evenp`	`(evenp$ x)`
`fceiling$`	`fceiling`	`(fceiling$ x divisor)`
`floor$`	`floor`	`(floor$ x divisor)`
`ffloor$`	`ffloor`	`(ffloor$ x divisor)`
`fround$`	`fround`	`(fround$ x divisor)`
`ftruncate$`	`ftruncate`	`(ftruncate$ x divisor)`
`incf$`	`incf`	`(incf$ x delta)`
`incf$-after`	`incf-after`	`(incf$-after x delta)`
`max$`	`max`	`(max$ x y z)`
`min$`	`min`	`(min$ x y z)`
`minusp$`	`minusp`	`(minusp$ x)`
`mod$`	`mod`	`(mod$ x divisor)`
`oddp$`	`oddp`	`(oddp$ x)`
`plusp$`	`plusp`	`(plusp$ x)`
`pushnew/incf$-acons`	`pushnew/incf-acons`	`(pushnew/incf$-acons 'x delta alist)`
`round$`	`round`	`(round$ x divisor)`
`truncate$`	`truncate`	`(truncate$ x divisor)`
`zerop$`	`zerop`	`(zerop$ x)`

The one-argument function **coerce$** provides convenient `single-float` coercion:

```
(setf x (coerce$ x))
```

Double-float operators

These `double-float` declared-numeric operators are defined in the `:gbbopen-tools` module:

Operator	Operation	Example
`$$`	`the double-float`	`($$ x)`
`+$$`	`+`	`(+$$ x y z)`
`-$$`	`-`	`(-$$ x y z)`
`1+$$`	`1+`	`(1+$$ x)`
`1-$$`	`1-`	`(1-$$ x)`
`*$$`	`*`	`(*$$ x y z)`
`/$$`	`/`	`(/$$ x y z)`
`=$$`	`=`	`(=$$ x y z)`
`/=$$`	`/=`	`(/=$$ x y z)`
`<$$`	`<`	`(<$$ x y z)`
`<=$$`	`<=`	`(<=$$ x y z)`
`>$$`	`>`	`(>$$ x y z)`
`>=$$`	`>=`	`(>=$$ x y z)`
`abs$$`	`abs`	`(abs$$ x)`
`bounded-value$$`	`bounded-value`	`(bounded-value$$ x y z)`
`ceiling$$`	`ceiling`	`(ceiling$$ x divisor)`
`decf$$`	`decf`	`(decf$$ x delta)`
`decf-$$after`	`decf-after`	`(decf$$-after x delta)`
`decf/delete$$-acons`	`decf/delete-acons`	`(decf/delete$$-acons x delta alist)`
`evenp$$`	`evenp`	`(evenp$$ x)`
`fceiling$$`	`fceiling`	`(fceiling$$ x divisor)`
`floor$$`	`floor`	`(floor$$ x divisor)`
`ffloor$$`	`ffloor`	`(ffloor$$ x divisor)`
`fround$$`	`fround`	`(fround$$ x divisor)`
`ftruncate$$`	`ftruncate`	`(ftruncate$$ x divisor)`
`incf$$`	`incf`	`(incf$$ x delta)`
`incf$$-after`	`incf-after`	`(incf$$-after x delta)`
`max$$`	`max`	`(max$$ x y z)`
`min$$`	`min`	`(min$$ x y z)`
`minusp$$`	`minusp`	`(minusp$$ x)`
`mod$$`	`mod`	`(mod$$ x divisor)`
`oddp$$`	`oddp`	`(oddp$$ x)`
`plusp$$`	`plusp`	`(plusp$$ x)`
`pushnew/incf$$-acons`	`pushnew/incf-acons`	`(pushnew/incf$$-acons 'x delta alist)`
`round$$`	`round`	`(round$$ x divisor)`
`truncate$$`	`truncate`	`(truncate$$ x divisor)`
`zerop$$`	`zerop`	`(zerop$$ x)`

The one-argument function **coerce$$** provides convenient `double-float` coercion:

```
(setf x (coerce$$ x))
```

Long-float operators

These `long-float` declared-numeric operators are defined in the `:gbbopen-tools` module:

Operator	Operation	Example
`$$$`	`the long-float`	`($$$ x)`
`+$$$`	`+`	`(+$$$ x y z)`
`-$$$`	`-`	`(-$$$ x y z)`
`1+$$$`	`1+`	`(1+$$$ x)`
`1-$$$`	`1-`	`(1-$$$ x)`
`*$$$`	`*`	`(*$$$ x y z)`
`/$$$`	`/`	`(/$$$ x y z)`
`=$$$`	`=`	`(=$$$ x y z)`
`/=$$$`	`/=`	`(/=$$$ x y z)`
`<$$$`	`<`	`(<$$$ x y z)`
`<=$$$`	`<=`	`(<=$$$ x y z)`
`>$$$`	`>`	`(>$$$ x y z)`
`>=$$$`	`>=`	`(>=$$$ x y z)`
`abs$$$`	`abs`	`(abs$$$ x)`
`bounded-value$$$`	`bounded-value`	`(bounded-value$$$ x y z)`
`ceiling$$$`	`ceiling`	`(ceiling$$$ x divisor)`
`decf$$$`	`decf`	`(decf$$$ x delta)`
`decf-$$$after`	`decf-after`	`(decf$$$-after x delta)`
`decf/delete$$$-acons`	`decf/delete-acons`	`(decf/delete$$$-acons x delta alist)`
`evenp$$$`	`evenp`	`(evenp$$$ x)`
`fceiling$$$`	`fceiling`	`(fceiling$$$ x divisor)`
`floor$$$`	`floor`	`(floor$$$ x divisor)`
`ffloor$$$`	`ffloor`	`(ffloor$$$ x divisor)`
`fround$$$`	`fround`	`(fround$$$ x divisor)`
`ftruncate$$$`	`ftruncate`	`(ftruncate$$$ x divisor)`
`incf$$$`	`incf`	`(incf$$$ x delta)`
`incf$$$-after`	`incf-after`	`(incf$$$-after x delta)`
`max$$$`	`max`	`(max$$$ x y z)`
`min$$$`	`min`	`(min$$$ x y z)`
`minusp$$$`	`minusp`	`(minusp$$$ x)`
`mod$$$`	`mod`	`(mod$$$ x divisor)`
`oddp$$$`	`oddp`	`(oddp$$$ x)`
`plusp$$$`	`plusp`	`(plusp$$$ x)`
`pushnew/incf$$$-acons`	`pushnew/incf-acons`	`(pushnew/incf$$$-acons 'x delta alist)`
`round$$$`	`round`	`(round$$$ x divisor)`
`truncate$$$`	`truncate`	`(truncate$$$ x divisor)`
`zerop$$$`	`zerop`	`(zerop$$$ x)`

The one-argument function **coerce$$$** provides convenient `long-float` coercion:

```
(setf x (coerce$$$ x))
```

3.3 Pseudo Probabilities

The :gbbopen-tools module provides a discretized fixnum representation for probability values in which probability values in the range [0.0..1.0] are represented as the nearest integers in the range [0..1000].

The following pseudo-probability operators are provided:

Operator	Operation	Example
%	the pseudo-probability	(% x)
+%	+&	(+% x y z)
-%	-&	(-% x y z)
1+%	1+&	(1+% x)
1-%	1-&	(1-% x)
*%	normalizing *&	(*% x y z)
/%	normalizing /&	(/% x y z)
=%	=&	(=% x y z)
/=%	/=&	(/=% x y z)
<%	<&	(<% x y z)
<=%	<=&	(<=% x y z)
>%	>&	(>% x y z)
>=%	>=&	(>=% x y z)
abs%	abs&	(abs% x)
bounded-value%	bounded-value&	(bounded-value% x y z)
ceiling%	normalizing ceiling&	(ceiling% x divisor)
decf%	decf&	(decf% x delta)
decf-%after	decf-after&	(decf%-after x delta)
decf/delete%-acons	decf/delete-acons&	(decf/delete%-acons x delta alist)
evenp%	evenp&	(evenp% x)
fceiling%	normalizing fceiling&	(fceiling% x divisor)
floor%	normalizing floor&	(floor% x divisor)
ffloor%	normalizing ffloor&	(ffloor% x divisor)
fround%	normalizing fround&	(fround% x divisor)
ftruncate%	normalizing ftruncate&	(ftruncate% x divisor)
incf%	incf&	(incf% x delta)
incf%-after	incf-after&	(incf%-after x delta)
max%	max&	(max% x y z)
min%	min&	(min% x y z)
minusp%	minusp&	(minusp% x)
mod%	mod&	(mod% x divisor)
oddp%	oddp&	(oddp% x)
plusp%	plusp&	(plusp% x)
pushnew/incf%-acons	pushnew/incf-acons&	(pushnew/incf%-acons 'x delta alist)
round%	normalizing round&	(round% x divisor)
truncate%	normalizing truncate&	(truncate% x divisor)
zerop%	zerop&	(zerop% x)

*% *pseudo-probability** ⇒ *pseudo-probability-product* [*Function*]

Purpose
Returns the product of *numbers*, performing pseudo-probability-normalization in the process.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
pseudo-probability A `fixnum` pseudo-probability
pseudo-probability-product A `fixnum` pseudo-probability

Returns
The `fixnum` pseudo-probability product of *numbers* or 1000 if no *numbers* are supplied.

See also
/% (page 151)
pprob2prob (page 155)
prob2pprob (page 156)

Examples
```
> (* 0.6 0.7)
0.42000002
> (*% 600 700)
420
> (* 0.6 0.7 0.8)
0.33600003
> (*% 600 700 800)
336
> (*%)
1000
>
```

/% *pseudo-probability-numerator pseudo-probability-denominator** ⇒ *quotient* [*Function*]

Purpose

Returns the quotient of dividing *pseudo-probability-numerator* by all of the *pseudo-probability-denominators*, performing pseudo-probability-normalization in the process.

Alternate syntax

/% *pseudo-probability-numerator* ⇒ *reciprocal*

Package :gbbopen-tools

Module :gbbopen-tools

Arguments

pseudo-probability-numerator A `fixnum` pseudo-probability
pseudo-probability-denominator A `fixnum` pseudo-probability
quotient A pseudo-probability-normalized `fixnum`
reciprocal A pseudo-probability-normalized `fixnum`

Returns

The pseudo-probability-normalized `fixnum` quotient (if one or more *pseudo-probability-denominators* is specified; otherwise the pseudo-probability-normalized `fixnum` reciprocal of *pseudo-probability-numerator*.

Description

pseudo-probability-normalization is performed by /%, but the `fixnum` result may not be a pseudo-probability. It is an error to specify more than one *pseudo-probability-denominator* value that leads to an intermediate result that is larger than a `fixnum`.

See also

*% (page 150)
pprob2prob (page 155)
prob2pprob (page 156)

Examples

```
> (/ 0.6 0.8)
0.75
> (/% 600 800)
750
> (/ 0.6 0.8 0.9)
0.8333334
> (/% 600 800 900)
833
> (/ 0.8 0.5)
1.6           ; not a probability
> (/% 800 500)
1600          ; not a pseudo-probability
> (/ 5.0)     ; not a probability
```

```
0.2
> (/% 5000)  ; not a pseudo-probability
200
> (/ 0.5)
2.0          ; not a probability
> (/% 500)
2000         ; not a pseudo-probability
>
```

/%

exp% *pseudo-probability-ln* ⇒ *pseudo-probability* [*Function*]

Purpose
Return the "pseudo natural logarithm" (pseudo-probability-ln) of a pseudo-probability value.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
pseudo-probability A `fixnum` pseudo-probability
pseudo-probability-ln A `fixnum` pseudo-probability-ln

Returns
The `fixnum` pseudo-probability-ln of *pseudo-probability*.

See also
ln% (page 154)
pprob2prob (page 155)
prob2pprob (page 156)

Examples
```
> (exp 0.0)
1.0
> (exp% 0)
1000
> (exp -6.907756)
9.999995e-4
> (exp% -6907756)
1
>
```

ln% *pseudo-probability** ⇒ *pseudo-probability-ln* [*Function*]

Purpose
Return the “pseudo natural logarithm” (pseudo-probability-ln) of a pseudo-probability value.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
pseudo-probability A `fixnum` pseudo-probability
pseudo-probability-ln A `fixnum` pseudo-probability-ln

Returns
The `fixnum` pseudo-probability-ln of *pseudo-probability*.

See also
exp% (page 153)
pprob2prob (page 155)
prob2pprob (page 156)

Examples
```
> (log 1.0)
0.0
> (ln% 1000)
0
> (log 0.001)
-6.9077554
> (ln% 1)
-6907756
>
```

pprob2prob *pseudo-probability* ⇒ *probability* [*Function*]

Purpose
Returns the value of *pseudo-probability* as a probability.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
pseudo-probability A `fixnum` pseudo-probability
probability A [0.0..1.0] probability

Returns
The converted *probability*.

See also
prob2pprob (page 156)

Examples
```
> (pprob2prob 0)
0.0
> (pprob2prob 1000)
1.0
> (pprob2prob 800)
0.8
>
```

prob2pprob *probability* ⇒ *pseudo-probability* [*Function*]

Purpose
Returns the value of *probability* as a pseudo-probability.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
probability A [0.0..1.0] probability
pseudo-probability A `fixnum` pseudo-probability

Returns
The converted *pseudo-probability*.

See also
pprob2prob (page 155)

Examples
```
> (prob2pprob 0.0)
0
> (prob2pprob 1.0)
1000
> (prob2pprob 0.8)
800
>
```

3.4 Date and Time

The `:gbbopen-tools` module includes useful date and time parsing and formatting entities.

In addition to UTC offsets, time-of-day entities support the following time-zone abbreviations for standard-time (non-daylight-savings) offsets:

Offset	Zone	Locations
1	WAT	West Africa (also Cape Verdes Islands, Atlantic Ocean)
2	AT	Azores
7/2	NST	Newfoundland
4	AST	Atlantic
5	EST	Eastern (North America)
6	CST	Central (North America)
7	MST	Mountain (North America)
8	PST	Pacific (North America)
9	AKST	Alaska
10	HAST	Hawaii-Aleutian
11	NT	Nome
12	IDLW	International Dateline West
0	GMT	Greenwich (also Portugal, Reykjavik (Iceland), Western Africa)
-1	CET	Central European (also Algeria, Nigeria, Angola)
-2	EET	Eastern European (also Finland, Balkans, Libya, Egypt, South Africa)
-3	MSK	Moscow (also Baghdad, Eastern Africa, Ethiopia, Kenya, Tanzania)
-4	ZP4	Samara (Russia Zone 3)
-5	ZP5	Yekaterinburg (Russia Zone 4)
-11/2	IST	Indian
-6	ZP6	Omsk (Russia Zone 5), Bangladesh)
-7	WAST	West Austrailian Standard (also Christmas Island, Krasnoyarsk (Russia Zone 6), Western Indonesia)
-8	AWST	Australian Western (also Irkutsk (Russia Zone 7), China, Hong Kong, Philippines, Central Indonesia)
-9	JST	Japan (also Yakutsk (Russia Zone 8), Korea, Eastern Indonesia)
-19/2	ACST	Australian Central
-10	AEST	Australian Eastern (also Vladivostok (Russia Zone 9), Papua New Guinea)
-21/2	NFT	Norfolk (Island)
-12	NZST	New Zealand (also Kamchatka (Russia), Fiji, Marshall Islands)

and for daylight-savings-time offsets:

Offset	Zone	Locations
5/2	NDT	Newfoundland Daylight
3	ADT	Atlantic Daylight
4	EDT	Eastern Daylight (North America)
5	CDT	Central Daylight (North America)
6	MDT	Mountain Daylight (North America)
7	PDT	Pacific Daylight (North America)
8	AKDT	Alaska Daylight
9	HADT	Hawaii-Aleutian Daylight
-1	BST	British Summer
-2	CEDT	Central European Daylight
-3	EEDT	Eastern European Daylight
-4	MSD	Moscow Daylight
-9	AWDT	Australian Western Daylight
-21/2	ACSD	Australian Central Daylight
-11	AEDT	Australian Eastern Daylight

Time-zone abbreviations used around the world are not unique or universal. The same hour offset can map onto multiple different zone abbreviations, and the same abbreviations are also used to refer to different zones. The above choices of supported abbreviations were made arbitrarily, and the use of UTC offsets is recommended for unambiguous textual representation of the time of day.

month-precedes-date [*Variable*]

Purpose
Control the default month and date ordering for GBBopen Tools date and time entities.

Package
`:gbbopen-tools` (home package is `:module-manager`)

Module
`:module-manager`

Value type
A generalized boolean

Initial value
True

See also

brief-date (page 161)
brief-date-and-time (page 163)
encode-date-and-time (page 168)
full-date-and-time (page 172)
parse-date (page 181)
parse-date-and-time (page 184)
parse-time (page 190)
very-brief-date (page 195)

Examples
Toggle between date formatting options:

```
> (let ((*month-precedes-date* 't))
    (brief-date-and-time))
"Feb 16 13:11"
> (let ((*month-precedes-date* nil))
    (brief-date-and-time))
"16 Feb 13:11"
>
```

Note
This variable is defined in the `:module-manager` module in order to make it available as early as possible.

time-first [*Variable*]

Purpose
Control the default date and time ordering for GBBopen Tools date and time entities.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Value type A generalized boolean

Initial value `nil`

See also
encode-date-and-time (page 168)
parse-date-and-time (page 184)

Example
Change the default date and time ordering to have the time precede the date:

```
> (let ((*time-first* 't))
    (parse-date-and-time "10:30pm 4/1/10"))
0
30
22
1
4
2010
nil
nil
14
>
```

year-first [*Variable*]

Purpose
Control the default year positioning for GBBopen Tools date entities.

Package
`:gbbopen-tools`

Module
`:gbbopen-tools`

Value type
A generalized boolean

Initial value
`nil`

See also

brief-date (page 161)
brief-date-and-time (page 163)
encode-date-and-time (page 168)
full-date-and-time (page 172)
parse-date (page 181)
parse-date-and-time (page 184)
very-brief-date (page 195)

Example
Change the default year positioning to have the year first:

```
> (let ((*year-first* 't))
    (full-date-and-time nil :all-numeric 't :year-first 't))
"2009/02/16 13:11"
> (let ((*year-first* 't))
    (parse-date "1-4-10" :year-first 't))
10
4
2001
6
>
```

brief-date `&optional` *universal-time* `&key` *time-zone month-precedes-date year-first include-year destination* ⇒ *result* [*Function*]

Purpose

Generate a brief date description.

Package

`:gbbopen-tools` (home package is `:module-manager`)

Module

`:module-manager`

Arguments

universal-time A Universal Time (default is `nil`, which is equivalent to the value returned by `(get-universal-time)`)

time-zone A time zone (default is `nil`, which is equivalent to the current time zone adjusted for daylight saving time)

month-precedes-date A generalized boolean (default is ***month-precedes-date***)

year-first A generalized boolean (default is ***year-first***)

include-year A generalized boolean (default is `t`)

destination Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)

result A string or `nil`

Returns

If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description

A 12-character description is generated (6 characters, if *include-year* is non-`nil`).

If *universal-time* is not supplied or is `nil`, the current time (as returned by `get-universal-time` is used.

If *time-zone* is not supplied or is `nil`, it defaults to the current time zone adjusted for daylight saving time. If *time-zone* is supplied, it is assumed to include any adjustment for daylight saving time.

If *month-precedes-date* is true, the month is presented in front of the date; otherwise the date precedes the month.

If *year-first* is supplied and is non-`nil`, the year is presented in front of the month and date; otherwise the year follows the month and date.

If *include-year* is supplied and is non-`nil`, the year is included in the presented time.

See also

month-precedes-date (page 158)
brief-date-and-time (page 163)
full-date-and-time (page 172)
http-date-and-time (page 176)
internet-text-date-and-time (page 177)
iso8601-date-and-time (page 179)
message-log-date-and-time (page 180)
very-brief-date (page 195)

Examples

Display the current date (with and without the year):

```
> (brief-date)
"Feb 16, 2008"
> (brief-date (get-universal-time) :include-year nil)
"Feb 16"
>
```

Display the date 10 days ago:

```
> (brief-date (- (get-universal-time) (* 60 60 24 10)))
"Feb 6, 2008"
>
```

Note

This function is loaded with the `:module-manager` module in order to to make it available as early as possible.

---

brief-date

brief-date-and-time `&optional` *universal-time* `&key` *time-zone month-precedes-date year-first include-seconds destination* ⇒ *result* [*Function*]

Purpose

Generate a brief date-and-time description.

Package

`:gbbopen-tools` (home package is `:module-manager`)

Module

`:module-manager`

Arguments

universal-time A Universal Time (default is `nil`, which is equivalent to the value returned by `(get-universal-time)`)

time-zone A time zone (default is `nil`, which is equivalent to the current time zone adjusted for daylight saving time)

month-precedes-date A generalized boolean (default is ***month-precedes-date***)

year-first A generalized boolean (default is ***year-first***)

include-seconds A generalized boolean (default is `nil`)

destination Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)

result A string or `nil`

Returns

If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description

A 12-character description (15 characters, if *include-seconds* is non-`nil`) is generated. If the *universal-time* value is within 120 days of the current time, the result *string* includes the time of day but not the year; otherwise, the year is included but not the time of day.

If *universal-time* is not supplied or is `nil`, the current time (as returned by `get-universal-time` is used.

If *time-zone* is not supplied or is `nil`, it defaults to the current time zone adjusted for daylight saving time. If *time-zone* is supplied, it is assumed to include any adjustment for daylight saving time.

If *month-precedes-date* is true, the month is presented in front of the date; otherwise the date precedes the month.

If *year-first* is supplied and is non-`nil`, the year is presented in front of the month and date; otherwise the year follows the month and date.

If *include-seconds* is supplied and is non-`nil`, seconds are included in the presented time.

See also

month-precedes-date (page 158)
brief-date (page 161)
full-date-and-time (page 172)
http-date-and-time (page 176)
internet-text-date-and-time (page 177)
iso8601-date-and-time (page 179)

message-log-date-and-time (page 180)
very-brief-date (page 195)

Examples

Display the current date and time:

```
> (brief-date-and-time)
"Feb 16 13:11"
>
```

Display the current date and time (with seconds):

```
> (brief-date-and-time nil :include-seconds 't)
"Feb 16 13:11:38"
>
```

Display the current date and time as GMT:

```
> (brief-date-and-time nil :time-zone 0)
"Feb 16 18:11"
>
```

The date and time 10 days ago:

```
> (brief-date-and-time (- (get-universal-time) (* 60 60 24 10)))
"Feb  6 13:11"
>
```

The date and time 125 days ago:

```
> (brief-date-and-time (- (get-universal-time) (* 60 60 24 125)))
"Oct 12, 2004"
>
```

The date and time 125 days ago (with seconds, but ignored because no time of day is included for dates that are not within 120 days of the given time):

```
> (brief-date-and-time (- (get-universal-time) (* 60 60 24 125))
:include-seconds 't)
"Oct 12, 2004   "
>
```

Note

This function is loaded with the `:module-manager` module in order to to make it available as early as possible.

---

brief-date-and-time

brief-duration *seconds* `&optional` *maximum-fields destination* ⇒ *result* [*Function*]

Purpose

Format a numeric time duration (in *seconds*) into brief descriptive text.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

seconds	A number
maximum-fields	An integer from 1–5 indicating maximum number of fields to include in the descriptive *string* (default is 5, indicating all fields should be included)
destination	Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)
result	A string or `nil`

Returns

If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description

The value of *seconds* is rounded to the nearest 100^{th} of a second before conversion. Fields omitted by *maximum-fields* cause appropriate rounding of the generated description.

See also

brief-run-time-duration (page 167)
parse-duration (page 188)
pretty-duration (page 192)
pretty-run-time-duration (page 194)

Examples

```
> (brief-duration 1000)
"16m 40s"
> (brief-duration -1000)
"-16m 40s"
> (brief-duration -1000.12345)
"-16m 40.12s"
> (brief-duration -1000.12543)
"-16m 40.13s"
> (brief-duration 166611.9)
"1d 22h 16m 51.91s"
> (brief-duration 166611.9 4)
"1d 22h 16m 52s"
> (brief-duration 166611.9 3)
"1d 22h 17m"
> (brief-duration 166611.9 2)
"1d 22h"
> (brief-duration 166611.9 1)
"2d"
```

```
> (brief-duration 31556952)
"365d 5h 49m 12s"
>
```

---

brief-duration

brief-run-time-duration *internal-time-units* `&optional` *maximum-fields destination* [*Function*]
⇒ *result*

Purpose
Format a run-time duration (in *internal-time-units*) into brief descriptive text.

Package
`:gbbopen-tools`

Module
`:gbbopen-tools`

Arguments
internal-time-units	A number
maximum-fields	An integer from 1–5 indicating maximum number of fields to include in the descriptive *string* (default is 5, indicating all fields should be included)
destination	Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)
result	A string or `nil`

Returns
If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description
The *internal-time-units* run-time duration is rounded to the nearest 100^{th} of a second before conversion. Fields omitted by *maximum-fields* cause appropriate rounding of the generated description.

See also
brief-duration (page 165)
parse-duration (page 188)
pretty-duration (page 192)
pretty-run-time-duration (page 194)

Examples
```
> internal-time-units-per-second
1000
> (brief-run-time-duration 1000)
"1s"
> (brief-run-time-duration 5)
"0s"
> (brief-run-time-duration 6)
"0.01s"
> most-positive-fixnum
536870911
> (brief-run-time-duration most-positive-fixnum)
"6d 5h 7m 50.94s"
>
```

encode-date-and-time *string* `&key` *start end junk-allowed date-separators time-separators month-precedes-date year-first default-to-current-year time-first time-zone* ⇒ *universal-time*, pos [*Function*]

Purpose

Parse and then encode a date-and-time specification string to a Universal Time.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

string	A string
start	Starting index into *string* (default is `0`)
end	Ending index into *string* (default is `nil`, meaning end of *string*)
junk-allowed	A generalized boolean (default is `nil`)
date-separators	A sequence of characters that are skipped and separate the date, month, and year fields in *string*, if needed (default is `"-/ ,"`)
time-separators	A sequence of characters that are skipped and separate the hour, minute, and second fields in *string*, if needed (default is `" :"`)
month-precedes-date	A generalized boolean (default is ***month-precedes-date***)
year-first	A generalized boolean (default is ***year-first***)
default-to-current-year	A generalized boolean (default is `nil`)
time-first	A generalized boolean (default is ***time-first***)
time-zone	A time zone (default is `nil`, which is equivalent to the current time zone adjusted for daylight saving time)
universal-time	A Universal Time
position	A index in *string*

Returns

Two values: *universal-time* and *position*

Errors

If *junk-allowed* is false, an error is signaled if a numeric field in *string* does not consist entirely of the representation of a integer, possibly surrounded on either side by characters in *separators*.

Description

Both the month and date must be specified in *string*, optionally followed by the year and the time of day. The month can be a numeric value (1–12), a three-letter abbreviation, or the full month name. If the month is specified numerically, then the value of *month-precedes-date* is used to determine the month and date ordering. If no year is specified in *string* and *default-to-current-year* is `nil`, the current calendar year is assumed, unless the specified month and date have passed, in which case the next year is assumed. If no year is specified in *string* and *default-to-current-year* is true, the current calendar year is always assumed. If no hour, minute, or second values are specified, they default to zero.

If *month-precedes-date* is true, the month is expected before the date; otherwise the date is expected to follow the month.

If *year-first* is supplied and is non-`nil`, the year must be provided and it is expected before the month and date; otherwise the year (if provided) is expected to follow the month and date.

If a time-zone is specified in *string*, it is used when encoding the *universal-time* value. Otherwise, if a non-`nil` *time-zone* argument was supplied, it used for the encoding. Otherwise, the current time zone adjusted for daylight saving time is used.

If *time-first* is true, the time-of-day is expected before the date; otherwise the time-of-day is expected to follow the date.

The returned *position* is the index within *string* where the parse ended.

See also

month-precedes-date (page 158)
encode-date-and-time (page 168)
encode-time-of-day (page 171)
parse-date (page 181)
parse-date-and-time (page 184)
parse-duration (page 188)
parse-time (page 190)

Examples

```
> (encode-date-and-time "1 Apr 2010")
3479083200
10
> (encode-date-and-time "April 1, 2010 10:30")
3479121000
19
> (encode-date-and-time "Thu 1 Apr 2010")
3479083200
14
> (encode-date-and-time "Thursday, April 1, 2010 10:30")
3479121000
29
> (encode-date-and-time "4/1/10 10:30pm")
3479164200
14
> (encode-date-and-time "10:30pm 4/1/10" :time-first 't)
3479164200
14
> (encode-date-and-time "Apr 1 2010 10:30 EDT")
3479121000
20
> (encode-date-and-time "Apr 1 2010 10:30PM EDT")
3479164200
22
> (encode-date-and-time "1 Apr 2010 10:30 IST")
3479086800
20
> (encode-date-and-time "1 Apr 2010 10:30" :time-zone -11/2)
3479121000
20
```

```
> (encode-date-and-time "1 Apr 2010 10:30 EDT" :time-zone -11/2)
3479086800
16
> (encode-date-and-time "April 1, 2010 10:30 UTC-4")
3479121000
25
>
```

REPL Note

The equivalent of:

```
(print (encode-date-and-time string))
```

can be invoked using the REPL command `:ut` *string*.

---

encode-date-and-time

encode-time-of-day *second minute hour* `&optional` *universal-time* ⇒ *universal-time* [*Function*]

Purpose

Return a Universal Time representing a specified time-of-day.

Package

`:gbbopen-tools` (or `:portable-threads` if Portable Threads is used without GBBopen Tools)

Module

`:gbbopen-tools` (or `:portable-threads` if Portable Threads is used without GBBopen Tools)

Arguments

second	An integer between 0 and 59, inclusive
minute	An integer between 0 and 59, inclusive
hour	An integer between 0 and 23, inclusive
universal-time	A Universal Time (default is `nil`, which is equivalent to the value returned by `(get-universal-time)`)

Description

If the specified time-of-date has already passed (relative to the *universal-time* value), the next day is assumed.

See also

encode-date-and-time (page 168)

Examples

Schedule a scheduled function that prints `"It's quitting time!"` every day at 5pm:

```
> (schedule-function
    (make-scheduled-function
      #'(lambda (scheduled-function)
          (declare (ignore scheduled-function))
          (print "It's quitting time!"))
      :name 'quitting-time)
     (encode-time-of-day 0 0 17) :repeat-interval #.(* 24 60 60))
>
```

Verbosely change `quitting-time` to 5:30pm every day:

```
> (schedule-function 'quitting-time (encode-time-of-day 0 30 17)
    :repeat-interval #.(* 24 60 60)
    :verbose 't)
;; Unscheduling #<scheduled-function quitting-time [17:00:00]>...
;; Scheduling #<scheduled-function quitting-time [17:30:00]>
;; as the next scheduled-function...
>
```

full-date-and-time `&optional` *universal-time* `&key` *time-zone daylight-savings-p all-numeric separator full-names month-precedes-date year-first include-day include-seconds include-time-zone utc-offset-only 12-hour destination* ⇒ *result* [*Function*]

Purpose

Generate a date-and-time description.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

universal-time	A Universal Time (default is `nil`, which is equivalent to the value returned by `(get-universal-time)`)
time-zone	A time zone (default is `nil`, which is equivalent to the current time zone adjusted for daylight saving time)
daylight-savings-p	A generalized boolean (default is `nil`)
all-numeric	A generalized boolean (default is `nil`)
separator	A character (default is `#\/`)
full-names	A generalized boolean (default is `nil`)
month-precedes-date	A generalized boolean (default is ***month-precedes-date***)
year-first	A generalized boolean (default is ***year-first***)
include-day	A generalized boolean (default is `nil`)
include-time-zone	A generalized boolean (default is `nil`)
utc-offset-only	A generalized boolean (default is `nil`)
12-hour	A generalized boolean (default is `nil`)
destination	Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)
result	A string or `nil`

Returns

If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description

A 17-character description (longer if *full-names*, *include-day*, *include-seconds*, *include-time-zone*, or *utc-offset-only* are non-`nil`) is generated.

If *universal-time* is not supplied or is `nil`, the current time (as returned by `get-universal-time` is used.

If *time-zone* is not supplied or is `nil`, it defaults to the current time zone adjusted for daylight saving time. If a *time-zone* is supplied, it is assumed to include any adjustment for daylight saving time unless *daylight-savings-p* is specified and is non-`nil`.

If *all-numeric* is supplied and is non-`nil`, the month is indicated by its numeric value, and the date, month, and year are separated by the *separator* character.

If *full-names* is supplied and is non-`nil`, the full name is generated rather than the abbreviated name for the month (if *include-day* is `nil`) and for the day of the week, (if *include-day* is non-`nil`).

If *month-precedes-date* is true, the month is presented in front of the date; otherwise the date precedes the month.

If *year-first* is supplied and is non-`nil`, the year is presented in front of the month and date; otherwise the year follows the month and date.

If *include-day* is supplied and is non-`nil`, the day of the week is included in front of the date and time.

If *include-seconds* is supplied and is non-`nil`, seconds are included in the presented time.

If either *include-time-zone* or *utc-offset-only* is true, a time-zone specification is appended to the date-and-time presentation. If *utc-offset-only* is true, the time zone is presented as a UTC offset—even if a time-zone abbreviation supported by GBBopen Tools is available for the time zone. If a *time-zone* is supplied, the value of *daylight-savings-p* is used when generating a non-UTC time zone abbreviation; otherwise, the local daylight-savings setting for the *universal-time* value (as determined by `decode-universal-time`) is used, and the *daylight-savings-p* argument is ignored.

If *12-hour* is supplied and is non-`nil`, the time-of-day is presented in AM/PM format.

See also

month-precedes-date (page 158)
brief-date (page 161)
brief-date-and-time (page 163)
encode-date-and-time (page 168)
http-date-and-time (page 176)
internet-text-date-and-time (page 177)
iso8601-date-and-time (page 179)
message-log-date-and-time (page 180)
parse-date-and-time (page 184)
very-brief-date (page 195)

Examples

Display the current date and time:

```
> (full-date-and-time)
"Feb 16 2009 13:11"
>
```

Display the current date and time (with seconds):

```
> (full-date-and-time nil :include-seconds 't)
"Feb 16 2009 13:11:38"
>
```

Display the current date and time (with the day of the week):

```
> (full-date-and-time nil :include-day 't)
"Mon Feb 16 2009 13:11"
>
```

Display the current date and time (with no abbreviations and with the day of the week):

```
> (full-date-and-time nil :full-names 't :include-day 't)
"Monday, February 16, 2009 13:11"
>
```

Display the current date and time (with no abbreviations, with the month following the date, and with the day of the week):

```
> (full-date-and-time nil :full-names 't :include-day 't
                          :month-precedes-date nil)
"Monday, 16 February, 2009 13:11"
>
```

Display the current date and time (with no abbreviations, with the year first and the month following the date, and with the day of the week):

```
> (full-date-and-time nil :full-names 't :include-day 't
                          :year-first 't :month-precedes-date nil)
"2009, 16 February, Monday 13:11"
>
```

Display the current date and time (all numeric, with the abbreviated day of the week):

```
> (full-date-and-time nil :all-numeric 't :include-day 't)
"Mon 02/16/2009 13:11"
>
```

Display the current date and time (all numeric, with hyphen (minus-sign) separators):

```
> (full-date-and-time nil :all-numeric 't :separator #)
"02-16-2009 13:11"
>
```

Display the current date and time (all numeric, with the year first and the abbreviated day of the week):

```
> (full-date-and-time nil :all-numeric 't :year-first 't :include-day 't)
"2009/02/16 Mon 13:11"
>
```

Display the current date and time (all numeric, with the month following the date, and with the full day of the week):

```
> (full-date-and-time nil :all-numeric 't :include-day 't
                          :full-names 't :month-precedes-date nil)
"Monday, 16/02/2009 13:11"
>
```

Display the current date and time (with time zone):

```
> (full-date-and-time nil :include-time-zone 't)
"Feb 16 2009 13:11 EST"
>
```

Display the current date and time (in 12-hour format with time zone):

```
> (full-date-and-time nil :12-hour 't :include-time-zone 't)
"Feb 16 2009 1:11PM EST"
>
```

Display the current date and time (with UTC-offset time zone):

```
> (full-date-and-time nil :utc-offset-only 't)
"Feb 16 2009 13:11 UTC-5"
>
```

Display the current date and time (with seconds and time zone):

```
> (full-date-and-time nil
    :include-seconds 't
    :include-time-zone 't)
"Feb 16 2009 13:11:38 EST"
>
```

Display the current date and time as GMT:

```
> (full-date-and-time nil :time-zone 0)
"Feb 16 2009 18:11"
>
```

Display the current date and time as GMT (with time zone):

```
> (full-date-and-time nil
    :time-zone 0
    :include-time-zone 't)
"Feb 16 2009 18:11 GMT"
>
```

The date and time 10 days ago:

```
> (full-date-and-time (- (get-universal-time) (parse-duration "10 days")))
"Feb  6 2009 13:11"
>
```

The date and time 125 days ago:

```
> (full-date-and-time (- (get-universal-time) (parse-duration "125 days")))
"Oct 12 2008 13:11"
>
```

The date and time 125 days ago (with seconds):

```
> (full-date-and-time (- (get-universal-time) (parse-duration "125d"))
    :include-seconds 't)
"Oct 12 2008 13:11:38"
>
```

REPL Note

The equivalent of:

```
(print (full-date-and-time universal-time
          :include-seconds 't))
```

can be invoked using the REPL command `:ut` *universal-time*.

full-date-and-time

http-date-and-time `&optional` *universal-time* `&key` *destination* ⇒ *result* [*Function*]

Purpose
Convert a Universal Time value into HTTP/1.1 time-stamp format.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
universal-time	A Universal Time (default is `nil`, which is equivalent to the value returned by `(get-universal-time)`)
destination	Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)
result	A string or `nil`

Returns
If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description
If *universal-time* is not supplied or is `nil`, the current time (as returned by `get-universal-time` is used.

See also

brief-date (page 161)
brief-date-and-time (page 163)
full-date-and-time (page 172)
http-date-and-time (page 176)
internet-text-date-and-time (page 177)
iso8601-date-and-time (page 179)
message-log-date-and-time (page 180)
very-brief-date (page 195)

Example
```
> (http-date-and-time)
"Wed, 05 Aug 2009 17:29:26 GMT"
>
```

internet-text-date-and-time `&optional` *universal-time* `&key` *time-zone* *daylight-savings-p utc-offset-only destination* ⇒ *result* [*Function*]

Purpose

Convert a Universal Time value into Internet Text Message format.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

universal-time	A Universal Time (default is `nil`, which is equivalent to the value returned by `(get-universal-time)`)
time-zone	A time zone (default is `nil`, which is equivalent to the current time zone adjusted for daylight saving time)
daylight-savings-p	A generalized boolean (default is `nil`)
utc-offset-only	A generalized boolean (default is `nil`)
destination	Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)
result	A string or `nil`

Returns

If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description

If *universal-time* is not supplied or is `nil`, the current time (as returned by `get-universal-time` is used.

If *utc-offset-only* is true, the time zone is represented as a UTC offset—even if a time-zone abbreviation supported by GBBopen Tools is available for the time zone. If a *time-zone* is supplied, the value of *daylight-savings-p* is used when generating a non-UTC time zone abbreviation; otherwise, the local daylight-savings setting for the *universal-time* value (as determined by `decode-universal-time`) is used, and the *daylight-savings-p* argument is ignored.

See also

brief-date (page 161)
brief-date-and-time (page 163)
full-date-and-time (page 172)
http-date-and-time (page 176)
iso8601-date-and-time (page 179)
message-log-date-and-time (page 180)
very-brief-date (page 195)

Examples

```
> (internet-text-date-and-time)
"Sat, 17 May 2008 04:02:49 -0400 (EDT)"
> (internet-text-date-and-time nil :time-zone 0)
"Sat, 17 May 2008 08:02:50 -0000 (GMT)"
>
```

internet-text-date-and-time

iso8601-date-and-time `&optional` *universal-time* `&key` *destination* ⇒ *result* [*Function*]

Purpose
Convert a Universal Time value into ISO8601 (XML dateTime) format.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
universal-time	A Universal Time (default is `nil`, which is equivalent to the value returned by `(get-universal-time)`)
destination	Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)
result	A string or `nil`

Returns
If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description
If *universal-time* is not supplied or is `nil`, the current time (as returned by `get-universal-time` is used.

See also

brief-date (page 161)
brief-date-and-time (page 163)
full-date-and-time (page 172)
http-date-and-time (page 176)
internet-text-date-and-time (page 177)
message-log-date-and-time (page 180)
very-brief-date (page 195)

Example
```
> (iso8601-date-and-time)
"2008-05-17T08:02:49Z"
>
```

message-log-date-and-time `&optional` *universal-time* `&key` *destination* ⇒ *result* [*Function*]

Purpose

Convert a Universal Time value into "message log" (`MMM DD HH:MM:SS`) format.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

universal-time	A Universal Time (default is `nil`, which is equivalent to the value returned by `(get-universal-time)`
destination	Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)
result	A string or `nil`

Returns

If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description

If *universal-time* is not supplied or is `nil`, the current time (as returned by `get-universal-time` is used.

See also

brief-date (page 161)
brief-date-and-time (page 163)
full-date-and-time (page 172)
http-date-and-time (page 176)
internet-text-date-and-time (page 177)
iso8601-date-and-time (page 179)
very-brief-date (page 195)

Example

```
> (message-log-date-and-time)
"May 17 04:02:49"
>
```

parse-date *string* `&key` *start end junk-allowed separators month-precedes-date year-first default-to-current-year* ⇒ *date, month, year, position* [*Function*]

Purpose

Parse a date-specification string.

Package

`:gbbopen-tools` (home package is `:module-manager`)

Module

`:module-manager`

Arguments

string	A simple string
start	Starting index into *string* (default is `0`)
end	Ending index into *string* (default is `nil`, meaning end of *string*)
junk-allowed	A generalized boolean (default is `nil`)
separators	A sequence of characters that are skipped and separate the date, month, and year fields in *string*, if needed (default is `"-/ ,"`)
month-precedes-date	A generalized boolean (default is ***month-precedes-date***)
year-first	A generalized boolean (default is ***year-first***)
default-to-current-year	A generalized boolean (default is `nil`)
date	An integer between 1 and up to 31, inclusive, depending on the month and year
month	An integer between 1 and 12, inclusive
year	An integer
position	A index in *string*

Returns

Four values: *date*, *month*, *year*, and *position*

Errors

If *junk-allowed* is false, an error is signaled if a numeric field in *string* does not consist entirely of the representation of a integer, possibly surrounded on either side by characters in *separators*.

Description

Both the month and date must be specified in *string*, optionally followed by the year and the time of day. The month can be a numeric value (1–12), a three-letter abbreviation, or the full month name. If the month is specified numerically, then the value of *month-precedes-date* is used to determine the month and date ordering. If no year is specified in *string* and *default-to-current-year* is `nil`, the current calendar year is assumed, unless the specified month and date have passed, in which case the next year is assumed. If no year is specified in *string* and *default-to-current-year* is true, the current calendar year is always assumed.

The returned *position* is the index within *string* where the parse ended.

See also

month-precedes-date (page 158)
encode-date-and-time (page 168)
encode-time-of-day (page 171)

parse-date-and-time (page 184)
parse-duration (page 188)
parse-time (page 190)

Examples

```
> (parse-date "1 Apr 2010")
1
4
2010
10
> (parse-date "April 1, 2010")
1
4
2010
13
> (parse-date "Thu 1 Apr 2010")
1
4
2010
14
> (parse-date "Thursday, April 1, 2010")
1
4
2010
23
> (parse-date "1-4-10" :month-precedes-date 't)
4
1
2010
6
> (parse-date "1-4-10" :month-precedes-date nil)
1
4
2010
6
> (parse-date "1-4-10" :month-precedes-date 't :year-first 't)
10
4
2001
6
> (parse-date "1-4-10" :month-precedes-date nil :year-first 't)
4
10
2001
6
> (parse-date "4 Jul") ;; entered May 1, 2008
4
7
2008
5
> (parse-date "4 Jul") ;; entered August 1, 2008
```

```
4
7
2009
5
> (parse-date "4 Jul"  ;; entered August 1, 2008
    :default-to-current-year 't)
4
7
2008
5
> (parse-date "4/7  " :month-precedes-date nil)
4
7
2008
5
> (parse-date "4/7  junk" :junk-allowed 't)
7
4
2008
5
>
```

Note

This function is loaded with the `:module-manager` module in order to to make it available as early as possible.

---

parse-date

parse-date-and-time *string* `&key` *start end junk-allowed date-separators time-separators month-precedes-date year-first default-to-current-year time-first* ⇒ *second, minute, hour, date, month, year, time-zone, daylight-savings-p, pos* [*Function*]

Purpose

Parse a date-and-time specification string.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

string	A simple string
start	Starting index into *string* (default is `0`)
end	Ending index into *string* (default is `nil`, meaning end of *string*)
junk-allowed	A generalized boolean (default is `nil`)
date-separators	A sequence of characters that are skipped and separate the date, month, and year fields in *string*, if needed (default is `"-/ ,"`)
time-separators	A sequence of characters that are skipped and separate the hour, minute, and second fields in *string*, if needed (default is `" :"`)
month-precedes-date	A generalized boolean (default is ***month-precedes-date***)
year-first	A generalized boolean (default is ***year-first***)
default-to-current-year	A generalized boolean (default is `nil`)
time-first	A generalized boolean (default is ***time-first***)
second	An integer between 0 and up to 59, inclusive
minute	An integer between 0 and up to 59, inclusive
hour	An integer between 0 and up to 23, inclusive
date	An integer between 1 and up to 31, inclusive, depending on the month and year
month	An integer between 1 and 12, inclusive
year	An integer
time-zone	A time zone: a rational multiple of 1/3600 between -24 and 24 that represents the number of hours offset from GMT
daylight-savings-p	A generalized boolean
position	A index in *string*

Returns

Nine values: *second*, *minute*, *hour*, *date*, *month*, *year*, *time-zone*, *daylight-savings-p*, and *position*

Errors

If *junk-allowed* is false, an error is signaled if a numeric field in *string* does not consist entirely of the representation of a integer, possibly surrounded on either side by characters in *separators*.

Description

Both the month and date must be specified in *string*, optionally followed by the year and the time of day. The month can be a numeric value (1–12), a three-letter abbreviation, or the full month name. If the month is specified numerically, then the value of *month-precedes-date* is used to determine the month and date ordering. If no year is specified in *string* and *default-to-current-year* is `nil`, the current calendar year is assumed, unless the specified month and date have passed, in which case the next year is assumed. If no year is specified in *string* and *default-to-current-year* is true, the current calendar year is always assumed. If no hour, minute, or second values are specified, they default to zero. The values returned for *time-zone* and *daylight-savings-p* will be `nil` unless a time-zone is specified in *string*.

If *month-precedes-date* is true, the month is expected before the date; otherwise the date is expected to follow the month.

If *year-first* is supplied and is non-`nil`, the year must be provided and it is expected before the month and date; otherwise the year (if provided) is expected to follow the month and date.

If a time-zone is specified in *string*, it is used when encoding the *universal-time* value. Otherwise, if a non-`nil` *time-zone* argument was supplied, it used for the encoding. Otherwise, the current time zone adjusted for daylight saving time is used.

If *time-first* is true, the time-of-day is expected before the date; otherwise the time-of-day is expected to follow the date.

The returned *position* is the index within *string* where the parse ended.

See also

month-precedes-date (page 158)
encode-date-and-time (page 168)
encode-time-of-day (page 171)
full-date-and-time (page 172)
parse-date (page 181)
parse-duration (page 188)
parse-time (page 190)

Examples

```
> (parse-date-and-time "1 Apr 2010")
0
0
0
1
4
2010
nil
nil
10
> (parse-date-and-time "April 1, 2010 10:30")
0
30
10
1
4
2010
```

```
nil
nil
19
> (parse-date-and-time "4/1/10 10:30pm")
0
30
22
1
4
2010
nil
nil
14
> (parse-date-and-time "10:30pm 4/1/10" :time-first 't)
0
30
22
1
4
2010
nil
nil
14
> (parse-date-and-time "Apr 1 2010 10:30 EDT")
0
30
10
1
4
2010
4
t
20
> (parse-date-and-time "1 Apr 2010 10:30 IST")
0
30
10
1
4
2010
-11/2
nil
20
> (parse-date-and-time "April 1, 2010 10:30 UTC-7")
0
30
10
1
4
2010
7
nil
```

```
25
>
```

parse-date-and-time

parse-duration *string* `&key` *start end separators* ⇒ *seconds* [*Function*]

Purpose
Parse a time-duration-specification string.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
string	A simple string
start	Starting index into *string* (default is `0`)
end	Ending index into *string* (default is `nil`, meaning end of *string*)
separators	A sequence of characters that are skipped between specification items (default is `" ,"`)

Returns
The duration, in seconds

Description
The following time-duration-unit specifiers are recognized (both singular and plural):

`"second"`	(or `"sec"` or `"s"`)
`"minute"`	(or `"min"` or `"m"`)
`"hour"`	(or `"hr"` or `"h"`)
`"day"`	(or `"d"`)
`"week"`	(or `"wk"`)
`"month"`	(or `"mon"`)
`"year"`	(or `"yr"`)

A `"month"` is interpreted as exactly 30 days; a `"year"` as 365 days.

See also

brief-duration (page 165)
encode-date-and-time (page 168)
encode-time-of-day (page 171)
parse-date (page 181)
parse-date-and-time (page 184)
pretty-duration (page 192)
parse-time (page 190)

Examples
```
> (parse-duration "2 minutes")
60
> (parse-duration "-2min")
-60
> (parse-duration "2m")
60
> (parse-duration "365 days, 5 hours, 49 minutes, 12 seconds"
```

```
31556952
> (parse-duration "365d 5h 49m 12s")
31556952
> (parse-duration "365d5h49m12s")
31556952
> (parse-duration "1min -2secs")
58
> (parse-duration "1m-2s")
58
> (parse-duration "1/2hr")
1800
> (parse-duration "0.5hr")
1800.0
> (parse-duration "2 months")
5184000
> (parse-duration "3 wks")
1814400
```

---

parse-duration

parse-time *string* `&key` *start end junk-allowed separators* ⇒ *second, minute, hour, time-zone, daylight-savings-p, pos* [*Function*]

Purpose

Parse a time-of-day specification string.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

string	A simple string
start	Starting index into *string* (default is `0`)
end	Ending index into *string* (default is `nil`, meaning end of *string*)
junk-allowed	A generalized boolean (default is `nil`)
separators	A sequence of characters that are skipped and separate the hour, minute, and second fields in *string*, if needed (default is `" :"`)
second	An integer between 0 and up to 59, inclusive
minute	An integer between 0 and up to 59, inclusive
hour	An integer between 0 and up to 23, inclusive
time-zone	A time zone: a rational multiple of 1/3600 between -24 and 24 that represents the number of hours offset from GMT
daylight-savings-p	A generalized boolean
position	A index in *string*

Returns

Six values: *second*, *minute*, *hour*, *time-zone*, *daylight-savings-p*, and *position*

Errors

If *junk-allowed* is false, an error is signaled if a numeric field in *string* does not consist entirely of the representation of a integer, possibly surrounded on either side by characters in *separators*.

Description

If no hour, minute, or second values are specified in *string*, they default to zero. The values returned for *time-zone* and *daylight-savings-p* will be `nil` unless a time-zone is specified in *string*.

The returned *position* is the index within *string* where the parse ended.

See also

encode-date-and-time (page 168)
encode-time-of-day (page 171)
parse-date (page 181)
parse-date-and-time (page 184)
parse-duration (page 188)
parse-time (page 190)

Examples

```
> (parse-time "10:30")
0
30
10
nil
nil
5
> (parse-time "10:30pm")
0
30
22
nil
nil
7
> (parse-time "10:30 EDT")
0
30
10
4
t
9
> (parse-time "10:30 IST")
0
30
10
-11/2
nil
9
> (parse-time "10:30 UTC-7")
0
30
10
7
nil
11
>
```

parse-time

pretty-duration *seconds* `&optional` *maximum-fields destination* ⇒ *result* [*Function*]

Purpose

Format a numeric time duration (in *seconds*) into descriptive text.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

seconds	A number
maximum-fields	An integer from 1–5 indicating maximum number of fields to include in the descriptive *string* (default is 5, indicating all fields should be included)
destination	Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)
result	A string or `nil`

Returns

If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description

The value of *seconds* is rounded to the nearest 100^{th} of a second before conversion. Fields omitted by *maximum-fields* cause appropriate rounding of the generated description.

See also

brief-duration (page 165)
brief-run-time-duration (page 167)
parse-duration (page 188)
pretty-run-time-duration (page 194)

Examples

```
> (pretty-duration 1000)
"16 minutes, 40 seconds"
> (pretty-duration -1000)
"minus 16 minutes, 40 seconds"
> (pretty-duration -1000.12345)
"minus 16 minutes, 40.12 seconds"
> (pretty-duration -1000.12543)
"minus 16 minutes, 40.13 seconds"
> (pretty-duration 166611.9)
"1 day, 22 hours, 16 minutes, 51.91 seconds"
> (pretty-duration 166611.9 4)
"1 day, 22 hours, 16 minutes, 52 seconds"
> (pretty-duration 166611.9 3)
"1 day, 22 hours, 17 minutes"
> (pretty-duration 166611.9 2)
"1 day, 22 hours"
> (pretty-duration 166611.9 1)
"2 days"
```

```
> (pretty-duration 31556952)
"365 days, 5 hours, 49 minutes, 12 seconds"
>
```

pretty-duration

pretty-run-time-duration *internal-time-units* `&optional` *maximum-fields destination* [*Function*]
⇒ *result*

Purpose
Format a run-time duration (in *internal-time-units*) into descriptive text.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
internal-time-units	A number
maximum-fields	An integer from 1–5 indicating maximum number of fields to include in the descriptive *string* (default is 5, indicating all fields should be included)
destination	Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)
result	A string or `nil`

Returns
If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description
The *internal-time-units* run-time duration is rounded to the nearest 100^{th} of a second before conversion. Fields omitted by *maximum-fields* cause appropriate rounding of the generated description.

See also

brief-duration (page 165)
brief-run-time-duration (page 167)
parse-duration (page 188)
pretty-duration (page 192)

Examples
```
> internal-time-units-per-second
1000
> (pretty-run-time-duration 1000)
"1 second"
> (pretty-run-time-duration 5)
"0 seconds"
> (pretty-run-time-duration 6)
"0.01 seconds"
> most-positive-fixnum
536870911
> (pretty-run-time-duration most-positive-fixnum)
"6 days, 5 hours, 7 minutes, 50.94 seconds"
>
```

very-brief-date `&optional` *universal-time* `&key` *time-zone month-precedes-date year-first include-year separator destination* ⇒ *result* [*Function*]

Purpose

Generate a very brief date description.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

universal-time A Universal Time (default is `nil`, which is equivalent to the value returned by `(get-universal-time)`)

time-zone A time zone (default is `nil`, which is equivalent to the current time zone adjusted for daylight saving time)

month-precedes-date A generalized boolean (default is ***month-precedes-date***)

year-first A generalized boolean (default is ***year-first***)

include-year A generalized boolean (default is `t`)

separator A character (default is `#\/`)

destination Either `nil`, `t`, a stream, or a string with a fill pointer (default is `nil`)

result A string or `nil`

Returns

If *destination* is non-`nil`, then `nil`; otherwise, a string.

Description

If *universal-time* is not supplied or is `nil`, the current time (as returned by `get-universal-time` is used.

If *time-zone* is not supplied or is `nil`, it defaults to the current time zone adjusted for daylight saving time. If *time-zone* is supplied, it is assumed to include any adjustment for daylight saving time.

If *month-precedes-date* is true, the month is presented in front of the date; otherwise the date precedes the month.

If *year-first* is supplied and is non-`nil`, the year is presented in front of the month and date; otherwise the year follows the month and date.

If *include-year* is supplied and is non-`nil`, the year is included in the presented time.

See also

month-precedes-date (page 158)
brief-date-and-time (page 163)
full-date-and-time (page 172)
http-date-and-time (page 176)
internet-text-date-and-time (page 177)
iso8601-date-and-time (page 179)
message-log-date-and-time (page 180)

Examples

Display the current date (with and without the year):

```
> (very-brief-date)
"2/16/2008"
> (very-brief-date (get-universal-time) :include-year nil)
"2/16"
>
```

---

very-brief-date

3.5 Offset Universal Time

Common Lisp has three time representations: Decoded Time, Universal Time, and Internal Time. Universal Time (UT) allows specific points in time from the beginning of 1900 to be represented with one-second resolution (ignoring leap seconds). The disadvantage of absolute Universal Time values is that they are bignums in most Common Lisp implementations.

To reduce computation and storage requirements, a fourth time representation, Offset Universal Time (OT), can be used. Offset Universal Time is Universal Time that is offset by an integer time-base value so that the most often used Offset Universal Time values in an application are fixnums.

Nearly all Common Lisp implementations provide fixnums of at least 30 bits (34 years of time range) or more, but CLISP on 32-bit machines provides only 25 bits (388 days). The ANSI standard requires an implementation to provide fixnums with at least 16 bits (only 18 hours), but fortunately Common Lisp implementations are considerably more generous!

When developing applications that must represent time values that exceed the fixnum range, it is important to choose the best time-base offset value to reduce bignum costs. Of course, existing Offset Universal Time values will appear shifted if the time-base offset value is changed.

ot-base [*Variable*]

Purpose

Holds the Offset Universal Time time-base value.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Value type

An Offset Universal Time time-base value

Initial value

Must be set by using **set-ot-base**

See also

check-ot-base (page 199)
ot2ut (page 200)
set-ot-base (page 202)
ut2ot (page 204)

check-ot-base `&optional` *suppress-warning* ⇒ *boolean* [*Function*]

Purpose

Check if the current time can be represented as a `fixnum` given the current Offset Universal Time time-base value.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

suppress-warning A generalized boolean (default is `nil`)
boolean A generalized boolean

Returns

True if the current time can be represented as a `fixnum` Offset Universal Time value; nil otherwise.

Errors

The Offset Universal Time time-base value has not been set.

See also

ot-base (page 198)
ot2ut (page 200)
set-ot-base (page 202)
ut2ot (page 204)

Examples

Set the time base for Offset Universal Time to today and check:

```
> (set-ot-base)
3410655616
> (check-ot-base)
t
>
```

Set the time base for Offset Universal Time to January 1, 1900 and check:

```
> (set-ot-base 1 1 1900)
16777216
> (check-ot-base)
;; Warning: The current time represented as an Offset~Universal~Time is not
a fixnum.
nil
> (check-ot-base 't)
nil
>
```

ot2ut *offset-universal-time* ⇒ *universal-time* [*Function*]

Purpose
Convert an Offset Universal Time value to a Universal Time value.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
offset-universal-time An Offset Universal Time value
universal-time A Universal Time

Returns
The equivalent Universal Time value

Errors
The Offset Universal Time time-base value has not been set.

See also
ot-base (page 198)
check-ot-base (page 199)
ut2ot (page 204)
set-ot-base (page 202)

Example
```
> (set-ot-base 1 7 2007)
3409014016
> (ot2ut -15071348)
3393942668
>
```

REPL Note
The equivalent of:

```
(full-date-and-time (ot2ut offset-universal-time)
   :include-seconds 't
   :destination *standard-output*)
```

can be invoked using the REPL command :ot *offset-universal-time*.

printvot *form** ⇒ *result** [*Macro*]

Purpose

Assist debugging involving Offset Universal Time values by printing forms and the results of evaluating them to `*trace-output*`. Any form producing a single-valued integer result is assumed to be an Offset Universal Time value and a convenient **full-date-and-time** format (including seconds) is printed as the result value.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

forms An implicit `progn` of forms to be evaluated and printed

results The values returned by evaluating the last *form* that is not the keyword symbol `:hr`

Returns

The values returned by evaluating the last *form* that is not the keyword symbol `:hr`

Description

The following is performed for each form in *forms*:

- if the form is the keyword symbol `:hr`, a dashed separator line printed to `*trace-output*`
- if the form is a string (before evaluation), it is treated as a label and printed to `*trace-output*` without enclosing double-quote characters
- if the form is a self-evaluating object, it is printed to `*trace-output*`
- otherwise, the form is printed to `*trace-output*`, then the form is evaluated and the result values are printed to `*trace-output*`. Unlike **printv**, a single-valued integer result is assumed to be an Offset Universal Time value and when that result is printed by **printvot**, it is first converted to Universal Time and then to **full-date-and-time** format (including seconds).

See also

full-date-and-time (page 172)
printv (page 108)

Examples

```
> (printvot "PRINTVOV example" *ot-base* (ut2ot) :hr)
;; PRINTVOV example
;;   *ot-base* => "Oct 22 2117 04:40:32"
;;   (ut2ot) => "Jun  4 2008 13:51:08"
;; ------------------------------------------------------------
-15071348
>
```

set-ot-base `&optional` *date month year time-zone* ⇒ *ot-base* [*Function*]

Purpose
Set the Offset Universal Time time-base value.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
date	An integer between 1 and up to 31, inclusive, depending on the month and year or `nil`, indicating the current date (default is `nil`)
month	An integer between 1 and 12, inclusive or `nil`, indicating the current month (default is `nil`)
year	An integer indicating the year A.D. or `nil`, indicating the current year (default is `nil`); if the *year* integer is between 0 and 99, the "obvious" year is assumed.
time-zone	A time zone: a rational multiple of 1/3600 between -24 and 24 that represents the number of hours offset from GMT (default is zero)

Returns
The Offset Universal Time time-base value

Description
Without any arguments, **set-ot-base** sets the time-base value to the current date. Setting the time base to the current date can be used in applications that do not need to represent historical dates and that do not save or communicate Offset Universal Time values. However, setting the time base to a specific date is recommended for most applications.

See also

ot-base (page 198)
check-ot-base (page 199)
ot2ut (page 200)
ut2ot (page 204)

Examples
Set the time base for Offset Universal Time to today:

```
> (set-ot-base)
3436662016
>
```

Note that the returned Offset Universal Time time-base value (above) is not the current Universal Time value:

```
> (get-universal-time)
3419947437
>
```

Set the time base for Offset Universal Time to July 1, 2007:

```
> (set-ot-base 1 7 2007)
3409014016
>
```

---

set-ot-base

ut2ot `&optional` *universal-time* ⇒ *offset-universal-time* [*Function*]

Purpose
Convert a Universal Time value to an Offset Universal Time value.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
universal-time A Universal Time value (default is the current Universal Time)
offset-universal-time An Offset Universal Time value

Returns
The equivalent Offset Universal Time value

Errors
The Offset Universal Time time-base value has not been set.

See also
ot-base (page 198)
check-ot-base (page 199)
ot2ut (page 200)
set-ot-base (page 202)

Examples
codeindexitmultiple-value-callcodeindexitencode-universal-time

```
> (set-ot-base 1 7 2007)
3409014016
> (ut2ot)
-15071348
> (ut2ot 3393942668)
-15071348
> (ut2ot (multiple-value-call #'encode-universal-time
           ;; noon on the Fourth of July, 2008:
           0 0 12 (parse-date "July 4, 2008")))
15161984
>
```

3.6 Transitioning Sets and Tables

Ideally, native Common Lisp hash tables operations should always be very fast. On some Common Lisp implementations, however, a small amount of time (and space) can be saved by using list-based representations for hash tables with small entry counts. These list-based representations transition automatically to regular hash tables as the entry count grows beyond the performance-advantage threshold (and back to the list representation as the count shrinks). These auto-transitioning representations add a small overhead to normal hash table operations, so their use should be considered very carefully (and restricted to situations where the counts tend to remain low).

ESETs

An ESET (short for `eq`-set) is a keys-only table that automatically transitions between list and hash-table implementations. The keys are used to represent elements of the ESET and are compared using `eq`.

ETs

An ET (short for `eq`-table) is a key-and-value table that automatically transitions between list and hash-table implementations. The keys are compared using `eq`.

---

add-to-eset *item eset* ⇒ *item* [*Function*]

Purpose

Add *item* to *eset* if it is not already present using `eq` as the comparison function.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

item An object
eset An ESET

Returns

The *item*

See also

in-eset (page 211)
make-eset (page 212)
delete-from-eset (page 207)

Examples

```
> (defparameter *eset* (make-eset))
*eset
> (in-eset 'x *eset*)
nil
nil
```

```
> (add-to-eset 'x *eset*)
x
> (in-eset 'x *eset*)
x
t
> (in-eset nil *eset*)
nil
nil
> (add-to-eset nil *eset*)
nil
> (in-eset nil *eset*)
nil
t
>
```

---

add-to-eset

delete-from-eset *item eset* ⇒ *deleted-p* [*Function*]

Purpose
Delete *item* from *eset* if it is present using `eq` as the comparison function.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
item An object
eset An ESET
deleted-p A generalized boolean

Returns
True, if *item* was deleted from *eset*; `nil` otherwise

See also
add-to-eset (page 205)
in-eset (page 211)
make-eset (page 212)

Examples
```
> (defparameter *eset* (make-eset))
*eset
> (add-to-eset 'x *eset*)
x
> (in-eset 'x *eset*)
x
t
> (delete-from-eset 'x *eset*)
t
> (in-eset 'x *eset*)
nil
nil
> (delete-from-eset 'x *eset*)
nil
>
```

delete-et *item et* ⇒ *deleted-p* [*Function*]

Purpose
Delete *item* from *et* if it is present using `eq` as the comparison function.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
item An object
et An ET
deleted-p A generalized boolean

Returns
True, if *item* was deleted from *et*; `nil` otherwise

See also
get-et (page 209)
make-et (page 213)

Examples
```
> (defparameter *et* (make-et))
*et
> (setf (get-et 'x *et*) 3)
3
> (get-et 'x *et*)
3
t
> (delete-et 'x *et*)
t
> (get-et 'x *et*)
nil
nil
> (delete-et 'x *et*)
nil
>
```

get-et *key et* `&optional` *default* ⇒ *value, present-p* [*Function*]

Purpose
Return the value associated with *key* in *et* using `eq` as the key-comparison function.

Setf syntax
(setf (get-et *key et* `&optional` *default*) *value*) ⇒ *value*

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
key An object
et An ET
value An object
present-p A generalized boolean

Returns
Two values:

- The value associated with the given *key* or *default* if no entry is associated with *key*
- True, if an entry was found; `nil` if there is no such entry

When `setf` is used with **get-et**, the supplied *value* is returned as a single value.

Description
`Setf` may be used with **get-et** to replace the value associated with the given *key*, or to add a new entry. When a **get-et** form is used as a setf place, any *default* which is supplied is evaluated, but its value is ignored.

See also
make-et (page 213)
assq (page 66)
delete-et (page 208)

Examples
```
> (defparameter *e* (make-et))
*et
> (get-et 'x *et*)
nil
nil
> (get-et 'x *et* ':missing)
:missing
nil
> (setf (get-et 'x *et*) 1)
1
> (get-et 'x *et*)
```

```
1
t
> (get-et nil *et*)
nil
nil
> (setf (get-et nil *et*) nil)
nil
> (get-et nil *et*)
nil
t
>
```

---

get-et

in-eset *item eset* ⇒ *item-or-nil, present-p* [*Function*]

Purpose
Determine if *item* is in *eset* using `eq` as the comparison function.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
item An object
eset An ESET
item-or-nil An object or `nil`
present-p A generalized boolean

Returns
Two values:

- *item*, if *item* is a member of *eset*; otherwise `nil`
- True, if an entry was found; `nil` if there is no such entry

See also
add-to-eset (page 205)
make-eset (page 212)
memq (page 102)
delete-from-eset (page 207)

Examples
```
> (defparameter *eset* (make-eset))
*eset
> (in-eset 'x *eset*)
nil
nil
> (add-to-eset 'x *eset*)
x
> (in-eset 'x *eset*)
x
t
> (in-eset nil *eset*)
nil
nil
> (add-to-eset nil *eset*)
nil
> (in-eset nil *eset*)
nil
t
>
```

make-eset <no arguments>⇒ *eset* [*Function*]

Purpose
Create an empty ESET.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
eset An ESET

Returns
An empty ESET

See also
add-to-eset (page 205)
in-eset (page 211)
delete-from-eset (page 207)

Examples
```
> (make-eset)
(0)
>
```

make-et <no arguments>⇒ *et* [*Function*]

Purpose
Create an empty ET.

Package
`:gbbopen-tools`

Module
`:gbbopen-tools`

Arguments
et An ET

Returns
An empty ET

See also
get-et (page 209)
delete-et (page 208)

Examples
```
> (make-et)
(0)
>
```

3.7 Search Trees

A red-black tree is binary search tree that is roughly balanced, keeping the worst-case time values for operations such as inserting, deleting, and searching proportional to the height of the tree.
Left-leaning red-black (LLRB) trees are a simpler version of red-black trees in which all red links must “lean” left except during inserts and deletes. LLRB trees have good worst-case running time for their operations and are efficient in general use.

llrb-tree-count *llrb-tree* ⇒ *count* [*Function*]

Purpose
Return number of entries stored in *llrb-tree*.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
llrb-tree An LLRB tree
count A non-negative integer

Returns
The number of entries stored in *llrb-tree*

See also
llrb-tree-delete (page 216)
llrb-tree-p (page 217)
llrb-tree-test (page 218)
llrb-tree-value (page 219)
make-llrb-tree (page 221)
map-llrb-tree (page 222)

Example
```
> (llrb-tree-count *tree*)
3
>
```

llrb-tree-delete *key llrb-tree* ⇒ *deleted-p* [*Function*]

Purpose
Delete the entry associated with a given key.

Package
`:gbbopen-tools`

Module
`:gbbopen-tools`

Arguments
key An object
llrb-tree An LLRB tree
deleted-p A generalized boolean

Returns
True if the entry was deleted; `nil` if it was not found

See also
llrb-tree-count (page 215)
llrb-tree-p (page 217)
llrb-tree-test (page 218)
llrb-tree-value (page 219)
make-llrb-tree (page 221)
map-llrb-tree (page 222)

Example
Delete the entry stored under `1` in the LLRB-tree `*tree*`, then try deleting it a second time:

```
> (llrb-tree-delete 1 *tree*)
t
> (llrb-tree-delete 1 *tree*)
nil
```

llrb-tree-p *object* ⇒ *boolean* [*Function*]

Purpose
Determine if an object is an LLRB tree.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
object An object
boolean A generalized boolean

Returns
True if *object* is an LLRB tree; `nil` otherwise.

See also
llrb-tree-count (page 215)
llrb-tree-delete (page 216)
llrb-tree-test (page 218)
llrb-tree-value (page 219)
make-llrb-tree (page 221)
map-llrb-tree (page 222)

Example
```
> (llrb-tree-p *tree*)
t
>
```

llrb-tree-test *llrb-tree* ⇒ *comparison-test* [*Function*]

Purpose
Return the three-way comparison test function used in *llrb-tree*.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
llrb-tree	An LLRB tree
comparison-test	A function designator specifying the three-way comparison test function object used in *llrb-tree*

Returns
The comparison test used in *llrb-tree*

See also
llrb-tree-count (page 215)
llrb-tree-delete (page 216)
llrb-tree-p (page 217)
llrb-tree-value (page 219)
make-llrb-tree (page 221)
map-llrb-tree (page 222)

Example
```
> (llrb-tree-test *tree*)
compare&
>
```

llrb-tree-value *key llrb-tree* `&optional` *default* ⇒ *value, present-p* [*Function*]

Purpose
Return the value associated with a given key.

Setf syntax
(setf (llrb-tree-value *key llrb-tree* `&optional` *default*) *value*) ⇒ *value* ⇒ *value, present-p*

Package
`:gbbopen-tools`

Module
`:gbbopen-tools`

Arguments
key	An object
llrb-tree	An LLRB tree
default	An object (default is `nil`)
value	An object
present-p	A generalized boolean

Returns
Two values:

- The value associated with the given *key* or *default* if no entry is associated with *key*
- True, if an entry was found; `nil` if there is no such entry

Description
`Setf` may be used with **llrb-tree-value** to replace the value associated with the given *key*, or to add a new entry. When a **llrb-tree-value** form is used as a setf place, any *default* which is supplied is evaluated, but its value is ignored.

See also
llrb-tree-count (page 215)
llrb-tree-delete (page 216)
llrb-tree-p (page 217)
llrb-tree-test (page 218)
make-llrb-tree (page 221)
map-llrb-tree (page 222)

Example
```
> (defparameter *tree* (make-llrb-tree #'compare&))
*tree*
> (llrb-tree-value 1 *tree*)
nil
> (setf (llrb-tree-value 1 *tree*) 'a)
a
> (llrb-tree-value 1 *tree*)
```

```
a
t
>
```

---

llrb-tree-value

make-llrb-tree *comparison-test* ⇒ *llrb-tree* [*Function*]

Purpose
Create an empty left-leaning red-black tree.

Package
`:gbbopen-tools`

Module
`:gbbopen-tools`

Arguments
comparison-test A three-way comparison test
llrb-tree An LLRB tree

Returns
An empty left-leaning red-black tree

See also
llrb-tree-count (page 215)
llrb-tree-delete (page 216)
llrb-tree-p (page 217)
llrb-tree-test (page 218)
llrb-tree-value (page 219)
map-llrb-tree (page 222)

Example
```
> (make-llrb-tree #'compare&)
#<compare& llrb-tree with 0 entries>
>
```

map-llrb-tree *function llrb-tree* [*Function*]

Purpose

Apply a function once to the key and value of each entry in an LLRB tree.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

function	A function designator specifying a function object of two arguments
llrb-tree	An LLRB tree

See also

llrb-tree-count (page 215)
llrb-tree-delete (page 216)
llrb-tree-p (page 217)
llrb-tree-test (page 218)
llrb-tree-value (page 219)
make-llrb-tree (page 221)

Example

```
> (map-llrb-tree
    #'(lambda (key value)
        (printv key value))
  *tree*)
;;  key => 1
;;  value => a
;;  key => 2
;;  value => b
;;  key => 3
;;  value => c
nil
>
```

4 Additional GBBopen Tools

Additional GBBopen Tool entities are grouped into modules that can be loaded as appropriate. Documentation for these additional-tool entities is arranged according to the following modules:

- uniform interfaces to implementation-specific thread (multiprocessing) capabilities are provided by the `:portable-threads` module (see page 224)
- polling functions for Common Lisp implementations that do not provide thread capabilities are provided by the `:polling-functions` module (see page 298)
- uniform socket interfaces are provided by the `:portable-sockets` module (see page 304)
- uniform interfaces to the operating system are provided by the `:os-interface` module (see page 317)
- Double Metaphone phonetic-code computation is provided by the `:double-metaphone` module (see page 315) module.

4.1 Portable Threads

GBBopen's Portable Threads provides a uniform interface to commonly used thread (multiprocessing) entities. Wherever possible, these entities do something reasonable in Common Lisp implementations that do not provide threads. However, entities that make no sense without threads signal errors in non-threaded implementations (as noted with each entity). The feature `:threads-not-available` is added on Common Lisp implementations without thread support, and the feature `:with-timeout-not-available` is added on implementations that do not support **with-timeout**.

Portable Threads entities are provided by the `:portable-threads` module in GBBopen. Stand-alone use of the Portable Threads interface is also easy, requiring only the `portable-threads.lisp` file for the portable-interface layer and, if desired, `scheduled-periodic-functions.lisp` for the scheduled and periodic function entities (see page 268).

Threads and Processes

Common Lisp implementations that provide multiprocessing capabilities use one of two approaches:

- *Application-level threads* (also called "Lisp processes") which are created, deleted, and scheduled internally by the Common Lisp implementation
- *Operating-system threads* (or "native threads") which are lightweight, operating-system threads that are created, deleted, and scheduled by the operating system

There are advantages and complexities associated with each approach, and the Portable Threads Interface is designed to provide a uniform abstraction over them that can be used to code applications that perform consistently and efficiently on any supported Common Lisp implementation.

Locks

Common Lisp implementations provide differing semantics for the behavior of mutual-exclusion locks that are acquired recursively by the same thread: some always allow recursive use, others provide special "recursive" lock objects in addition to non-recursive locks, and still others allow recursive use to be specified at the time that a lock is being acquired. To enable behavioral consistency in all Common Lisp implementations, the `:portable-threads` interface module provides (non-recursive) locks and recursive locks and a single acquisition form, **with-lock-held**, that behaves appropriately for each lock type.

Condition Variables

POSIX-style condition variables provide an atomic means for a thread to release a lock that it holds and go to sleep until it is awakened by another thread. Once awakened, the lock that it was holding is reacquired atomically before the thread is allowed to do anything else.

A condition variable must always be associated with a lock (or recursive lock) in order to avoid a race condition created when one thread signals a condition while another thread is preparing to wait on it. In this situation, the second thread would be perpetually waiting for the signal that has already been sent. In the POSIX model, there is no explicit link between the lock used to control access to the condition variable and the condition variable. The Portable Threads Interface makes this association explicit by bundling the lock with the **condition-variable** CLOS object instance and allowing the **condition-variable** object to be used directly in lock entities.

Hibernation

Sometimes it is desirable to put a thread to sleep (perhaps for a long time) until some event has occurred. The Portable Threads Interface provides two entities that make this situation easy to code: **hibernate-thread** and **awaken-thread**. Thread hibernation can only be performed by the thread on itself, eliminating issues of a thread being hibernated at an undesirable time. Note that there is the potential for a hibernate/awaken race condition if a thread hibernates itself again soon after being awakened (when a second **awaken-thread** intended for the original hiberation is applied to the second hibernation rather than being ignored because the target thread is not hibernating). Using a **condition-variable** is preferable in this situation.

When a thread is hibernating, it remains available to respond to **run-in-thread** and **symbol-value-in-thread** operations as well as to be awakened by a dynamically surrounding **with-timeout**.

What about Process Wait?

Thread coordination functions, such as `process-wait`, are expensive to implement with operating-system threads. Such functions stop the executing thread until a Common Lisp predicate function returns a true value. With application-level threads, the Lisp-based scheduler evaluates the predicate function periodically when looking for other threads that can be run. With operating-system threads, however, thread scheduling is performed by the operating system and evaluating a Common Lisp predicate function requires complex and expensive interaction between the operating-system thread scheduling and the Common Lisp implementation. Given this cost and complexity, many Common Lisp implementations that use operating-system threads have elected not to provide `process-wait`-style coordination functions, and this issue extends to the Portable Threads Interface as well.

Fortunately, most uses of `process-wait` can be replaced by a different strategy that relies on the producer of a change that would affect the `process-wait` predicate function to signal the event rather than having the consumers of the change use predicate functions to poll for it. Condition variables, the Portable Threads **hibernate-thread** and **awaken-thread** mechanism, or blocking I/O functions cover most of the typical uses of `process-wait`.

Portable Threads entities

Descriptions of the Portable Threads entities follow.

all-threads <no arguments> ⇒ *list-of-threads* [*Function*]

Purpose
Return a list of all known threads.

Package
`:portable-threads`

Module
`:portable-threads`

Arguments
list-of-threads A proper list

Returns
A list of objects representing the threads.

Description
The returned list of threads is accurate only at the precise instant the **all-threads** function is called. New threads may be created or existing threads killed at any time, so the returned list is always potentially outdated.

See also
current-thread (page 244)
thread-alive-p (page 257)
threadp (page 261)
spawn-form (page 254)
spawn-thread (page 255)

Example
```
> (all-threads)
(#<thread Listener 1>)
>
```

Notes
On Common Lisp implementations without threads, `nil` is returned.

The returned list of threads should not be destructively altered.

as-atomic-operation *form** ⇒ *primary-value* [*Macro*]

Purpose
Execute *forms* as an atomic operation.

Package :portable-threads

Module :portable-threads

Arguments
primary-value The first value returned by evaluating the last *form*

Returns
The primary value returned by evaluating the last *form*.

Description
This macro provides atomicity in the following entities (when the Common Lisp implementation does not support them directly): **atomic-decf**, **atomic-decf&**, **atomic-delete**, **atomic-flush**, **atomic-incf**, **atomic-incf&**, **atomic-push**, **atomic-pushnew**, and **atomic-pop**. It is intended only for implementing very brief atomic operations and should not be used for long computations or computations that wait or block.

Note that **as-atomic-operation** is only guaranteed to return a single value, not multiple values.

See also

atomic-decf (page 228)
atomic-decf& (page 229)
atomic-delete (page 230)
atomic-flush (page 232)
atomic-incf (page 233)
atomic-incf& (page 234)
atomic-push (page 236)
atomic-pushnew (page 237)
atomic-pop (page 235)

Example
Define an atomic **nsorted-insert**:

```
(defun atomic-nsorted-insert (&rest args)
  (declare (dynamic-extent args))
  (as-atomic-operation (apply #'nsorted-insert args)))
```

atomic-decf *place* [*delta-form*] ⇒ *new-place-value* [*Macro*]

Purpose
Decrement the value stored in *place* as an atomic operation.

Package :portable-threads

Module :portable-threads

Arguments
place A form which is suitable for use as a generalized reference
delta-form A form that is evaluated to produce a delta value (default is 1)
new-place-value A number

Returns
The new value of *place*.

See also
as-atomic-operation (page 227)
atomic-decf& (page 229)
atomic-delete (page 230)
atomic-flush (page 232)
atomic-incf (page 233)
atomic-incf& (page 234)
atomic-pop (page 235)
atomic-pushnew (page 237)

Examples
```
> x
5
> (atomic-decf x)
4
> (atomic-decf x 1.5)
2.5
>
```

atomic-decf& *place* [*delta-form*] ⇒ *new-place-value* [*Macro*]

Purpose
Decrement the fixnum value stored in *place* as an atomic operation.

Package :portable-threads

Module :portable-threads

Arguments
place A form which is suitable for use as a generalized reference containing a fixnum value
delta-form A form that is evaluated to produce a fixnum delta value (default is 1)
new-place-value A fixnum

Returns
The new fixnum value of *place*.

See also
as-atomic-operation (page 227)
atomic-decf (page 228)
atomic-delete (page 230)
atomic-flush (page 232)
atomic-incf (page 233)
atomic-incf& (page 234)
atomic-pop (page 235)
atomic-pushnew (page 237)

Examples
```
> x
5
> (atomic-decf& x)
4
> (atomic-decf& x 2)
2
>
```

atomic-delete *item place* `&key` *from-end test test-not start end count key* ⇒ *sequence* [*Macro*]

Purpose

As an atomic operation, set *place* to the sequence in *place* from which the elements that satisfy the *test* have been removed.

Package `:portable-threads`

Module `:portable-threads`

Arguments

item	An object
place	A form which is suitable for use as a generalized reference that contains a proper sequence
from-end	A generalized boolean (default is `nil`)
test	A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'eql`)
test-not	A function designator specifying a function object of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)
start	Starting index into *sequence* (default is `0`)
end	Ending index into *sequence* (default is `nil`, meaning end of *sequence*)
count	An integer or `nil` (default is `nil`)
key	A function designator specifying a function object of one argument, or `nil` (default is `nil`)
sequence	A sequence

Returns

The sequence in *place* from which the elements that satisfy the test have been removed.

Description

Replaces *place* with the sequence in *place* from which elements that satisfy the *test* have been deleted. The supplied *place* sequence may be modified in constructing the result; however, modification of the sequence itself is not guaranteed.

Specifying a *from-end* value of true matters only when the *count* is provided, and in that case only the rightmost *count* elements satisfying the *test* are deleted.

See also

as-atomic-operation (page 227)
atomic-flush (page 232)
atomic-pop (page 235)
atomic-push (page 236)
atomic-pushnew (page 237)
counted-delete (page 75)
delq (page 80)
delq-one (page 81)

Example

```
> list
(1 2 3)
> (atomic-delete 2 list)
(2 3)
> list
(2 3)
>
```

---

atomic-delete

atomic-flush *place* ⇒ *old-place-value* [*Macro*]

Purpose

As an atomic operation, set the value of *place* to `nil`, and return the value *place* had prior to being set.

Package `:portable-threads`

Module `:portable-threads`

Arguments

place	A form which is suitable for use as a generalized reference
old-place-value	An object

Returns

The *place* value prior to being set to `nil`.

See also

as-atomic-operation (page 227)
atomic-delete (page 230)
atomic-pop (page 235)
atomic-push (page 236)
atomic-pushnew (page 237)

Example

```
> list
(1 2 3)
> (atomic-flush list)
(1 2 3)
> list
nil
>
```

atomic-incf *place* [*delta-form*] ⇒ *new-place-value* [*Macro*]

Purpose
Increment the value stored in *place* as an atomic operation.

Package :portable-threads

Module :portable-threads

Arguments
place	A form which is suitable for use as a generalized reference
delta-form	A form that is evaluated to produce a delta value (default is 1)
new-place-value	A number

Returns
The new value of *place*.

See also

as-atomic-operation (page 227)
atomic-decf (page 228)
atomic-decf& (page 229)
atomic-delete (page 230)
atomic-flush (page 232)
atomic-incf& (page 234)
atomic-pop (page 235)
atomic-pushnew (page 237)

Examples
```
> x
2
> (atomic-incf x)
3
> (atomic-incf x 1.5)
4.5
>
```

atomic-incf& *place* [*delta-form*] ⇒ *new-place-value* [*Macro*]

Purpose
Increment the fixnum value stored in *place* as an atomic operation.

Package `:portable-threads`

Module `:portable-threads`

Arguments
place	A form which is suitable for use as a generalized reference containing a fixnum value
delta-form	A form that is evaluated to produce a fixnum delta value (default is 1)
new-place-value	A fixnum

Returns
The new fixnum value of *place*.

See also

as-atomic-operation (page 227)
atomic-decf (page 228)
atomic-decf& (page 229)
atomic-delete (page 230)
atomic-flush (page 232)
atomic-incf (page 233)
atomic-pop (page 235)
atomic-pushnew (page 237)

Examples
```
> x
2
> (atomic-incf& x)
3
> (atomic-incf& x 2)
5
>
```

atomic-pop *place* ⇒ *element* [*Macro*]

Purpose

As an atomic operation, remove the first element from the list stored in *place*, store the updated list in *place*, and return the removed first element.

Package :portable-threads

Module :portable-threads

Arguments

place A form which is suitable for use as a generalized reference that contains a proper list or a dotted list

element An object

Returns

The first element (the car) of the list stored in *place*.

See also

as-atomic-operation (page 227)
atomic-delete (page 230)
atomic-flush (page 232)
atomic-push (page 236)
atomic-pushnew (page 237)

Example

```
> list
(1 2 3)
> (atomic-pop list)
1
> list
(2 3)
>
```

atomic-push *item place* ⇒ *new-place-value* [*Macro*]

Purpose

As an atomic operation, prepend *item* to the list stored in *place* and store the updated list in *place*.

Package `:portable-threads`

Module `:portable-threads`

Arguments

item	An object
place	A form which is suitable for use as a generalized reference
new-place-value	A proper list

Returns

The new value of *place*.

See also

as-atomic-operation (page 227)
atomic-delete (page 230)
atomic-flush (page 232)
atomic-pop (page 235)
atomic-pushnew (page 237)

Example

```
> list
(1 2 3)
> (atomic-push 10 list)
(10 1 2 3)
>
```

atomic-pushnew *item place* `&key` *key test test-not* ⇒ *new-place-value* [*Macro*]

Purpose

As an atomic operation, when *item* is not the same as any element in the list stored in *place*, prepend *item* to the list and store the updated list in *place*.

Package `:portable-threads`

Module `:portable-threads`

Arguments

item	An object
place	A form which is suitable for use as a generalized reference that contains a proper list
key	A function designator specifying a function object of one argument, or `nil` (default is `nil`)
test	A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'eql`)
test-not	A function designator specifying a function object of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)
new-place-value	A proper list

Returns

The new value of *place*.

See also

as-atomic-operation (page 227)
atomic-delete (page 230)
atomic-flush (page 232)
atomic-pop (page 235)
atomic-push (page 236)

Examples

```
> list
(1 2 3)
> (atomic-pushnew 2 list)
(1 2 3)
> (atomic-pushnew 10 list)
(10 1 2 3)
>
```

awaken-thread *thread* [*Function*]

Purpose
Awaken a hibernating thread.

Package
`:portable-threads`

Module
`:portable-threads`

Arguments
thread A thread

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Description
An attempt to awaken a non-hibernating thread is ignored.

See also
hibernate-thread (page 245)

Example
```
(awaken-thread thread)
```

condition-variable [*Class*]

Package

`:portable-threads`

Module

`:portable-threads`

Description

The class **condition-variable** is a subclass of `standard-object`. Instances of **condition-variable** include an associated lock, which can be either a lock (the default) or a recursive lock.

See also

make-condition-variable (page 247)
define-class (page 83)

condition-variable-broadcast *condition-variable* [*Function*]

Purpose
Unblock all threads that are blocked on *condition-variable*.

Package `:portable-threads`

Module `:portable-threads`

Arguments
condition-variable A condition variable

Errors
The lock (or recursive lock) associated with *condition-variable* is not held by the executing process.

Description
If no threads are blocked on *condition-variable*, this function is a no-op.

See also

condition-variable-signal (page 241)
condition-variable-wait (page 242)
condition-variable-wait-with-timeout (page 243)
make-condition-variable (page 247)
with-lock-held (page 263)
without-lock-held (page 267)

Example
Acquire the lock associated with `condition-variable` and then signal all blocked threads that are waiting on it:

```
(with-lock-held (condition-variable)
  (condition-variable-broadcast condition-variable))
```

Note
On Common Lisp implementations without threads, this function does nothing.

condition-variable-signal *condition-variable* [*Function*]

Purpose
Unblock one thread that is blocked on *condition-variable*.

Package `:portable-threads`

Module `:portable-threads`

Arguments
condition-variable A condition variable

Errors
The lock (or recursive lock) associated with *condition-variable* is not held by the executing process.

Description
If no threads are blocked on *condition-variable*, this function is a no-op.

See also

condition-variable-broadcast (page 240)
condition-variable-wait (page 242)
condition-variable-wait-with-timeout (page 243)
make-condition-variable (page 247)
with-lock-held (page 263)
without-lock-held (page 267)

Example
Acquire the lock associated with `condition-variable` and then signal one blocked thread that is waiting on it:

```
(with-lock-held (condition-variable)
  (condition-variable-signal condition-variable))
```

Note
On Common Lisp implementations without threads, this function does nothing.

condition-variable-wait *condition-variable* [*Function*]

Purpose
Block the current thread on *condition-variable*.

Package
`:portable-threads`

Module
`:portable-threads`

Arguments
condition-variable A condition variable

Errors
The lock (or recursive lock) associated with *condition-variable* is not held by the executing process.

Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also

condition-variable-broadcast (page 240)
condition-variable-signal (page 241)
condition-variable-wait-with-timeout (page 243)
make-condition-variable (page 247)
with-lock-held (page 263)
without-lock-held (page 267)

Example
Acquire the condition-variable lock and then wait until signaled by another thread:

```
(with-lock-held (condition-variable)
  (condition-variable-wait condition-variable))
```

condition-variable-wait-with-timeout *condition-variable seconds* ⇒ *boolean* [*Function*]

Purpose
Block the current thread on *condition-variable* or until *seconds* seconds have elapsed.

Package `:portable-threads`

Module `:portable-threads`

Arguments
condition-variable	A condition variable
seconds	A number
boolean	A generalized boolean

Returns
True if *condition-variable* is unblocked before *seconds* seconds have elapsed; `nil` if the timeout has occurred.

Errors
The lock (or recursive lock) associated with *condition-variable* is not held by the executing process.

Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also

condition-variable-broadcast (page 240)
condition-variable-signal (page 241)
condition-variable-wait (page 242)
make-condition-variable (page 247)
with-lock-held (page 263)
without-lock-held (page 267)

Example
Acquire the condition-variable lock and then wait until signaled by another thread or until 5 seconds have elapsed:

```
(with-lock-held (condition-variable)
  (condition-variable-wait-with-timeout condition-variable 5))
```

current-thread <no arguments> ⇒ *thread* [*Function*]

Purpose
Return the object representing the current thread.

Package `:portable-threads`

Module `:portable-threads`

Arguments
thread A thread

Returns
The object representing the current thread.

See also
all-threads (page 226)
spawn-form (page 254)
spawn-thread (page 255)

Example
```
> (current-thread)
#<thread Listener 1>
>
```

Note
On Common Lisp implementations without threads, the keyword symbol `:threads-not-available` is returned.

hibernate-thread <no arguments> [*Function*]

Purpose
Hibernate the current thread.

Package
`:portable-threads`

Module
`:portable-threads`

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
awaken-thread (page 238)

Example
Hibernate the current thread:

```
(hibernate-thread)
```

kill-thread *thread* [*Function*]

Purpose
Kill a thread.

Package :portable-threads

Module :portable-threads

Arguments
thread A thread

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
spawn-form (page 254)
spawn-thread (page 255)
thread-alive-p (page 257)

Example
```
(kill-thread thread)
```

make-condition-variable `&rest` *initargs* `&key` *class lock* ⇒ *condition-variable* [*Function*]

Purpose
Create a new condition variable.

Package `:portable-threads`

Module `:portable-threads`

Arguments
initargs	An initialization argument list
class	The name of the class for the created condition-variable instance (default is `condition-variable`)
lock	A lock, a recursive lock, or a condition variable (default is a non-recursive lock)
condition-variable	A condition variable

Returns
The created **condition-variable**.

See also

make-instance (page 364)
make-lock (page 249)
make-recursive-lock (page 250)
with-lock-held (page 263)
without-lock-held (page 267)

Examples
Make a **condition-variable** instance with a non-recursive lock:
```
> (make-condition-variable)
#<condition-variable>
>
```

Make a **condition-variable** instance with a recursive lock:
```
> (make-condition-variable :lock (make-recursive-lock))
#<condition-variable>
>
```

Define a subclass of **condition-variable** that includes a `state` slot:
```
(defclass state-cv (condition-variable)
  ((state :initarg :state
          :initform nil
          :accessor state-of)))
```

and then create a `state-cv` instance with a recursive lock:
```
> (make-condition-variable :class 'state-cv
                           :lock (make-recursive-lock))
#<state-cv>
>
```

Note

The **make-condition-variable** function is equivalent to using **make-instance** with the desired class for the created condition-variable instance. However, using **make-condition-variable** is preferable stylistically.

---

make-condition-variable

make-lock `&key` *name* ⇒ *lock* [*Function*]

Purpose
Create a lock.

Package `:portable-threads`

Module `:portable-threads`

Arguments
name A string.
lock A lock

Returns
The newly created lock.

See also
make-condition-variable (page 247)
make-recursive-lock (page 250)
thread-holds-lock-p (page 262)
with-lock-held (page 263)
without-lock-held (page 267)

Example
```
> (make-lock :name "Priority Queue")
#<lock Priority Queue>
>
```

Note
On Common Lisp implementations without threads, a “pseudo-lock” object is returned.

make-recursive-lock `&key` *name* ⇒ *recursive-lock* [*Function*]

Purpose
Create a recursive lock.

Package `:portable-threads`

Module `:portable-threads`

Arguments
name A string.
lock A recursive lock

Returns
The newly created recursive lock.

See also

make-condition-variable (page 247)
make-lock (page 249)
thread-holds-lock-p (page 262)
with-lock-held (page 263)
without-lock-held (page 267)

Example
```
> (make-recursive-lock :name "Priority Queue")
#<recursive-lock Priority Queue>
>
```

Note
On Common Lisp implementations without threads, a “pseudo-recursive-lock” object is returned.

nearly-forever-seconds [*Constant*]

Purpose

The maximum number of seconds supported by `sleep`.

Package

`:portable-threads`

Module

`:portable-threads`

Value type

A fixnum

Value

Implementation dependent

See also

sleep-nearly-forever (page 253)

run-in-thread *thread function* `&rest` *args* [*Function*]

Purpose
Force *thread* to apply *function* to *args*.

Package `:portable-threads`

Module `:portable-threads`

Arguments
thread A thread
function A function designator
args Arguments to the function

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
spawn-form (page 254)
spawn-thread (page 255)

Example
```
(run-in-thread thread
               #'(lambda (result) (throw ':exit result))
               result)
```

sleep-nearly-forever `&optional` *seconds* [*Function*]

Purpose
A maximum-time-bounded `sleep`.

Package `:portable-threads`

Module `:portable-threads`

Arguments
seconds An integer (default is **nearly-forever-seconds**)

Description
Calls `sleep` with *seconds* or **nearly-forever-seconds**, whichever is less. Using **nearly-forever-seconds** protects against exceeding the duration limit of the Common Lisp implementation's `sleep` function for very long duration sleeping, by truncating the duration to **nearly-forever-seconds**.

See also
nearly-forever-seconds (page 251)

Examples
```
> (sleep-nearly-forever)    ; sleep for a very long time
   ...
> (sleep-nearly-forever     ; sleep as long as the above
     (* 2 nearly-forever-seconds))
   ...
>
```

spawn-form *form**⇒ *thread* [*Macro*]

Purpose
Evaluate *forms* in a new thread.

Package :portable-threads

Module :portable-threads

Arguments
form A form
thread A thread

Returns
The object representing the new thread.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also

all-threads (page 226)
awaken-thread (page 238)
current-thread (page 244)
hibernate-thread (page 245)
kill-thread (page 246)
spawn-thread (page 255)
thread-alive-p (page 257)
thread-name (page 258)
thread-whostate (page 259)
threadp (page 261)
run-in-thread (page 252)
symbol-value-in-thread (page 256)

Example
```
> (spawn-form (sleep 60))
#<thread Form (sleep 60)>
>
```

spawn-thread *name function* `&rest` *args* ⇒ *thread* [*Function*]

Purpose
Spawn a new thread.

Package `:portable-threads`

Module `:portable-threads`

Arguments
name A string
function A function designator
args Arguments to the function
thread A thread

Returns
The object representing the new thread.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
all-threads (page 226)
awaken-thread (page 238)
current-thread (page 244)
hibernate-thread (page 245)
kill-thread (page 246)
spawn-form (page 254)
thread-alive-p (page 257)
thread-name (page 258)
thread-whostate (page 259)
threadp (page 261)
run-in-thread (page 252)
symbol-value-in-thread (page 256)

Example
```
> (spawn-thread "Sleepy" #'sleep 60)
#<thread Sleepy>
>
```

symbol-value-in-thread *symbol thread* ⇒ *object, boundp* [*Function*]

Purpose
Return the value of *symbol* in a thread.

Package
`:portable-threads`

Module
`:portable-threads`

Arguments
symbol A symbol
thread A thread
object An object
boundp A generalized boolean

Returns
Two values:

- the value of *symbol* in *thread* or nil if no value is bound
- `t` if *symbol* is specially or globally bound in *thread*; otherwise `nil`

Description
The global symbol value is returned as the first value if no thread-local value is bound.

See also
spawn-form (page 254)
spawn-thread (page 255)

Examples
```
> (symbol-value-in-thread '*x* thread)
33
t
> (symbol-value-in-thread 'pi thread)
3.141592653589793d0
t
> (symbol-value-in-thread '*unbound* thread)
nil
nil
>
```

Note
On Common Lisp implementations without threads, this function obtains the global symbol value.

thread-alive-p *thread* ⇒ *boolean* [*Function*]

Purpose
Determine if a thread is alive.

Package :portable-threads

Module :portable-threads

Arguments
thread A thread
boolean A generalized boolean

Returns
True if *thread* is alive; `nil` otherwise.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
all-threads (page 226)
kill-thread (page 246)
spawn-form (page 254)
spawn-thread (page 255)

Examples
```
> (defparameter *silly-thread* (spawn-thread "Sleeper" 'sleep 10000))
*silly-thread*
> (thread-alive-p *silly-thread*)
t
> (kill-thread *silly-thread*)
t
> (thread-alive-p *silly-thread*)
nil
>
```

thread-name *thread* ⇒ *name-string* [*Function*]

Purpose
Return the name of a thread.

Setf syntax
(setf (thread-name *thread*) *name-string*) ⇒ *name-string*

Package :portable-threads

Module :portable-threads

Arguments
thread A thread
name-string A string

Returns
The name of *thread*.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
spawn-form (page 254)
spawn-thread (page 255)

Examples
```
> (thread-name thread)
"Initial"
> (setf (thread-name thread) "Version 2")
"Version 2"
> (thread-name thread)
"Version 2"
>
```

Note
Digitool's Macintosh Common Lisp does not support changing the thread name via **setf**.

thread-whostate *thread* ⇒ *whostate* [*Function*]

Purpose
Return a string that describes the current state of a thread.

Setf syntax
(setf (thread-whostate *thread*) *whostate*) ⇒ *whostate*

Package :portable-threads

Module :portable-threads

Arguments
thread A thread
whostate A string or `nil`

Returns
The whostate string of the thread or `nil`.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
spawn-form (page 254)
spawn-thread (page 255)
with-lock-held (page 263)
without-lock-held (page 267)

Example
```
> (thread-whostate thread)
"Running"
>
```

Note
Although the *whostate* value can provide helpful information when debugging, specific *whostate* values and their meanings vary among Common Lisp implementations and should not be used programmatically.

Only Allegro CL, Clozure CL, and Digitool's Macintosh Common Lisp support user-settable whostates; **(setf whostate)** is a no-op on other Common Lisp implementations.

thread-yield <no arguments> [*Function*]

Purpose

Give other threads a chance to execute.

Package

`:portable-threads`

Module

`:portable-threads`

Example

```
(thread-yield)
```

Note

On Common Lisp implementations without thread support, this function executes **run-polling-functions** if the `:polling-functions` module has been loaded. Otherwise, it is a no-op on non-threaded implementations.

threadp *object* ⇒ *boolean* [*Function*]

Purpose
Check if *object* is an object representing a thread.

Package `:portable-threads`

Module `:portable-threads`

Arguments
object An object
boolean A generalized boolean

Returns
True if *object* is an object representing a thread; `nil` otherwise.

See also
all-threads (page 226)
spawn-form (page 254)
spawn-thread (page 255)
thread-alive-p (page 257)

Example
```
> (threadp (car (all-threads)))
t
>
```

thread-holds-lock-p *lock* ⇒ *boolean* [*Function*]

Purpose
Determine if *lock* is held by the current thread.

Package
`:portable-threads`

Module
`:portable-threads`

Arguments
lock A lock, a recursive lock, or a condition variable
boolean A generalized boolean

Returns
True if the current thread holds *lock*; `nil` otherwise.

See also
make-condition-variable (page 247)
make-lock (page 249)
make-recursive-lock (page 250)
with-lock-held (page 263)
without-lock-held (page 267)

Examples
Two simple examples using a lock:

```
> (thread-holds-lock-p lock)
nil
> (with-lock-held (lock)
    (thread-holds-lock-p lock))
t
>
```

Two more simple examples using a condition variable:

```
> (thread-holds-lock-p condition-variable)
nil
> (with-lock-held (condition-variable)
    (thread-holds-lock-p condition-variable))
t
>
```

with-lock-held (*lock* `&key` *whostate*) *form** $\Rightarrow$ *result** [*Macro*]

Purpose
After acquiring a lock or a recursive lock, execute forms and then release the lock.

Package `:portable-threads`

Module `:portable-threads`

Arguments
lock A lock, a recursive lock, or a condition variable
whostate A string (default `"With Lock Held"`)
forms An implicit **progn** of forms to be evaluated
results The values returned by evaluating the last *form*

Returns
The values returned by evaluating the last *form*.

Errors
A thread attempts to re-acquire a (non-recursive) lock that it holds.

Description
If a thread executes a **with-lock-held** that is dynamically inside another **with-lock-held** involving the same recursive lock, the inner **with-lock-held** simply proceeds as if it had acquired the lock.

See also
make-condition-variable (page 247)
make-lock (page 249)
make-recursive-lock (page 250)
thread-holds-lock-p (page 262)
thread-whostate (page 259)
without-lock-held (page 267)

Examples
Acquire the lock controlling access to a critical section of code:

```
(with-lock-held (lock :whostate "Waiting for Critical Lock")
  (critical-section))
```

A silly example showing a recursive re-acquisition of a recursive lock:

```
(with-lock-held (recursive-lock :whostate "Waiting for Critical Lock")
  (with-lock-held (recursive-lock :whostate "Again Waiting for Critical
Lock")
    (critical-section)))
```

Acquire the lock associated with `condition-variable` and then signal all blocked threads that are waiting on it:

```
(with-lock-held (condition-variable)
  (condition-variable-signal condition-variable))
```

Note

The *whostate* value is ignored by SBCL.

---

with-lock-held

with-timeout (*seconds timeout-form**) *form** ⇒ *result** [*Macro*]

Purpose

Bound the time allowed to evaluate *forms* to *seconds*, evaluating *timeout-forms* if the time limit is reached.

Package

`:portable-threads`

Module

`:portable-threads`

Arguments

seconds A number

timeout-forms An implicit **progn** of forms to be evaluated if the timed *forms* do not complete before *seconds* seconds have elapsed

forms An implicit **progn** of forms to be evaluated

results The values returned by evaluating the last *form* or the last *timeout-form*

Returns

The values returned by evaluating the last *form* if completed in less than *seconds* seconds; otherwise the values returned by evaluating the last *timeout-form*

Errors

Threads (multiprocessing) is not supported on the Common Lisp implementation. However, **with-timeout** is also supported on non-threaded SBCL.

Description

If the evaluation of *forms* does not complete within *seconds* seconds, execution of *forms* is terminated and the *timeout-forms* are evaluated, returning the result of the last *timeout-form*. The *timeout-forms* are not evaluated if the *forms* complete within *seconds* seconds, in which case the result of the last *form* is returned.

See also

condition-variable-wait-with-timeout (page 243)

Examples

Evaluate a simple form, with a one-second time out:

```
> (with-timeout (1 ':timed-out)
    ':did-not-time-out)
:did-not-time-out
>
```

Again, but this time sleep for two seconds to cause a time out:

```
> (with-timeout (1 ':timed-out)
    (sleep 2)
    ':did-not-time-out)
:timed-out                ; (after 1 second)
>
```

Uses of **with-timeout** can be nested:

```
> (with-timeout (1 ':timed-out-outer)
      (with-timeout (2 ':timed-out-inner)
         (sleep 3)
         ':did-not-time-out))
:timed-out-outer          ; (after 1 second)
> (with-timeout (2 ':timed-out-outer)
      (with-timeout (1 ':timed-out-inner)
         (sleep 3)
         ':did-not-time-out))
:timed-out-inner          ; (after 1 second)
>
```

---

with-timeout

without-lock-held (*lock* `&key` *whostate*) *form** ⇒ *result** [*Macro*]

Purpose
Temporarily release a lock or a recursive lock, execute forms and then reacquire the lock.

Package `:portable-threads`

Module `:portable-threads`

Arguments
lock A lock, a recursive lock, or a condition variable
whostate A string (default `"Without Lock Held"`)
forms An implicit **progn** of forms to be evaluated
results The values returned by evaluating the last *form*

Returns
The values returned by evaluating the last *form*.

Errors
A thread attempts to release a lock that it does not hold.

See also
make-condition-variable (page 247)
make-lock (page 249)
make-recursive-lock (page 250)
thread-holds-lock-p (page 262)
thread-whostate (page 259)
with-lock-held (page 263)

Example
Acquire and temporarily release a lock controlling access to several critical sections of code:

```
(with-lock-held (lock :whostate "Waiting for Critical Lock")
  (critical-section-1)
  (without-lock-held (lock :whostate "Doing non-critical stuff")
    (non-critical-section))
  (critical-section-2))
```

Note
The *whostate* value is ignored by SBCL.

4.2 Scheduled and Periodic Functions

GBBopen provides two forms of time-scheduled functions: scheduled functions and periodic functions, both built using Portable Threads (see page 224). These time-scheduled-function entities are provided by the `:portable-threads` module.

Scheduled Functions

A scheduled function is an object that contains a function to be run at a specified time. When that specified time arrives, the function is invoked with a single argument: the scheduled function object. A repeat interval (in seconds) can also be specified for the scheduled function. This value is used whenever the scheduled function is invoked to schedule itself again at a new time relative to the current invocation. Scheduled functions can be scheduled to a resolution of one second.

Scheduled functions are scheduled and invoked by a separate `"Scheduled-Function Scheduler"` thread. Unless the run time of the invoked function is brief, the invoked function should spawn a new thread in which to perform its activities so as to avoid delaying the invocation of a subsequent scheduled function.

Periodic Functions

A periodic function is a function to be run repeatedly at a specified interval. Unlike scheduled functions, which can be scheduled only to a resolution of one second, a periodic function can be repeated at intervals as brief as is supported by the `sleep` function of the Common Lisp implementation. A periodic function is scheduled and executed in its own thread. As with scheduled functions, the invoked function should spawn a new thread in which to perform its activities, unless its run time is brief.

Portable Threads entities

Descriptions of the Portable Threads entities follow.

periodic-function-verbose [*Variable*]

Purpose
Controls whether initiation and termination of periodic-function threads are printed as comments.

Package
`:portable-threads`

Module
`:portable-threads`

Value type
A generalized boolean

Initial value
`nil`

Description
The value of ***periodic-function-verbose*** can be changed globally to display the management of periodic functions.

See also
kill-periodic-function (page 272)
spawn-periodic-function (page 294)

Example
Schedule a simple periodic function with verbose printing enabled:

```
> (setf *periodic-function-verbose* 't)
t
> (spawn-periodic-function #'(lambda () (print "Hello!")) 0.1
    :name 'hello
    :count 2)
;; Spawning periodic-function thread for hello...
#<thread Periodic Function hello>
>
"Hello!"
"Hello!"
;; Exiting periodic-function thread hello
>
```

schedule-function-verbose [*Variable*]

Purpose
Controls whether scheduling changes made to scheduled functions are printed as comments.

Package `:portable-threads`

Module `:portable-threads`

Value type A generalized boolean

Initial value `nil`

Description
The value of ***schedule-function-verbose*** can be changed globally to display the activities of the scheduled function scheduler.

See also
schedule-function (page 278)
schedule-function-relative (page 281)
unschedule-function (page 296)

Example
Change the invocation time of scheduled function `quitting-time` from 5pm to 5:30pm with verbose printing enabled:

```
> (setf *schedule-function-verbose* 't)
t
> (schedule-function 'quitting-time (encode-time-of-day 0 30 17)
    :repeat-interval #.(* 24 60 60))
;; Unscheduling #<scheduled-function quitting-time [17:00:00]>...
;; Scheduling #<scheduled-function quitting-time [17:30:00]>
;; as the next scheduled-function...
>
```

all-scheduled-functions <no arguments> ⇒ *list-of-scheduled-functions* [*Function*]

Purpose
Return the list of all scheduled functions that are currently scheduled.

Package
`:portable-threads`

Module
`:portable-threads`

Arguments
list-of-scheduled-functions A proper list

Returns
A list of `scheduled-function` objects.

See also
make-scheduled-function (page 273)
schedule-function (page 278)
schedule-function-relative (page 281)
unschedule-function (page 296)

Example
```
> (all-threads)
(#<thread Listener 1>)
>
```

Notes
On Common Lisp implementations without threads, `nil` is returned.

The returned list of scheduled functions should not be destructively altered. In particular, unscheduling a scheduled function (using **unschedule-function**) or changing a scheduled function's invocation time (using **schedule-function**) may modify the list of scheduled functions, so a copy of the returned list of scheduled functions should be used for any iteration that involves these operations.

kill-periodic-function <no arguments> [*Function*]

Purpose
Terminate the thread invoking a periodic function.

Package :portable-threads

Module :portable-threads

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Kill-periodic-function called outside the dynamic scope of a periodic function.

See also

periodic-function-verbose (page 269)
all-threads (page 226)
kill-thread (page 246)
spawn-periodic-function (page 294)

Example
Define and spawn a periodic function that is invoked every 0.5 seconds to signal a `half-second-interrupt-event`, continuing as long as the control shell is running:

```
> (define-event-class half-second-timer-event (timer-interrupt-event)
    ())
half-second-timer-event
> (defun half-second-timer ()
    (unless (control-shell-running-p)
      (kill-periodic-function))
    (signal-event 'half-second-timer-event))
half-second-timer
> (spawn-periodic-function 'half-second-timer 0.5)
#<thread Periodic Function half-second-timer>
>
```

make-scheduled-function *function* `&key` *name name-test marker marker-test context* [*Function*]
⇒ *scheduled-function*

Purpose
Create a scheduled function.

Package `:portable-threads`

Module `:portable-threads`

Arguments
function	A function designator specifying a function object of one argument
name	An object (typically a string or a symbol; default is *function*, if *function* is a symbol, otherwise `nil`)
name-test	A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'eql`)
marker	An object (default is `nil`)
marker-test	A function designator specifying a function object of two arguments that returns a generalized boolean (default is `#'eql`)
context	An object (default is `nil`)
scheduled-function	A scheduled function

Returns
The newly created scheduled function.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Description
Unless the run time to perform the scheduled function is brief, it should spawn a new thread in which to perform its activities so as to avoid delaying the invocation of a subsequent scheduled function.

The optional *marker* and associated comparison *marker-test* can be specified to distinguish scheduled functions with the same *name* in calls to **schedule-function**, **schedule-function-relative**, and **unschedule-function**.

The optional *context* object can be specified to store an invocation context in the *scheduled-function* object that is passed to the scheduled *function* when it is invoked.

See also

schedule-function-verbose	(page 270)
all-scheduled-functions	(page 271)
pause-scheduled-function-scheduler	(page 275)
restart-scheduled-function-scheduler	(page 276)
resume-scheduled-function-scheduler	(page 277)
schedule-function	(page 278)
schedule-function-relative	(page 281)
scheduled-function-context	(page 284)

Examples

Create a scheduled function that simply prints `"Hello"` when invoked:

```
> (make-scheduled-function
    #'(lambda (scheduled-function)
        (declare (ignore scheduled-function))
        (print "Hello"))
    :name 'hello)
#<scheduled-function hello [unscheduled]>
>
```

Create a scheduled function that individualizes the `"Hello"` when invoked:

```
> (make-scheduled-function
    #'(lambda (scheduled-function)
        (format t "~&Hello ~a~%"
          (scheduled-function-context scheduled-function)))
    :name 'hello
    :context "Bob")
#<scheduled-function hello [unscheduled]>
>
```

A more complex scheduled function that spawns a new thread to do its work and randomly sets whether to reschedule itself (and at what interval):

```
> (defun complex-function (scheduled-function)
    (let ((interval (random 100)))
      (setf (scheduled-function-repeat-interval scheduled-function)
            (if (plusp interval)
                ;; repeat 1-99 seconds from now:
                interval
                ;; don't repeat 1% of the time:
                nil)))
    (spawn-thread "Lots of stuff doer" #'do-lots-of-stuff))
complex-function
> (make-scheduled-function 'complex-function)
#<scheduled-function complex-function [unscheduled]>
>
```

pause-scheduled-function-scheduler <no arguments> [*Function*]

Purpose
Pause scheduled-function scheduling.

Package :portable-threads

Module :portable-threads

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Description
Pausing the scheduled-function scheduler causes all scheduled functions whose invocation time arrives to be held pending until the scheduled-function scheduler is resumed.

See also
resume-scheduled-function-scheduler (page 277)
scheduled-function-scheduler-paused-p (page 292)
scheduled-function-scheduler-running-p (page 293)

Example
Pause the scheduled-function scheduler:

```
> (scheduled-function-scheduler-paused-p)
nil
> (pause-scheduled-function-scheduler)
> (scheduled-function-scheduler-paused-p)
t
>
```

restart-scheduled-function-scheduler <no arguments> ⇒ *thread* [*Function*]

Purpose
Restart the scheduled-function scheduling thread.

Package :portable-threads

Module :portable-threads

Arguments
thread A thread or `nil`

Returns
The object representing the newly spawned scheduled-function scheduler thread or `nil` if the scheduled-function scheduler was already running.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Description
If the scheduled-function scheduler thread has been killed accidentally, this function can be used to start a new scheduler thread.

See also

resume-scheduled-function-scheduler (page 277)
schedule-function (page 278)
scheduled-function-repeat-interval (page 290)
scheduled-function-scheduler-paused-p (page 292)
scheduled-function-scheduler-running-p (page 293)
unschedule-function (page 296)

Examples
Restart the scheduled-function scheduler:

```
> (restart-scheduled-function-scheduler)
#<thread Scheduled-Function Scheduler>
>
```

Restarting a scheduled-function scheduler that is already running has no effect:

```
> (restart-scheduled-function-scheduler)
;; The scheduled-function scheduler is already running.
nil
>
```

resume-scheduled-function-scheduler <no arguments> [*Function*]

Purpose
Resume scheduled-function scheduling.

Package
`:portable-threads`

Module
`:portable-threads`

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Description
Resuming the scheduled-function scheduler causes all scheduled functions that have been held pending while the scheduled-function scheduler was paused to be invoked as if their invocation time had just occurred.

See also
pause-scheduled-function-scheduler (page 275)
scheduled-function-scheduler-paused-p (page 292)
scheduled-function-scheduler-running-p (page 293)

Example
Resume the scheduled-function scheduler:

```
> (scheduled-function-scheduler-paused-p)
t
> (resume-scheduled-function-scheduler)
> (scheduled-function-scheduler-paused-p)
nil
>
```

schedule-function *name-or-scheduled-function invocation-time* [*Function*]
`&key` *marker context repeat-interval verbose*

Purpose
Schedule a scheduled function at an absolute invocation time.

Package `:portable-threads`

Module `:portable-threads`

Arguments
name-or-scheduled-function	An object (typically a string or a symbol) naming a currently scheduled scheduled function or a `scheduled-function` object
invocation-time	A Universal Time
marker	An object (default is `nil`)
context	An object (default is `nil`)
repeat-interval	A positive integer (representing seconds) or `nil` (default is `nil`)
verbose	A generalized boolean (default is ***schedule-function-verbose***)

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Description
If the scheduled function is unscheduled, the `scheduled-function` object must be specified as the *name-or-scheduled-function* value. In this case, the scheduled function it is added to the list of currently scheduled scheduled functions with the specified *invocation-time* and optional *repeat-interval*, if specified.

If the *scheduled-function* object is currently scheduled, either the `scheduled-function` object or the optional *name* value that was specified when the `scheduled-function` object was created with **make-scheduled-function** can be specified as the *name-or-scheduled-function* value. If a name is specified as the *name-or-scheduled-function* value and more than one scheduled function with the specified *name* is currently scheduled, the scheduled function with the earliest invocation time is selected. If an optional *marker* value was specified along with *name* when the `scheduled-function` object was created with **make-scheduled-function**, the *marker* value can also be specified to restrict the selected scheduled function to the one with the earliest invocation time that matches both the *name* and *marker* values. The selected scheduled function is first unscheduled and then rescheduled with the specified *invocation-time* and optional *repeat-interval*, if specified.

The optional *context* object can be specified to replace the invocation context in the *scheduled-function* object.

See also

schedule-function-verbose	(page 270)
all-scheduled-functions	(page 271)
encode-time-of-day	(page 171)
make-scheduled-function	(page 273)
restart-scheduled-function-scheduler	(page 276)
schedule-function-relative	(page 281)

Examples

Schedule a scheduled function that simply prints `"Happy New Year!"` at midnight (local time) on January 1, 2014:

```
> (schedule-function
    (make-scheduled-function
      #'(lambda (scheduled-function)
          (declare (ignore scheduled-function))
          (print "Happy New Year!")))
     (encode-universal-time 0 0 0 1 1 2014))
> (all-scheduled-functions)
(#<scheduled-function nil [Jan 1, 2014 00:00:00]>)
>
```

Schedule a scheduled function that prints `"It's quitting time!"` every day at 5pm:

```
> (schedule-function
    (make-scheduled-function
      #'(lambda (scheduled-function)
          (declare (ignore scheduled-function))
          (print "It's quitting time!"))
      :name 'quitting-time)
    (encode-time-of-day 0 0 17)
    :repeat-interval #.(* 24 60 60))
>
```

Verbosely change `quitting-time` to 5:30pm every day:

```
> (schedule-function 'quitting-time (encode-time-of-day 0 30 17)
    :repeat-interval #.(* 24 60 60)
    :verbose 't)
;; Unscheduling #<scheduled-function quitting-time [17:00:00]>...
;; Scheduling #<scheduled-function quitting-time [17:30:00]>
;; as the next scheduled-function...
>
```

Schedule a scheduled function that prints an individualized `"Hello"` message to Bob every day at 9am:

```
> (schedule-function
    (make-scheduled-function
      #'(lambda (scheduled-function)
          (format t "~&Hello ~a~%"
            (scheduled-function-context scheduled-function)))
      :name 'hello
      :context "Bob")
    (encode-time-of-day 0 0 9)
    :repeat-interval #.(* 24 60 60))
>
```

Verbosely change the scheduled function to prints the individualized `"Hello"` message to Lisa every day at 8:30am (Bob will no longer be greeted):

```
> (schedule-function 'hello (encode-time-of-day 0 30 8)
    :repeat-interval #.(* 24 60 60)
    :context "Lisa"
    :verbose 't)
;; Unscheduling #<scheduled-function hello [09:00:00]>...
;; Scheduling #<scheduled-function hello [08:30:00]>
;; as the next scheduled-function...
>
```

---

schedule-function

schedule-function-relative *name-or-scheduled-function seconds* [*Function*]
`&key` *context marker repeat-interval verbose*

Purpose
Schedule a scheduled function a specified number of seconds from now.

Package `:portable-threads`

Module `:portable-threads`

Arguments
name-or-scheduled-function	An object (typically a string or a symbol) naming a currently scheduled scheduled function or a `scheduled-function` object
seconds	A positive integer
context	An object (default is `nil`)
marker	An object (default is `nil`)
repeat-interval	A positive integer (representing seconds) or `nil` (default is `nil`)
verbose	A generalized boolean (default is ***schedule-function-verbose***)

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Description
If the scheduled function is unscheduled, the `scheduled-function` object must be specified as the *name-or-scheduled-function* value. In this case, the scheduled function it is added to the list of currently scheduled scheduled functions with an invocation time of *interval* seconds from the current time and optional *repeat-interval*, if specified.

If the *scheduled-function* object is currently scheduled, either the `scheduled-function` object or the optional *name* value that was specified when the `scheduled-function` object was created with **make-scheduled-function** can be specified as the *name-or-scheduled-function* value. If a name is specified as the *name-or-scheduled-function* value and more than one scheduled function with the specified *name* is currently scheduled, the scheduled function with the earliest invocation time is selected. If an optional *marker* value was specified along with *name* when the `scheduled-function` object was created with **make-scheduled-function**, the *marker* value can also be specified to restrict the selected scheduled function to the one with the earliest invocation time that matches both the *name* and *marker* values. The selected scheduled function is first unscheduled and then rescheduled with an invocation time of *interval* seconds from the current time and optional *repeat-interval*, if specified.

The optional *context* object can be specified to replace the invocation context in the *scheduled-function* object.

See also

schedule-function-verbose	(page 270)
all-scheduled-functions	(page 271)
make-scheduled-function	(page 273)
restart-scheduled-function-scheduler	(page 276)
schedule-function	(page 278)

Examples

Schedule a scheduled function that simply prints `"Hello!"` 5 seconds from now:

```
> (schedule-function-relative
    (make-scheduled-function
      #'(lambda (scheduled-function)
          (declare (ignore scheduled-function))
          (print "Hello!")))
     5)
>
```

Schedule a scheduled function that signals a GBBopen `timer-interrupt-event` every 30 seconds:

```
> (schedule-function-relative
    (make-scheduled-function
      #'(lambda (scheduled-function)
          (declare (ignore scheduled-function))
          (signal-event 'timer-interrupt-event)))
    30
    :repeat-interval 30)
>
```

Schedule a scheduled function that prints an individualized `"Hello"` message to Bob five seconds from now, repeating annoyingly every hour thereafter:

```
> (schedule-function-relative
    (make-scheduled-function
      #'(lambda (scheduled-function)
          (format t "~&Hello ~a~%"
            (scheduled-function-context scheduled-function)))
      :name 'hello
      :context "Bob")
    5
    :repeat-interval 3600)
>
```

Verbosely change Bob's annoying repeating `"Hello"` message to greet Lisa only once, 10 seconds from now:

```
> (schedule-function-relative 'hello 10
    :context "Lisa"
    :verbose 't)
;; Unscheduling #<scheduled-function hello [09:46:11]>...
;; Scheduling #<scheduled-function hello [09:46:27]>...
>
```

Note

The form `(schedule-function-relative` ***scheduled-function*** `10)` is equivalent to `(schedule-function` ***scheduled-function*** `(+ (get-universal-time) 10))`.

---

schedule-function-relative

scheduled-function-context *scheduled-function* ⇒ *object* [*Function*]

Purpose
Return the context object of a scheduled function.

Package :portable-threads

Module :portable-threads

Arguments
scheduled-function A scheduled function
context An object

Returns
The context object of *scheduled-function*.

See also
all-scheduled-functions (page 271)
make-scheduled-function (page 273)
schedule-function (page 278)
schedule-function-relative (page 281)
scheduled-function-invocation-time (page 285)
scheduled-function-marker (page 286)
scheduled-function-marker-test (page 287)
scheduled-function-name (page 288)
scheduled-function-name-test (page 289)
scheduled-function-repeat-interval (page 290)

Example
Return the context object of scheduled function `scheduled-function`:

```
> (scheduled-function-context scheduled-function)
"Bob"
>
```

scheduled-function-invocation-time *scheduled-function* ⇒ *invocation-time* [*Function*]

Purpose
Return the invocation time of a scheduled function.

Package :portable-threads

Module :portable-threads

Arguments
scheduled-function A scheduled function
invocation-time A Universal Time

Returns
The invocation time of *scheduled-function*.

See also
all-scheduled-functions (page 271)
make-scheduled-function (page 273)
schedule-function (page 278)
schedule-function-relative (page 281)
scheduled-function-context (page 284)
scheduled-function-marker (page 286)
scheduled-function-marker-test (page 287)
scheduled-function-name (page 288)
scheduled-function-name-test (page 289)
scheduled-function-repeat-interval (page 290)

Example
Return the invocation-time of scheduled function `scheduled-function`:

```
> (scheduled-function-invocation-time scheduled-function)
3465679813
>
```

scheduled-function-marker *scheduled-function* ⇒ *marker* [*Function*]

Purpose
Return the marker of a scheduled function.

Package :portable-threads

Module :portable-threads

Arguments
scheduled-function A scheduled function
marker An object

Returns
The marker of *scheduled-function*.

See also
all-scheduled-functions (page 271)
make-scheduled-function (page 273)
schedule-function (page 278)
schedule-function-relative (page 281)
scheduled-function-context (page 284)
scheduled-function-invocation-time (page 285)
scheduled-function-marker-test (page 287)
scheduled-function-name (page 288)
scheduled-function-name-test (page 289)
scheduled-function-repeat-interval (page 290)

Example
Return a list of the markers of all currently scheduled scheduled functions:

```
> (mapcar #'scheduled-function-marker (all-scheduled-functions))
(nil)
>
```

scheduled-function-marker-test *scheduled-function* ⇒ *marker-predicate* [*Function*]

Purpose

Return the marker-comparison predicate of a scheduled function.

Package

`:portable-threads`

Module

`:portable-threads`

Arguments

scheduled-function A scheduled function

marker-predicate A function designator specifying a function object of two arguments that returns a generalized boolean

Returns

The marker-comparison predicate of *scheduled-function*.

See also

all-scheduled-functions (page 271)
make-scheduled-function (page 273)
schedule-function (page 278)
schedule-function-relative (page 281)
scheduled-function-context (page 284)
scheduled-function-invocation-time (page 285)
scheduled-function-marker (page 286)
scheduled-function-name (page 288)
scheduled-function-name-test (page 289)
scheduled-function-repeat-interval (page 290)

Example

Return a list of the marker-comparison predicates of all currently scheduled scheduled functions:

```
> (mapcar #'scheduled-function-marker-test (all-scheduled-functions))
(#'eql)
>
```

scheduled-function-name *scheduled-function* ⇒ *name* [*Function*]

Purpose
Return the name of a scheduled function.

Package :portable-threads

Module :portable-threads

Arguments
scheduled-function A scheduled function
name An object (typically a string or a symbol)

Returns
The name of *scheduled-function*.

See also
all-scheduled-functions (page 271)
make-scheduled-function (page 273)
schedule-function (page 278)
schedule-function-relative (page 281)
scheduled-function-context (page 284)
scheduled-function-invocation-time (page 285)
scheduled-function-marker (page 286)
scheduled-function-marker-test (page 287)
scheduled-function-name-test (page 289)
scheduled-function-repeat-interval (page 290)

Example
Return the names of all currently scheduled scheduled functions:

```
> (mapcar #'scheduled-function-name (all-scheduled-functions))
(quitting-time)
>
```

scheduled-function-name-test *scheduled-function* ⇒ *name-predicate* [*Function*]

Purpose

Return the name-comparison predicate of a scheduled function.

Package

`:portable-threads`

Module

`:portable-threads`

Arguments

scheduled-function	A scheduled function
name-predicate	A function designator specifying a function object of two arguments that returns a generalized boolean

Returns

The name-comparison predicate of *scheduled-function*.

See also

all-scheduled-functions (page 271)
make-scheduled-function (page 273)
schedule-function (page 278)
schedule-function-relative (page 281)
scheduled-function-context (page 284)
scheduled-function-invocation-time (page 285)
scheduled-function-marker (page 286)
scheduled-function-name (page 288)
scheduled-function-name-test (page 289)
scheduled-function-repeat-interval (page 290)

Example

Return a list of the name-comparison predicates of all currently scheduled scheduled functions:

```
> (mapcar #'scheduled-function-name-test (all-scheduled-functions))
(#'eql)
>
```

scheduled-function-repeat-interval *scheduled-function* ⇒ *repeat-interval* [*Function*]

Purpose
Return the repeat interval of a scheduled function.

Setf syntax
(setf (scheduled-function-repeat-interval *scheduled-function*) *repeat-interval*) ⇒ *repeat-interval*

Package :portable-threads

Module :portable-threads

Arguments
scheduled-function A scheduled function
repeat-interval A positive integer (representing seconds) or `nil`

Returns
The repeat interval of *scheduled-function*.

See also
all-scheduled-functions (page 271)
make-scheduled-function (page 273)
schedule-function (page 278)
schedule-function-relative (page 281)
scheduled-function-context (page 284)
scheduled-function-invocation-time (page 285)
scheduled-function-marker (page 286)
scheduled-function-marker-test (page 287)
scheduled-function-name (page 288)
scheduled-function-name-test (page 289)
scheduled-function-repeat-interval (page 290)

Examples
Display the `scheduled-function` object and its repeat interval for each currently scheduled scheduled function:

```
> (dolist (scheduled-function (all-scheduled-functions))
   (format t "~&;; ~s ~s~%"
           scheduled-function
           (scheduled-function-repeat-interval scheduled-function)))
;; #<scheduled-function quitting-time [17:00:00]> 86400
nil
>
```

Define a function to be used as a scheduled function that randomly sets whether to reschedule itself (and at what interval):

```
(defun complex-function (scheduled-function)
  (let ((interval (random 100)))
    (setf (scheduled-function-repeat-interval scheduled-function)
          (if (plusp interval)
              ;; repeat 1-99 seconds from now:
              interval
              ;; don't repeat 1% of the time:
              nil)))
  (do-some-stuff))
```

scheduled-function-repeat-interval

scheduled-function-scheduler-paused-p <no arguments> ⇒ *boolean* [*Function*]

Purpose
Determine if scheduled-function scheduling is paused.

Package
`:portable-threads`

Module
`:portable-threads`

Arguments
boolean A generalized boolean

Returns
True if the scheduled-function scheduler is paused; `nil` otherwise.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
pause-scheduled-function-scheduler (page 275)
resume-scheduled-function-scheduler (page 277)
scheduled-function-scheduler-running-p (page 293)

Example
Pause the scheduled-function scheduler:

```
> (scheduled-function-scheduler-paused-p)
nil
> (pause-scheduled-function-scheduler)
> (scheduled-function-scheduler-paused-p)
t
>
```

scheduled-function-scheduler-running-p <no arguments> ⇒ *boolean* [*Function*]

Purpose
Determine if the scheduled-function scheduler is running.

Package
`:portable-threads`

Module
`:portable-threads`

Arguments
boolean A generalized boolean

Returns
True if the scheduled-function scheduler is running; `nil` otherwise.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
pause-scheduled-function-scheduler (page 275)
restart-scheduled-function-scheduler (page 276)
resume-scheduled-function-scheduler (page 277)
scheduled-function-scheduler-paused-p (page 292)

Example
Check that the scheduled-function scheduler is running:

```
> (scheduled-function-scheduler-running-p)
t
>
```

spawn-periodic-function *function repeat-interval* `&key` *count name verbose* ⇒ *thread* [*Function*]

Purpose
Spawn a thread invoking *function* every *repeat-interval* seconds.

Package `:portable-threads`

Module `:portable-threads`

Arguments
function	A function designator specifying a function object of no arguments
repeat-interval	A number (representing seconds)
count	A number or `nil` (default is `nil`)
name	An object (typically a string or a symbol; default is *function*, if *function* is a symbol, otherwise `nil`)
verbose	A generalized boolean (default is ***periodic-function-verbose***)
thread	A thread

Returns
The object representing the thread associated with the periodic function.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Description
If *count* is `nil`, *function* will continue to be invoked every *repeat-interval* seconds until the periodic-function thread is killed or until *function* calls **kill-periodic-function**. Otherwise, *count* is decremented by one prior to each invocation of *function* and, if it is negative, the periodic function is terminated.

See also

periodic-function-verbose	(page 269)
all-threads	(page 226)
kill-periodic-function	(page 272)
kill-thread	(page 246)
make-scheduled-function	(page 273)
schedule-function	(page 278)
schedule-function-relative	(page 281)

Examples
Spawn a simple periodic function that is invoked every 0.1 seconds, but that only runs twice:

```
> (spawn-periodic-function #'(lambda () (print "Hello!")) 0.1
    :name 'hello
    :count 2)
#<thread Periodic Function hello>
>
```

```
"Hello!"
"Hello!"
```

Spawn a simple periodic function that is invoked every 0.1 seconds that runs up to 20 times, but with a 10% chance on each invocation of terminating early:

```
> (spawn-periodic-function
     #'(lambda ()
         (when (zerop (random 10))
           (kill-periodic-function))
         (print "Hello!"))
     0.1
     :count 20
     :verbose 't)
;; Spawning periodic-function thread for...
#<thread Periodic Function>
>
"Hello!"
"Hello!"
"Hello!"
"Hello!"
;; Killing periodic-function...
;; Exiting periodic-function thread
```

Define and spawn a periodic function that is invoked every 0.5 seconds to signal a `half-second-interrupt-event`, continuing as long as the control shell is running:

```
> (define-event-class half-second-timer-event (timer-interrupt-event)
    ())
half-second-timer-event
> (defun half-second-timer ()
    (unless (control-shell-running-p)
      (kill-periodic-function))
    (signal-event 'half-second-timer-event))
half-second-timer
> (spawn-periodic-function 'half-second-timer 0.5)
#<thread Periodic Function half-second-timer>
>
```

spawn-periodic-function

unschedule-function *name-or-scheduled-function* `&key` *marker warnp verbose* [*Function*]
⇒ *boolean*

Purpose
Cancel the upcoming invocation (and subsequent repeat-interval scheduling) of a currently scheduled scheduled function.

Package `:portable-threads`

Module `:portable-threads`

Arguments
name-or-scheduled-function	An object (typically a string or a symbol) naming a currently scheduled scheduled function or a `scheduled-function` object
marker	An object (default is `nil`)
verbose	A generalized boolean (default is ***schedule-function-verbose***)
warnp	A generalized boolean (default is `t`)
boolean	A generalized boolean

Returns
The scheduled function if it was unscheduled; `nil` if the scheduled function was not currently scheduled or was not found.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Description
If the *scheduled-function* object is currently scheduled, either the `scheduled-function` object or the optional *name* value that was specified when the `scheduled-function` object was created with **make-scheduled-function** can be specified as the *name-or-scheduled-function* value. If a name is specified as the *name-or-scheduled-function* value and more than one scheduled function with the specified *name* is currently scheduled, the scheduled function with the earliest invocation time is selected. If an optional *marker* value was specified along with *name* when the `scheduled-function` object was created with **make-scheduled-function**, the *marker* value can also be specified to restrict the selected scheduled function to the one with the earliest invocation time that matches both the *name* and *marker* values. The selected scheduled function is removed from the list of currently scheduled scheduled functions.

If *warnp* is true, a warning is issued if the scheduled function was not currently scheduled or was not found.

See also

schedule-function-verbose	(page 270)
all-scheduled-functions	(page 271)
make-scheduled-function	(page 273)
schedule-function	(page 278)
schedule-function-relative	(page 281)

Examples

Unschedule the `quitting-time` scheduled function:

```
> (unschedule-function 'quitting-time)
#<scheduled-function quitting-time [unscheduled]>
>
```

Unschedule all currently scheduled scheduled functions:

```
> (all-scheduled-functions)
(#<scheduled-function nil [Jan 1, 2014 00:00:00]>)
> (mapc #'unschedule-function (all-scheduled-functions))
(#<scheduled-function nil [unscheduled]>)
> (all-scheduled-functions)
nil
>
```

Unschedule a non-existent scheduled function:

```
> (unschedule-function 'non-existent)
;; Warning: Scheduled-function non-existent was not scheduled; no action
taken.
nil
>
```

---

unschedule-function

4.3 Polling Functions

The `:polling-functions` module provides a set of *polling functions* that can be used to support "event-loop" processing on Common Lisp implementations that do not provide threads. These functions are available for use with all Common Lisp implementations.

add-polling-function *function* `&key` *priority* [*Function*]

Purpose
Add a polling function to the list of polling functions at the position indicated by *priority*.

Package `:gbbopen-tools`

Module `:polling-functions`

Arguments
function A function designator specifying a function object of no arguments
priority A `fixnum` (default is `0`)

Description
The description is printed to the ***standard-output*** stream.

See also
describe-all-polling-functions (page 300)
remove-all-polling-functions (page 302)
remove-polling-function (page 301)
run-polling-functions (page 303)

Example
Add the function `check-for-new-connection` to the list of polling functions (with priority -10):

```
(add-polling-function #'check-for-new-connection
   :priority -10)
```

describe-all-polling-functions <no arguments> [*Function*]

Purpose
Describe the polling functions in the list of polling functions.

Package `:gbbopen-tools`

Module `:polling-functions`

Description
The description is printed to the ***standard-output*** stream.

See also

add-polling-function (page 299)
remove-all-polling-functions (page 302)
remove-polling-function (page 301)
run-polling-functions (page 303)

Example
Describe list of polling functions:

```
> (describe-all-polling-functions)
;; Polling functions:
;;    -10 #<Function check-for-new-connection>
>
```

remove-polling-function *function* [*Function*]

Purpose
Remove a polling function from the list of polling functions.

Package `:gbbopen-tools`

Module `:polling-functions`

Arguments
function A function designator

See also

add-polling-function (page 299)
describe-all-polling-functions (page 300)
remove-all-polling-functions (page 302)
run-polling-functions (page 303)

Example
Remove the function `check-for-new-connection` from the list of polling functions:

```
(remove-polling-function #'check-for-new-connection)
```

remove-all-polling-functions <no arguments> [*Function*]

Purpose

Remove all polling functions from the list of polling functions.

Package

`:gbbopen-tools`

Module

`:polling-functions`

See also

add-polling-function (page 299)
describe-all-polling-functions (page 300)
remove-polling-function (page 301)
run-polling-functions (page 303)

Example

Remove all functions from the list of polling functions:

```
(remove-all-polling-functions)
```

run-polling-functions <no arguments> [*Function*]

Purpose

Run every polling function in the list of polling functions.

Package

`:gbbopen-tools`

Module

`:polling-functions`

See also

add-polling-function (page 299)
describe-all-polling-functions (page 300)
remove-all-polling-functions (page 302)
remove-polling-function (page 301)
start-control-shell (page 622)

Example

Run the polling functions (once, in sequence):

```
(run-polling-functions)
```

Note

When a non-`nil` `:run-polling-functions` value is supplied to **start-control-shell** (the default on Common Lisp implementations without threads), **run-polling-functions** is called at the beginning of every control-shell-cycle and at one-half-second intervals when the Agenda Shell is hibernating due to quiescence.

4.4 Portable Sockets

The `:portable-sockets` module provides a uniform interface to commonly used socket entities.

accept-connection *passive-socket* `&key` *wait* ⇒ *socket-stream* [*Function*]

Purpose
Accept a socket-stream connection.

Package `:portable-sockets`

Module `:portable-sockets`

Arguments
passive-socket A passive socket
wait A generalized boolean (default is `t`)
socket-stream A socket stream

Returns
A socket stream.

See also
shutdown-socket-stream (page 311)
start-connection-server (page 312)

Example
Accept a connection made to a newly created passive socket:

```
> (let* ((passive-socket (make-passive-socket 5555))
         (connection (accept-connection passive-socket)))
    (close-passive-socket passive-socket)
    connection)
#<socket stream connected from localhost/5555 to localhost/59946>
>
```

Note
Connections should always be closed using **close** (from both sides) to free up operating-system resources when they are no longer needed.

close-passive-socket *passive-socket* [*Function*]

Purpose
Close a passive socket.

Package :portable-sockets

Module :portable-sockets

Arguments
passive-socket A passive socket

See also
make-passive-socket (page 308)

Example
Close a passive socket:

```
(close-passive-socket passive-socket)
```

local-hostname-and-port *socket-stream* `&optional` *do-not-resolve* ⇒ *hostname*, *port* [*Function*]

Purpose

Return the name of the host on the local side of the *socket-stream* connection and its port number.

Package `:portable-sockets`

Module `:portable-sockets`

Arguments

socket-stream	A socket stream
do-not-resolve	A generalized boolean (default is `nil`)
hostname	A string
port	An integer

Returns

Two values:

- a string containing the name of the local host
- the integer port number at the local host

See also

open-connection (page 309)
remote-hostname-and-port (page 310)
with-open-connection (page 314)

Examples

Return the local hostname and port of an open socket-stream connection to the `wiki.alu.org` web server:

```
> (local-hostname-and-port connection)
"192.168.240.104 (ruby.gbbopen.org)"
56833
>
```

Return the local hostname and port of the open socket-stream connection, but without hostname resolution:

```
> (local-hostname-and-port connection 't)
"192.168.240.104"
56833
>
```

make-passive-socket *port* `&key` *backlog interface reuse-address* ⇒ *passive-socket* [*Function*]

Purpose
Create a passive socket that can accept connections.

Package `:portable-sockets`

Module `:portable-sockets`

Arguments
port	An integer or a string specifying the service port
backlog	An integer (default is 5)
interface	A 32-bit internet address or a string specifying a network interface on the local machine or `nil`
reuse-address	A generalized boolean (default is `nil`)
passive-socket	A passive socket

Returns
The new passive socket.

Description
An *interface* string can be either a host name, such as `"localhost"` or a “dotted” IP address, such as `"127.0.0.1"`.

The value of *backlog* tells the operating system how many unprocessed connections can be held pending (connected but still awaiting an **accept-connection**).

See also
accept-connection (page 305)
start-connection-server (page 312)

Example
Create a passive socket, listening on port 5555:

```
> (make-passive-socket 5555)
#<passive socket waiting for connection at */5555>
>
```

Note
The passive socket should be closed using **close-passive-socket** when the service is no longer needed in order to free up operating system resources.

open-connection *host port* ⇒ *socket-stream* [*Generic Function*]

Purpose
Open a socket-stream connection to server *host*.

Method signatures
open-connection (*host* `integer`) *port* ⇒ *socket-stream*

open-connection (*host* `string`) *port* ⇒ *socket-stream*

Package `:portable-sockets`

Module `:portable-sockets`

Arguments
host	A 32-bit internet address or a string specifying the remote host
port	An integer or a string specifying the service port
socket-stream	A socket stream

Returns
A socket stream.

Description
A *host* string can be either a host name or a "dotted" IP address, such as `"127.0.0.1"`.

String values available for specifying *port* are found in the operating system's services file and labeled as being `tcp` services. On Unix systems, the services file is `/etc/services`. On Windows, it is the file `services` in the `Windows` directory.

See also
shutdown-socket-stream (page 311)
with-open-connection (page 314)

Example
Open a socket connection to the GBBopen Project web server:

```
> (open-connection "GBBopen.org" 80)
#<socket stream connected from localhost/51756 to gbbopen.org/80>
>
```

Note
Connections should always be closed using **close** (from both sides) when they are no longer needed to free up operating-system resources.

remote-hostname-and-port *socket-stream* `&optional` *do-not-resolve* [*Function*]
⇒ *hostname*, *port*

Purpose
Return the name of the host on the remote side of the *socket-stream* connection and its port number.

Package `:portable-sockets`

Module `:portable-sockets`

Arguments
socket-stream A socket stream
do-not-resolve A generalized boolean (defaulto is `nil`)
hostname A string
port An integer

Returns
Two values:

- a string containing the name of the remote host
- the integer port number at the remote host

See also
open-connection (page 309)
local-hostname-and-port (page 307)
with-open-connection (page 314)

Examples
Return the remote hostname and port of an open socket-stream connection to the `wiki.alu.org` web server:

```
> (remote-hostname-and-port connection)
"206.169.106.4 (bibop.alu.org)"
80
>
```

Return the remote hostname and port of the open socket-stream connection, but without hostname resolution:

```
> (remote-hostname-and-port connection 't)
"206.169.106.4"
80
>
```

shutdown-socket-stream *socket-stream direction* [*Function*]

Purpose
Shut down (close) one direction of an open connection.

Package `:portable-sockets`

Module `:portable-sockets`

Arguments
socket-stream A socket stream
direction The keyword symbol `:input` or `:output` specifying the direction to be closed

See also
start-connection-server (page 312)
open-connection (page 309)
with-open-connection (page 314)

Example
Tell the other end of a socket connection that we are done sending output on the socket stream (send an end-of-file indication):

```
(shutdown-socket-stream socket-stream ':output)
```

Note
Connections should always be closed using **close** (from both sides) when they are no longer needed to free up operating-system resources.

start-connection-server *function port* `&key` *backlog interface name reuse-address* [*Function*]
⇒ *thread*

Purpose
Create a connection-server thread that accepts connections and processes them according to *function*.

Package `:portable-sockets`

Module `:portable-sockets`

Arguments
function	A function designator specifying a function object of one argument
port	An integer or a string specifying the service port
backlog	An integer (default is 5)
interface	A 32-bit internet address or a string specifying a network interface on the local machine or `nil`
name	A string (default is `"Connection Server"`)
reuse-address	A generalized boolean (default is `nil`)
thread	A thread

Returns
The new connection-server thread.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Description
The connection server will not accept another connection until *function* returns, so normally *function* should spawn another thread to handle the connection.

An *interface* string can be either a host name, such as `"localhost"` or a “dotted” IP address, such as `"127.0.0.1"`.

The value of *backlog* tells the operating system how many unprocessed connections can be held pending (connected but still awaiting an **accept-connection**).

See also

kill-thread (page 246)
open-connection (page 309)
with-open-connection (page 314)

Example
Start a simple connection server that accepts connections on port 5555, reads one line of input, and closes the connection:

```
> (start-connection-server
     #'(lambda (connection)
          (let ((line (read-line connection nil)))
             (format t "~&;; New Connection: ~a~%" line)
```

```
                (close connection)))
      5555)
  #<thread Connection Server>
```

Note

Use **kill-thread** to kill the connection-server thread.

---

start-connection-server

with-open-connection (*var host port*) *declaration** *form** [*Macro*]

Purpose

Open a socket-stream connection to server *host*, perform a series of operations on the connection, and then close the connection.

Package :portable-sockets

Module :portable-sockets

Arguments

var A variable symbol
host A 32-bit internet address or a string specifying the remote host
port An integer or a string specifying the service port
declaration A declare expression (not evaluated)
forms An implicit **progn** of forms to be evaluated

Description

This macro ensures that the opened connection is closed when control leaves the body of the macro.

A *host* string can be either a host name or a "dotted" IP address, such as `"127.0.0.1"`.

String values available for specifying *port* are found in the operating system's services file and labeled as being `tcp` services. On Unix systems, the services file is `/etc/services`. On Windows, it is the file `services` in the `Windows` directory.

See also

open-connection (page 309)
shutdown-socket-stream (page 311)

Example

Open a socket connection to the GBBopen Project web server:

```
> (with-open-connection (connection "GBBopen.org" 80)
    (flet ((write-crlf (stream)
             ;; HTTP requires CR/LF line termination:
             (write-char #\return stream)
             (write-char #\linefeed stream)))
      (format connection "GET / HTTP/1.1")
      (write-crlf connection)
      (format connection "Host: ~a:~a" host port)
      (write-crlf connection)
      (write-crlf connection)
      (force-output connection)
      (let ((line (read-line connection)))
        (format t "~&;; Received: ~a~%" line))))
;; Received: HTTP/1.1 200 OK
>
```

4.5 Double Metaphone

The `:double-metaphone` module provides Double Metaphone phonetic-code generation.

The Metaphone algorithm, published by Lawrence Philips in 1990, improved early phonetic approaches, such as Soundex, by attempting to correctly code cases where “gh” is pronounced as “f” as in “laugh” and when it is silent, as in “dough,” or where “t,” “c,” and “s” are pronounced as “sh” (or “ch,” which is treated as a sound that is similar enough to “sh” to be mapped to the same encoding value) as in “ratio,” “ciao,” and “erosion.” Although Metaphone was an improvement, it failed to encode many common words accurately, including the silent “l” in “lincoln,” and the case of “school” where “ch” is pronounced as “k.” Also, by using the Soundex style of encoding an initial vowel as it appears, Metaphone results in different encodings for “Otto” and “auto,” which sound similar enough to match.

Double Metaphone was published by Lawrence Philips in 2000 to improve accuracy further than was achieved by Metaphone. It maps all initial vowels to “A,” matching “Otto” to “auto.” It attempts to correctly encode a number of common words and names commonly found in the United States that are of non-english origin and are usually pronounced correctly according to their non-english spellings, by Americans, such as “Jose” or “pizza.” It also attempts to account for cases where more than one pronunciation may be common in the United States, such as the Spanish name “Cabrillo” which might be plausibly pronounced as “cabreeyo” or “cabrillo.” Double Metaphone addresses exceptions in regular English pronunciation, such as the many cases of silent consonants, such as the silent “l” in “lincoln” or the silent “s” in “island,” as well as the pronunciation of “s” as “sh” in “sugar,” or an anomaly like “caesar,” an unusual case of a “c” followed by an “a” where the “c” is pronounced as “s.”

Even with this more detailed treatment, Double Metaphone still misses a number of common exceptions, such as the silent “p” in “receipt,” many cases where “ch” is pronounced as “k” instead of “ch” as in “monarch,” many cases where Americans pronounce words of non-English origin according to their non-English pronunciations, such as “chutzpah,” and exceptions such as “colonel,” pronounced “kernal,” and “tucson,” pronounced “tooson.”

In addition to providing strict adherence to the Double Metaphone algorithm, this Common Lisp implementation also supports an extended-encoding option that activates additional encoding rules that separate the sounds used for “B” and “P,” for “D” and “T,” for “F” and “V,” and for “S” and “Z.” The extended-encoding option often produces phonetic-code results that are more natural and intuitive.

double-metaphone *string* `&optional` *extended-p* ⇒ *primary-index [, secondary-index]* [*Function*]

Purpose

Compute the primary and secondary Double Metaphone phonetic-code strings of *string*.

Package `:gbbopen-tools`

Module `:double-metaphone`

Arguments

string A string
extended-p A generalized boolean (default is `nil`)
primary-index A string
secondary-index A string

Returns

One or two values: the primary and secondary phonetic-code strings. If there is no secondary code for *string*, only the primary string is returned.

Description

If *extended-p* is `nil`, strict Double Metaphone encoding is used. If *extended-p* is true, additional encoding rules that separate sounds are used for "B" and "P," for "D" and "T," for "F" and "V," and for "S" and "Z."

Examples

```
> (double-metaphone "testing")
"TSTN"
> (double-metaphone "Smith")
"SM0"
"XMT"
> (double-metaphone "Schmidt")
"XMT"
"SMT"
> (double-metaphone "batboy")
"PTP"
> (double-metaphone "batboy" 't)
"BTB"
> (double-metaphone "Barlow")
"PRL"
"PRLF"
> (double-metaphone "Barlow" 't)
"BRL"
"BRLF"
> (double-metaphone "buzz")
"PS"
> (double-metaphone "buzz" 't)
"BZ"
>
```

4.6 OS Interface

The `:os-interface` module provides a uniform interface to commonly used operating-system entities.

browse-hyperdoc *symbol* ⇒ *boolean* [*Function*]

Purpose
Display the GBBopen Hyperdoc page for *symbol* in a browser window.

Package :gbbopen-tools

Module :os-interface

Arguments
symbol A symbol
boolean A generalized boolean

Returns
True if the Hyperdoc file associated with *symbol* is available and has been passed to the preferred browser; no value otherwise.

Description
The desired browser can be specified in `*preferred-browser*` (see the discussion in GBBopen hyperdoc (see page 7) for details).

See also
preferred-browser (page 11)

Example
```
> (browse-hyperdoc 'standard-event-instance)
t
>
```

close-external-program-stream *stream* [*Function*]

Purpose

Close a stream created by **run-external-program**.

Package :gbbopen-tools

Module :os-interface

Arguments

stream The stream to be closed

See also

run-external-program (page 321)

Example

```
> (let ((stream (run-external-program "date" nil)))
     (print (read-line stream))
     (close-external-program-stream stream))
"Mon Jul  4 14:06:04 EDT 2005"
>
```

kill-external-program *os-process* `&optional` *signal-number* [*Function*]

Purpose
Terminate or signal an external program.

Package `:gbbopen-tools`

Module `:os-interface`

Arguments
os-process An implementation-dependent process representation or `nil`
signal-number A small integer (default is `15`, the software termination signal)

Errors
This function is not supported on Windows platforms.

See also
run-external-program (page 321)

Example
```
> (multiple-value-bind (stream os-process)
        (run-external-program "sleep" '("120"))
      (sleep 10)
      (kill-external-program os-process))
t
>
```

run-external-program *program args* &key *input output wait* [*Function*]
⇒ *bidirectional-stream, os-process*

Purpose
Run an external program.

Package
`:gbbopen-tools`

Module
`:os-interface`

Arguments
program	A string specifying the name of the program to be run
args	A list of strings passed to *program* as arguments
input	A stream specification (default is `:stream`, see below)
output	A stream specification (default is `:stream`, see below)
wait	A generalized boolean (default is `nil`)
bidirectional-stream	A stream or `nil`
os-process	An implementation-dependent process representation or `nil`

Returns
Two values:

- an input, output, or bi-directional stream or `nil`
- an operating-system process representation, if available, or `nil`

Errors
Use of a true value for *wait* and a `:stream` value for *input* or *output* is problematic or an error in most Common Lisp implementations.

Description
The values of *input* and *output* can be:

- `:stream` (the default) which creates a stream that is returned as the first result value; if both *input* and *output* are specified as `:stream`, a bi-directional stream is created and returned
- a string specifying a file to be used as input or output

LispWorks (non-Windows platforms) and SBCL do not use a search path for locating *program*, the full path must be specified in the *program* string.

See also
close-external-program-stream (page 319)

Example
```
> (let ((stream (run-external-program "date" nil)))
     (print (read-line stream))
     (close-external-program-stream stream))
"Mon Jul  4 14:06:04 EDT 2005"
>
```

svn-version `&key` *directory program* ⇒ *version-string-or-nil* [*Function*]

Purpose

Obtain the Subversion compact version number of a working copy.

Package `:gbbopen-tools`

Module `:os-interface`

Arguments

directory One of the following:

- A string specifying a directory
- A pathname specifying a directory
- A keyword naming a root directory

(default is the GBBopen install directory)

program A string specifying the name of the program to be run (default is `"svnversion"`)

version-string-or-nil A string or `nil`

Returns

A string containing the compact version number or `nil` if the program `svnversion` cannot be found or the specified directory is not in a Subversion working copy.

Description

LispWorks (non-Windows platforms) and SBCL do not use a search path for locating *program*, the full path must be specified in the *program* string.

Examples

```
> (svn-version)
"525"
> (svn-version :program "/usr/bin/svnversion")
"525"
> (svn-version :directory ':my-app-root)
"73:76M"
>
```

5 GBBopen Core

The GBBopen Core module, `:gbbopen-core`, provides support for the blackboard repository, unit and space classes and instances, inter-instance links, and event signaling.

Documentation for unit-class and unit-instance entities, as well as general-purpose `:gbbopen-core` entities, is included in this section. Documentation for the remaining `:gbbopen-core` entities is arranged into the following sections:

- link entities (Section 5.1)
- event, event function, event printing, and event signaling entities (Section 5.2)
- interval manipulation entities (Section 5.3)
- space-instance and blackboard-repository entities (Section 5.4)
- instance retrieval and iteration/mapping entities (Section 5.5)
- saving/sending and loading/reading entities (Section 5.6)
- queue-management entities (Section 5.7)

skip-deleted-unit-instance-class-change [*Variable*]

Purpose

Controls whether the class of a unit instance is changed to a **deleted-unit-instance** when it is deleted.

Package

`:gbbopen`

Module

`:gbbopen-core`

Value type

A generalized boolean

Initial value

`nil`

Description

When ***skip-deleted-unit-instance-class-change*** is `nil`, the class of a unit instance is changed to a **deleted-unit-instance** when it is deleted. This helps identify a deleted unit instance that is used inadvertently, at the minor additional cost of the class change when the instance is deleted. In situations where unit instances are created and deleted at a high rate, the class change can be skipped by binding ***skip-deleted-unit-instance-class-change*** to a non-`nil` value. The predicate **instance-deleted-p** can be used to detect either form of a deleted unit instance.

See also

check-for-deleted-instance (page 327)
delete-instance (page 334)
deleted-instance-class (page 336)
deleted-unit-instance (page 337)
instance-deleted-p (page 353)

Example

Create and delete a `hyp` unit instance, first with the default changing-class behavior of **delete-instance** and then with the class-change skipped:

```
> (delete-instance (make-instance 'hyp))
#<deleted-unit-instance hyp 1>
> (instance-deleted-p *)
t
> (let ((*skip-deleted-unit-instance-class-change* 't))
    (delete-instance (make-instance 'hyp)))
#<hyp [Deleted] 2>
> (instance-deleted-p *)
t
>
```

change-class *instance new-class* `&key &allow-other-keys` ⇒ *changed-instance* [*Generic Function*]

Purpose

Change the class of an instance to *new-class*.

Method signatures

change-class : `around (`***instance*** `standard-object) (`***new-class*** `standard-unit-class) &key &allow-other-keys` ⇒ ***instance***

change-class : `around (`***instance*** `standard-object) (`***new-class*** `standard-unit-class) &key &allow-other-keys` ⇒ ***instance***

change-class : `before (`***instance*** `standard-unit-instance) (`***new-class*** `standard-class) &key &allow-other-keys`

change-class : `after (`***instance*** `standard-unit-instance) (`***new-class*** `standard-unit-class) &key &allow-other-keys`

change-class : `after (`***instance*** `standard-object) (`***new-class*** `standard-unit-class) &key &allow-other-keys`

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

instance	A `standard-object` instance
new-class	A class designator
changed-instance	A `standard-object` instance

Returns

The destructively modified *instance*.

Events

If *instance* is a unit instance, a `change-instance-class-event` is signaled at the start of the class-change process. If *new-class* is a unit class, an `instance-changed-class-event` is signaled when the class change has been completed.

The following events may also be signaled:

- `unlink-event`
- `instance-removed-from-space-instance-event`
- `link-event`
- `nonlink-slot-updated-event`
- `instance-added-to-space-instance-event`

Errors

The existing or supplied instance name of *instance* is identical to the instance name of an existing unit instance of *new-class*.

Description

When *new-class* is a unit class, an instance-name conflict with an existing unit instance of *new-class* must be avoided. If the old class of *instance* and *new-class* are both unit classes that use the global instance-name counter, the old instance-name value can be retained safely. If *new-class* is a unit class with a class-based counter, specifying a new instance-name value (using **next-class-instance-number**) is recommended.

When the old class and *new-class* are both unit classes and no space-instances value is supplied to **change-class**, the changed *instance* remains on all space instances that allow *new-class* unit instances. In addition, *instance* is also added to all space instances defined as initial-space-instances for *new-class*. If a space-instances value is supplied, *instance* is removed from all space instances and then added to the supplied space instances.

See also

define-unit-class (page 330)
make-duplicate-instance-changing-class (page 361)
next-class-instance-number (page 366)

Example

Change the class of unit instance `hyp` from `probable-hyp` to `rejected-hyp`:

```
> (change-class hyp 'rejected-hyp
    :instance-name (next-class-instance-number 'rejected-hyp))
#<rejected-hyp 1409 (896 388) .68>
>
```

Note that the `:instance-name` initarg can be eliminated if both the old and new classes for *hyp* are using the global instance-name counter.

---

change-class

check-for-deleted-instance *unit-instance* `&optional` *operation* [*Function*]

Purpose
Signal an error if the supplied unit instance has been deleted.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
unit-instance A unit instance
operation A symbol (default is `nil`)

See also
delete-instance (page 334)
instance-deleted-p (page 353)

Examples
Create, then delete, then check, a `hyp` unit instance:

```
  > (check-for-deleted-instance (delete-instance (make-instance 'hyp)))
  Error: Instance #<deleted-unit-instance hyp 7> has been deleted
  >> :abort
  > (check-for-deleted-instance (delete-instance (make-instance 'hyp))
                                'my-operation)
  Error: my-operation attempted with a deleted instance:
#<deleted-unit-instance hyp 8>
  >>
```

check-instance-locators *unit-instance* [*Function*]

Purpose
Signal an error if the locators on any space-instance are inconsistent with the dimension values of the supplied unit instance.

Package :gbbopen

Module :gbbopen-core

Arguments
unit-instance A unit instance

Description
Changes to slot values that affect a unit instance's dimension values must be indicated to GBBopen using the **with-changing-dimension-values** macro. Otherwise, locators for the unit instance will not be updated to reflect the new dimension values, resulting in the inability to retrieve the instance where it should be located on a space instance and removed or deleted instances incorrectly retained on a space instance due to the inconsistent locators. **Check-instance-locators** can be used as a debugging aid during development to check for inconsistent locators.

See also
with-changing-dimension-values (page 372)

Examples
Check the locators of all unit instances that are stored on any space instance:

```
> (map-instances-on-space-instances #'check-instance-locators t t)
nil
>
```

Intentionally create inconsistent locators for a `hyp` unit instance by changing its `location` without using **with-changing-dimension-values**. Then check its locators:

```
> (defparameter *hyp* (find-instance-by-name 419 'hyp))
*hyp*
> *hyp*
#<hyp 419 (1835 4791) 0.85 [5..35]>
>(check-instance-locators *hyp*)
nil
> (setf (location-of *hyp*) '(2000 2000))
(2000 2000)
> (check-instance-locators *hyp*)
Error: Instance #<hyp 419 (2000 2000) 0.85 [5..35]> is missing (bucket
[21,21])
        in #<2d-Uniform-Buckets (bb hyps) (hyp+) (x y) 8>
>>
```

class-instances-count *unit-class-or-name* ⇒ *count* [*Generic Function*]

Purpose
Obtain the current count of unit instances of a unit class.

Method signatures
class-instances-count (*unit-class-name* `symbol`) ⇒ *count*
class-instances-count (*unit-class-specifier* `cons`) ⇒ *count*
class-instances-count (*unit-class* `standard-unit-class`) ⇒ *count*

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-class-or-name A unit class or a symbol naming a unit class
count An integer

Returns
The count of unit instances of the specified unit class. If an extended unit-classes specification is supplied, the sum of the unit instance counts of the specified classes is returned.

See also
next-class-instance-number (page 366)

Examples
Return the count of unit instances of `standard-space-instance`:

```
> (class-instances-count 'standard-space-instance)
8
>
```

Return the count of all space instance:

```
> (class-instances-count '(standard-space-instance :plus-subclasses))
14
>
```

or simply:

```
> (class-instances-count '(standard-space-instance +))
14
>
```

define-unit-class *unit-class-name* (*{superclass-name}**) (*{slot-specifier}**) *{class-option}** [*Macro*]
⇒ *new-unit-class*

Purpose
Define or redefine a unit class.

Package :gbbopen

Module :gbbopen-core

Arguments
unit-class-name	A non-`nil`, non-keyword symbol that names the unit class
superclass-name	A non-`nil`, non-keyword symbol that specifies a direct superclass of the unit class *unit-class-name*
slot-specifiers	See below
class-options	See below
new-unit-class	A new or modified unit class object

Returns
The newly defined or modified unit class object.

Errors
The specified *superclass-names* do not include at least one unit class name. This error is signaled on class finalization.

Detailed syntax
(Syntax shown in gray is not supported in GBBopen Version 1.5, but will become available in a future release.)

slot-specifier ::= *slot-name* |
(*nonlink-slot-name* [[*nonlink-slot-option*]]) |
(*link-slot-name* [[*link-slot-option*]])
nonlink-slot-name ::= *slot-name*
link-slot-name ::= *slot-name*
link-slot-option ::= *slot-option* |
{:link *inverse-link-slot-specifier*} |
{:singular *boolean*} |
{:sort-function *function*} |
{:sort-key *function*}
inverse-link-slot-specifier ::= (*unit-class-name link-slot-name* [:singular *boolean*]) |
:reflexive
nonlink-slot-option ::= *slot-option* |
{:reader *reader-function-name*}* |
{:writer *writer-function-name*}*
slot-option ::= {:accessor *reader-function-name*}* |
{:allocation *allocation-type*} |
{:documentation *string*} |
{:initarg *initarg-name*}* |
{:initform *form*} |
{:type *type-specifier*}

class-option ::= (:abstract *boolean*) |
(:default-initargs . *initarg-list*) |
(:dimensional-values *dimension-value-specifier**) |
(:documentation *string*) |
(:estimated-instances *size-form*) |
(:export-accessors *boolean*) |
(:export-class-name *boolean*) |
(:export-slot-names *direct-slots-specifier*) |
(:generate-accessors *direct-slots-specifier*) |
(:generate-accessors-format {:prefix | :suffix} |
(:generate-accessors-prefix {*string* | *symbol*}) |
(:generate-accessors-suffix {*string* | *symbol*}) |
(:generate-initargs *direct-slots-specifier*) |
(:initial-space-instances *initial-space-instance-specifier*) |
(:instance-name-comparison-test *instance-name-comparison-test*) |
(:metaclass *class-name*) |
(:retain {*boolean* | :propagate}) |
(:use-global-instance-name-counter *boolean*)
initial-space-instance-specifier ::= {*space-instance-path*+ | *function*}
dimension-value-specifier ::= *incomposite-dv-specifier* | *composite-dv-specifier*
incomposite-dv-specifier ::= (*dimension-name dimension-value-spec dimension-value-place*)
composite-dv-specifier ::= (*dimension-name dimension-value-specifier*
composite-type dimension-value-place)
composite-type ::= :set | :sequence |
{:ascending-series *ordering-dimension-name*} |
{:descending-series *ordering-dimension-name*}
dimension-value-specifier ::= *dimension-value-type* |
(*ordered-dimension-value-type* [*ordered-comparison-type*]) |
(*enumerated-dimension-value-type* [*enumerated-comparison-type*]) |
(*boolean-dimension-value-type* [*boolean-comparison-type*])
dimension-value-type ::= *ordered-dimension-value-type* |
enumerated-dimension-value-type |
boolean-dimension-value-type
ordered-dimension-value-type ::= :point | :interval | :mixed
enumerated-dimension-value-type ::= :element
boolean-dimension-value-type ::= :boolean
ordered-comparison-type ::= number | fixnum | short-float | single-float |
double-float | long-float |
pseudo-probability
enumerated-comparison-type ::= eq | eql | equal | equalp
boolean-comparison-type ::= t
dimension-value-place ::= {*slot-name* [*slot-name*]} | {*function* [*slot-name*]}
direct-slots-specifier ::= nil | t | *included-slot-name** |
{t :exclude *excluded-slot-name**}

The default *ordered-comparison-type*, if unspecified, is number. The default *enumerated-comparison-type*, if unspecified, is eql. The default *boolean-comparison-type* is t.

A *dimension-value-place* with two *slot-names* is allowed only for an :interval dimension-value specification.

Terms

class-name	A non-`nil`, non-keyword symbol that names a class
dimension-name	A symbol specifying a dimension
documentation	A documentation string
initarg-list	An initialization argument list
instance-name-comparison-test	One of the four standardized hash table test function names: `eq`, `eql`, `equal`, or `equalp` (default for classes of metaclass **standard-unit-class** is `eql`)
ordering-dimension-name	A symbol specifying the ordering dimension used to order series-composite dimension values
size-form	An integer or an expression that evaluates to an integer (evaluation occurs whenever the unit class is initialized, reinitialized, or reset)
slot-name	A non-`nil`, non-keyword symbol

Description

A *dimension-value-place* with two *slot-names* can be specified only for `:interval` dimension-value types.

If *dimension-value-place* is specified as a *function* without a qualifying *slot-name*, *function* is called with the unit instance rather than a slot value. In this case, *function* is responsible for handling any unbound slots that it references, returning **unbound-value-indicator** when appropriate.

Each *superclass-name* argument specifies a direct superclass of the new class. If the superclass list is empty, then the direct superclass defaults to the single class **standard-unit-instance**.

The `:metaclass` class option, if specified, must be a subclass of **standard-unit-class**. The default metaclass value is **standard-unit-class**.

Inheritance of class options

The set of *dimensional-values* for a unit class is the union of the sets specified in the *dimensional-values* options of the class and its superclasses. When more than one dimension-value specification is supplied for a given dimension, the one supplied by the most specific class is used.

The effective *initial-space-instances* value for a unit class is the value specified in the definition of the most specific unit class. (No additive inheritance of *initial-space-instances* is performed.) If no definitions specify an *initial-space-instances* value, `nil` is used.

The *instance-name-comparison-test* value is not inherited. If no value is specified in the unit-class definition, the default initialization value associated with the metaclass is used.

If a *retain* value is not specified, a value of `:propagate` is used as the default if any parent unit classes have a `:propagate` retention value; otherwise `nil` is used as the default value.

The *use-global-instance-name-counter* value is not inherited. If no value is specified in the unit-class definition, the default initialization value associated with the metaclass is used.

See also

define-space-class (page 438)
define-class (page 83)
delete-blackboard-repository (page 442)
deleted-unit-instance (page 337)
find-all-instances-by-name (page 479)
find-instance-by-name (page 481)

link-slot-p (page 387)
make-instance (page 364)
standard-unit-class (page 369)
standard-unit-instance (page 370)
with-generate-accessors-format (page 136)

Examples

Define a unit class, `hyp`, that illustrates a number of **define-unit-class** capabilities:

```
> (define-unit-class hyp ()
    ((belief :initform 0.0)
     (location :initform nil)
     (velocity-range)
     (color)
     (classification :initform '(:car :truck :bus :motorcycle :train
:duck-boat
                                 :lawn-mower :anything))
     (supporting-hyps
      :link (hyp supported-hyps))
     (supported-hyps
      :link (hyp supporting-hyps)))
    (:dimensional-values
     (belief :point belief)
     (velocity-range :interval velocity-range)
     (color (:element eq) color)
     (classification (:element eq) :set classification)
     (x (:point fixnum) #'location.x location)
     (y (:point fixnum) #'location.y location))
    (:initial-space-instances (bb hyps)))
#<standard-unit-class hyp>
>
```

Define a unit class, `word`, whose instances are indexed by the word's individual characters in an enumerated dimension and the **code-char** values of the individual characters in an ordered dimension:

```
> (define-unit-class word ()
    ((string :initform "what's"))
    (:dimensional-values
     (character :element :set string)
     (char-code (:point fixnum) :set
                #'(lambda (string)
                    (map 'list #'char-code string))
                string))
    (:initial-space-instances (words)))
#<standard-unit-class word>
>
```

delete-instance *unit-instance* ⇒ *deleted-unit-instance* [*Generic Function*]

Purpose
Delete a unit instance.

Method signatures
delete-instance (*unit-instance* `deleted-unit-instance`) ⇒ *deleted-unit-instance*
delete-instance (*unit-instance* `standard-unit-instance`) ⇒ *deleted-unit-instance*
delete-instance (*space-instance* `standard-space-instance`) ⇒ *deleted-unit-instance*

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
unit-instance The unit instance (or space instance) to be deleted
deleted-unit-instance The deleted *unit-instance* object, whose class has been changed to a **deleted-unit-instance** unless ***skip-deleted-unit-instance-class-change*** is true

Returns
The deleted instance *deleted-unit-instance*.

Events
A `delete-instance-event` is signaled at the start of the deletion process and an `instance-deleted-event` is signaled when the deletion has been completed. The following events may also be signaled:

- `unlink-event`
- `instance-removed-from-space-instance-event`

Description
If ***skip-deleted-unit-instance-class-change*** is `nil`, the generic function **deleted-instance-class** is called by **delete-instance** to determine the class to be used as the changed class for the *deleted-unit-instance*.

See also

skip-deleted-unit-instance-class-change (page 324)
delete-all-space-instances (page 444)
delete-blackboard-repository (page 442)
delete-space-instance (page 445)
deleted-instance-class (page 336)
deleted-unit-instance (page 337)
make-instance (page 364)
reset-gbbopen (page 461)
reset-unit-class (page 367)

Example

Create, then delete, a `hyp` unit instance:

```
> (delete-instance (make-instance 'hyp :location '(896 388) :belief .68))
#<deleted-unit-instance hyp 311>
>
```

---

delete-instance

deleted-instance-class *unit-instance* ⇒ *class* [*Generic Function*]

Purpose

Return the class to be used for a deleted unit instance.

Method signatures

deleted-instance-class (*unit-instance* `standard-unit-instance`) ⇒
`#<standard-class deleted-unit-instance>`

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

unit-instance The unit instance (or space instance) to be deleted

class A class or a non-`nil`, non-keyword symbol that names a class (the default method returns the class **deleted-unit-instance**)

Returns

The class or symbol naming the class to be used as the changed class for the deleted instance.

Description

This generic function is called by **delete-instance** to determine the class to be used as the changed class for a deleted unit instance. The returned *class* must be a subclass of **deleted-unit-instance**, but it must not be a subclass of **standard-unit-instance**.

See also

delete-instance (page 334)
deleted-unit-instance (page 337)

Example

Define a class to be used for deleted `hyp` unit instances and a **deleted-instance-class** method to use `deleted-hyp` as the class for deleted `hyp` (and subclasses of `hyp`) unit instances:

```
> (define-class deleted-hyp (deleted-unit-instance)
    (location
     classification
     supporting-hyps)
#<standard-class deleted-hyp>
> (defmethod deleted-instance-class ((hyp hyp))
    (load-time-value (find-class 'deleted-hyp)))
deleted-instance-class
>
```

deleted-unit-instance [*Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **deleted-unit-instance** is an instance of **standard-class** and an instance of **deleted-unit-instance** represents a deleted unit instance or space instance. It is a subclass of **standard-gbbopen-instance**.

See also

delete-instance (page 334)
standard-gbbopen-instance (page 126)
standard-space-instance (page 464)
standard-unit-instance (page 370)

describe-instance *unit-instance* [*Generic Function*]

Purpose
Describe a unit instance (or a space instance, as a unit instance).

Method signatures
describe-instance `(`*unit-instance* `standard-unit-instance) &optional` *stream*

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
unit-instance A unit instance (or space instance)
stream A stream (default is `*standard-output*`)

Description
The description is printed to the output *stream*.

See also

describe-instance-slot-value (page 340)
describe-space-instance (page 448)
describe-space-instance-storage (page 449)
make-instance (page 364)
make-space-instance (page 455)

Example
Describe the `hyp` unit instance:

```
> (describe-instance hyp)
Hyp #<hyp 419 (1835 4791) 0.85 [5..35]>
  Instance name: 419
  Space instances: ((bb hyps))
  Dimensional values:
    belief:  0.85
    classification:  (:car :truck)
    color:  :red
    velocity-range:  (5 35)
    x:  1835
    y:  4791
  Non-link slots:
    belief:  0.85
    classification:  (:car :truck)
    color:  :red
    location:  (1835 4791)
    velocity-range:  (5 35)
  Link slots:
    supported-hyps:  nil
```

```
    supporting-hyps:  (#<hyp 183 (1835 4791) 0.82 [0..35]>
                       #<hyp 233 (1835 4791) 0.89 [5..35]>)
  Space instances: (#<standard-space-instance (bb hyps)>)
>
```

REPL Note

Describe-instance can be invoked using the REPL command:

```
:di instance | {instance-name [unit-classes-specifier]}
```

which also sets = to the described unit instance.

describe-instance-slot-value *unit-instance slot-name value* [*Generic Function*]
`&optional` ***stream***

Purpose
Customize slot-value printing by **describe-instance**.

Method signatures
describe-instance-slot-value (***unit-instance*** `standard-unit-instance`) ***slot-name value***
`&optional` ***stream***

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-instance	A unit instance (or space instance)
slot-name	A non-`nil`, non-keyword symbol
value	An object
stream	A stream (default is `*standard-output*`)

Description
Describe-instance-slot-value is called by **describe-instance** to print each slot value; it should not be called directly. The slot-value representation is printed to the output *stream*.

See also
describe-instance (page 338)

Example
Describe the `observation-duration` slot of a `hyp` unit instance as a human-readable duration:

```
(defmethod describe-instance-slot-value
    ((hyp hyp)
     (slot-name (eql 'observation-duration)) value
     &optional (stream *standard-output*))
  (if value
      (pretty-duration duration 5 stream)
      (prin1 nil stream)))
```

describe-unit-class *unit-class-name* [*Generic Function*]

Purpose
Print information about a unit class.

Method signatures
describe-unit-class (*unit-class-name* `symbol`)
describe-unit-class (*unit-class-specifier* `cons`)
describe-unit-class (*unit-class* `standard-unit-class`)

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
unit-class-name A unit-class or an extended unit-classes specification (see below)

Detailed syntax
unit-classes-specifier ::= `t` | *single-unit-class-specifier* | (*single-unit-class-specifier*+)
single-unit-class-specifier ::= *atomic-unit-class* | (*atomic-unit-class subclassing-specifier*)
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
The description is printed to the ***standard-output*** stream.

Example
```
> (describe-unit-class 'hyp)
Standard-unit-class #<standard-unit-class hyp>
  Direct superclasses:
    standard-unit-instance (abstract)
  Direct subclasses: None
  Direct nonlink slots:
    belief
      :allocation :instance
      :initargs (:belief)
      :initform 0.0
      :readers (belief-of)
      :writers ((setf belief-of))
    classification
      :allocation :instance
      :initargs (:classification)
      :initform '(:car :truck :bus :motorcycle :train :duck-boat :lawn-mower
                  :anything)
      :readers (classification-of)
     :writers ((setf classification-of))
    color
```

```
    :allocation :instance
    :initargs (:color)
    :readers (color-of)
    :writers ((setf color-of))
  location
    :allocation :instance
    :initargs (:location)
    :initform nil
    :readers (location-of)
    :writers ((setf location-of))
  velocity-range
    :allocation :instance
    :initargs (:velocity-range)
    :readers (velocity-range-of)
    :writers ((setf velocity-range-of))
Direct link slots:
  supported-hyps
    :allocation :instance
    :initargs (:supported-hyps)
    :initform nil
    :readers (supported-hyps-of)
    :writers ((setf supported-hyps-of))
    :link (hyp supporting-hyps)
  supporting-hyps
    :allocation :instance
    :initargs (:supporting-hyps)
    :initform nil
    :readers (supporting-hyps-of)
    :writers ((setf supporting-hyps-of))
    :link (hyp supported-hyps)
Effective nonlink slots:
  belief
    :allocation :instance
    :initargs (:belief)
    :initform 0.0
  classification
    :allocation :instance
    :initargs (:classification)
    :initform '(:car :truck :bus :motorcycle :train :duck-boat :lawn-mower
                :anything)
  color
    :allocation :instance
    :initargs (:color)
  instance-name
    :allocation :instance
    :initargs (:instance-name)
  location
    :allocation :instance
    :initargs (:location)
    :initform nil
  velocity-range
    :allocation :instance
```

```
      :initargs (:velocity-range)
  Effective link slots:
    supported-hyps
      :allocation :instance
      :initargs (:supported-hyps)
      :initform nil
    supporting-hyps
      :allocation :instance
      :initargs (:supporting-hyps)
      :initform nil
  Dimensional values:
    belief (:point number)
    classification (:element eq) :set
    color (:element eq)
    velocity-range (:interval number)
    x (:point fixnum)
    y (:point fixnum)
  Effective dimensional values:
    belief (:point number)
    classification (:element eq) :set
    color (:element eq)
    velocity-range (:interval number)
    x (:point fixnum)
    y (:point fixnum)
  Initial space instances:
    (bb hyps)
  Effective initial space instances:
    (bb hyps)
  Retain: nil
>
```

---

describe-unit-class

dimensions-of *space-instance-or-unit-classes-specifier* [*Generic Function*]
⇒ *dimension-specifications-list*

Purpose
Return the dimension specifications of a space instance or one or more unit classes.

Method signatures
dimensions-of (*space-instance* `standard-space-instance`) ⇒ *dimension-specifications-list*
dimensions-of (*unit-classes-specifier* `cons`) ⇒ *dimension-specifications-list*
dimensions-of (*unit-class* `standard-unit-class`) ⇒ *dimension-specifications-list*

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
space-instance-or-unit-classes-specifier	A space instance, an extended unit-classes specification or a list of extended unit-classes specifications
dimension-specifications-list	A proper list

Returns
A list of `(dimension-name (dimension-type comparison-type))` pairs.

See also
define-unit-class (page 330)
make-space-instance (page 455)
instance-dimension-value (page 354)
instance-dimension-values (page 356)

Examples
Return the dimensions defined for instances of unit class `hyp`:

```
> (dimensions-of 'hyp)
((x (:ordered fixnum)) (y (:ordered fixnum)) (belief (:ordered number))
 (velocity-range (:ordered number)) (color (:enumerated eq))
 (classification (:enumerated eq)))
>
```

Return the dimensions of the `(bb hyps)` space instance:

```
> (dimensions-of (find-space-instance-by-path '(bb hyps)))
((x (:ordered fixnum)) (y (:ordered fixnum)) (belief (:ordered number))
 (velocity-range (:ordered number)) (color (:enumerated eq))
 (classification (:enumerated eq)))
>
```

Note that the dimensions of a space instance (its dimensional extent as a container for other unit instances) is independent of any dimension values that the space instance might have as a unit instance:

```
  > (instance-dimension-values (class-of (find-space-instance-by-path '(bb
hyps))))
  nil
  >
```

or the dimensions that the unit class of the space instance might have:

```
  > (dimensions-of (class-of (find-space-instance-by-path '(bb hyps))))
  nil
  >
```

Note

The returned list of dimension specifications for a space instance or for a unit class should not be destructively altered.

---

dimensions-of

direct-nonlink-slot-definition [*Metaobject Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **direct-nonlink-slot-definition** is the default direct nonlink-slot-definition metaobject class of unit classes created by **define-unit-class**. **Direct-nonlink-slot-definition** is a subclass of **gbbopen-direct-slot-definition**.

See also

direct-link-definition (page 379)
effective-nonlink-slot-definition (page 347)
gbbopen-direct-slot-definition (page 348)

effective-nonlink-slot-definition [*Metaobject Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **effective-nonlink-slot-definition** is the default effective nonlink-slot-definition metaobject class of unit classes created by **define-unit-class**. **Effective-nonlink-slot-definition** is a subclass of **gbbopen-effective-slot-definition**.

See also

direct-nonlink-slot-definition (page 346)
effective-link-definition (page 380)
gbbopen-effective-slot-definition (page 349)

gbbopen-direct-slot-definition [*Metaobject Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **gbbopen-direct-slot-definition** is the parent class of **direct-link-definition** and **direct-nonlink-slot-definition**.

See also

direct-link-definition (page 379)
direct-nonlink-slot-definition (page 346)
gbbopen-effective-slot-definition (page 349)

gbbopen-effective-slot-definition [*Metaobject Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **gbbopen-effective-slot-definition** is the parent class of **effective-link-definition** and **effective-nonlink-slot-definition**.

See also

effective-link-definition (page 380)
effective-nonlink-slot-definition (page 347)
gbbopen-direct-slot-definition (page 348)

gbbopen-implementation-version <no arguments> ⇒ *string* [*Function*]

Purpose
Return the GBBopen implementation version.

Package :gbbopen

Module :gbbopen-core

Arguments
string A string

Returns
The GBBopen implementation-version string

Example
Return the GBBopen implementation-version string:

```
> (gbbopen-implementation-version)
"1.0"
>
```

incomplete-instance-p *unit-instance* ⇒ *boolean* [*Function*]

Purpose
Determine if a unit instance is incomplete.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-instance A unit instance
boolean A generalized boolean

Returns
True if the unit instance is incomplete; `nil` otherwise.

See also
instance-deleted-p (page 353)

Example
Define an event function that recomputes the area of a complete unit instance when the value of its `height` or `width` slot changes:

```
(defun update-area-evfn (event-name
                         &key instance
                         &allow-other-keys)
  (declare (ignore event-name))
  ;; Do not compute the area of an incomplete instance:
  (unless (incomplete-instance instance)
    (setf (area-of instance)
          (* (width-of instance) (height-of instance)))))

(add-event-function 'update-area-evfn
                    'rectangle
                    :slot-names '(width height))
```

initial-class-instance-number *unit-class-name-or-instance* ⇒ *integer* [*Generic Function*]

Purpose
Return the initial instance-name counter value associated with a unit class.

Method signatures
initial-class-instance-number (*unit-class-name* `symbol`) ⇒ *integer*
initial-class-instance-number (*unit-instance* `standard-unit-instance`) ⇒ `0`

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-class-name-or-instance A unit instance or a symbol naming a unit class
integer An integer

Returns
The initial instance-name counter value associated with the unit class.

See also
next-class-instance-number (page 366)
make-instance (page 364)

Examples
Return the initial instance-name value associated with unit class `ksa`:

```
> (initial-class-instance-number 'ksa)
0
>
```

Note
The initial instance-name value associated with a unit class is not used if the `:use-global-instance-name-counter` class option is true for the unit class.

instance-deleted-p *unit-instance* ⇒ *boolean* [*Function*]

Purpose
Determine if a unit instance has been deleted.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-instance A unit instance
boolean A generalized boolean

Returns
True if the unit instance is deleted; `nil` otherwise.

See also
skip-deleted-unit-instance-class-change (page 324)
check-for-deleted-instance (page 327)
delete-instance (page 334)
incomplete-instance-p (page 351)

Example
Create, then delete, then check, a `hyp` unit instance:

```
> (instance-deleted-p (delete-instance (make-instance 'hyp)))
t
>
```

instance-dimension-value *unit-instance dimension-name* [*Function*]
⇒ *dimension-value, dimension-value-type,* comparison-type, composite-type, | ordering-

Purpose
Obtain a dimension value of a unit instance.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-instance A unit instance
dimension-name A symbol specifying a dimension of *unit-instance*
dimension-value An object
dimension-value-type One of: `:point`, `:interval`, `:mixed`, `:element`, or `:boolean`
comparison-type One of: `number`, `fixnum`, `short-float`, `single-float`, `double-float`, `long-float`, or `pseudo-probability`
composite-type One of `:set`, `:sequence`, (`:ascending-series` *ordering-dimension-name*), (`:descending-series` *ordering-dimension-name*), or `nil`
ordering-dimension-name A non-keyword symbol or `nil`

Returns
Five values:

- the dimension value of the *unit-instance* in the specified dimension
- the dimension-value type of the specified dimension
- the comparison type of the specified dimension
- the composite type of the dimension value if it is a composite dimension value; `nil` if it is an incomposite dimension value
- the name of the ordering dimension, if the dimension value is a series-composite dimension value; `nil` otherwise

Errors
The dimension *dimension-name* is not defined for *unit-instance*.

See also
define-unit-class (page 330)
dimensions-of (page 344)
instance-dimension-values (page 356)
make-instance (page 364)

Examples
Return the `x` dimension value of the unit instance, `hyp`:

```
> (instance-dimension-value hyp 'x)
1835
:point
:fixnum
```

```
nil
nil
>
```

and its `classification` dimension value:

```
> (instance-dimension-value hyp 'classification)
(:car :truck)
:element
eq
:set
nil
>
```

Return the `character` dimension value of the unit instance, `word`:

```
> (instance-dimension-value w 'character)
"what's"
:element
eql
:set
nil
>
```

and its `char-code` dimension value:

```
> (instance-dimension-value word 'char-code)
(119 104 97 116 39 115)
:point
fixnum
:set
nil
>
```

instance-dimension-value

instance-dimension-values *unit-instance* `&optional` *dimension-names* [*Function*]
⇒ *dimension-values-alist*

Purpose
Return dimension values of a unit instance.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
unit-instance	A unit instance
dimension-names	A list of symbols naming dimensions of *unit-instance* or `t`, indicating all dimensions of *unit-instance* (default is `t`)
dimension-values-alist	An association list

Returns
An association list of dimension name and dimension value pairs.

Errors
A dimension name specified in *dimension-names* is not defined for *unit-instance*.

See also
define-unit-class (page 330)
dimensions-of (page 344)
instance-dimension-value (page 354)
make-instance (page 364)

Examples
Return all dimension values of the unit instance, `hyp`:
```
> (instance-dimension-values hyp)
((belief . 0.85) (velocity-range 5 35) (color . :red)
 (classification :car :truck) (x . 1835) (y . 4791))
>
```

Return the `x` and `y` dimension values of the unit instance, `hyp`:
```
> (instance-dimension-values hyp '(x y))
((x . 1835) (y . 4791))
>
```

Return all dimension values of the unit instance, `word`:
```
> (instance-dimension-values word)
((character . "what's") (char-code 119 104 97 116 39 115))
>
```

Note
Using **instance-dimension-value** is preferable to using **instance-dimension-values** when obtaining individual dimension values of a unit instance.

instance-name-of *unit-instance* ⇒ *instance-name* [*Generic Accessor*]

Purpose
Return the instance name of a unit instance.

Setf syntax
(setf (instance-name-of *instance*) *instance-name*) ⇒ *instance-name*

Method signatures
instance-name-of (*unit-instance* `standard-event-instance`) ⇒ *instance-name*
instance-name-of (*unit-instance* `standard-unit-instance`) ⇒ *instance-name*

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
unit-instance A unit instance or an event instance
instance-name An object

Returns
The instance name of the *unit-instance*.

Errors
The supplied *instance-name* provided to **(setf instance-name-of)** is identical to the instance name of an existing unit instance of the same class as *unit-instance*.

See also
make-instance (page 364)

Examples
Return the instance names of the unit instances supporting `hyp` unit instance `180`:

```
> (mapcar #'instance-name-of
          (supporting-hyps-of (find-instance-by-name 'hyp 180)))
(123 158 94)
>
```

Change the instance name of `hyp 180` to `"bogus-180"`:

```
> (setf (instance-name-of (find-instance-by-name 180 'hyp)) "bogus-180")
"bogus-180"
>
```

make-duplicate-instance *instance unduplicated-slot-names* `&rest` *initargs* `&key` `&allow-other-keys` ⇒ *new-instance, slots* [*Generic Function*]

Purpose

Create a duplicate instance of *instance*.

Method signatures

make-duplicate-instance (*instance* `standard-object`) *unduplicated-slot-names* `&rest` *initargs* ⇒ *new-instance, slots*

make-duplicate-instance (*instance* `standard-space-instance`) *unduplicated-slot-names* `&rest` *initargs* ⇒ *new-instance, slots*

make-duplicate-instance (*instance* `standard-unit-instance`) *unduplicated-slot-names* `&rest` *initargs* ⇒ *new-instance, slots*

make-duplicate-instance `:around` (*instance* `standard-unit-instance`) *unduplicated-slot-names* `&rest` *initargs* ⇒ *new-instance, slots*

Package

`:gbbopen-tools` (re-exported by `:gbbopen`)

Module

`:gbbopen-tools` (the unit-instance and space-instance methods are added by `:gbbopen-core`)

Arguments

instance	A `standard-object` instance
unduplicated-slot-names	A list of slot names
initargs	An initialization argument list
new-instance	A `standard-object` instance
slots	A proper list of slot objects

Returns

Two values: the (newly created) duplicate instance and a list of the slots that were duplicated or explicitly initialized.

Events

When a unit instance is duplicated, events are signaled in the following sequence:

1. An `nonlink-slot-updated-event` or `link-event` is signaled for each initialized slot in the duplicated unit instance. A `link-event` is also signaled for each inverse pointer from an existing unit instance to the (newly created) duplicate unit instance.
2. An `instance-added-to-space-instance-event` is signaled for each space instance on which the duplicate unit instance is added.
3. A `instance-created-event` is signaled.

Errors

Use of an initialization argument that has not been declared as valid.

Description

Slot initialization during instance duplication behaves as follows, regardless of whether the slots are local or shared:

- If an initarg in *initargs* specifies a value for a slot, that value is stored into the slot in the duplicated instance.
- All slots in the duplicated instance that are unbound at this point and that are not named in *unduplicated-slot-names* or in the list of slot-names returned from calling **unduplicated-slot-names** on *instance* are set to the value of the slot in *instance*.
- Any slots that remain unbound and are named in *unduplicated-slot-names* or in the list of slot-names returned from calling **unduplicated-slot-names** on *instance* are initialized according to their `:initform` forms.

When duplicating a unit instance, specifying a `:instance-name` initialization argument causes that value to be used as the instance name of the newly created unit instance instead of the instance-name counter value associated with the unit class (if the `:use-global-instance-name-counter` class option is `nil` or was not specified for the unit class) or the global instance-name counter value (if the `:use-global-instance-name-counter` class option is true for the unit class).

A `:space-instances` initialization argument must be provided when duplicating a space instance.

See also

change-class (page 325)
delete-instance (page 334)
describe-instance (page 338)
initial-class-instance-number (page 352)
instance-name-of (page 357)
make-duplicate-instance-changing-class (page 361)
make-instance (page 364)
make-space-instance (page 455)
next-class-instance-number (page 366)
unduplicated-slot-names (page 371)

Examples

Create some simple duplicate instances:

```
> (define-class foo ()
     ((a :initform 1)
      (b :initform 2)))
#<standard-class foo>
> (defparameter *x* (make-instance 'foo :a 11 :b 12))
*x*
> :ds *x*
#<foo #x110fe6ea> is an instance of #<standard-class foo>:
 The following slots have :instance allocation:
  a   11
  b   12
> :ds (make-duplicate-instance *x* nil :a -1)
#<foo #x110ff93a> is an instance of #<standard-class foo>:
 The following slots have :instance allocation:
  a   -1
  b   12
> :ds (make-duplicate-instance *x* '(b) :a -1)
#<foo #x11104542> is an instance of #<standard-class foo>:
 The following slots have :instance allocation:
```

```
  a   -1
  b   2
> :ds (make-duplicate-instance *x* '(b))
#<foo #x11108c82> is an instance of #<standard-class foo>:
 The following slots have :instance allocation:
  a   11
  b   2
> (make-duplicate-instance *x* nil)
#<foo @ #x1110a0e2>
(#<standard-effective-slot-definition b>
 #<standard-effective-slot-definition a>)
>
```

Create a duplicate of the `hyp 419` unit instance:

```
> (make-duplicate-instance (find-instance-by-name 19 'hyp) nil)
#<hyp 681 (1835 4791) 0.85 [5..35]>
>
```

---

make-duplicate-instance

make-duplicate-instance-changing-class *instance new-class* [*Generic Function*]
unduplicated-slot-names
`&rest` *initargs*
`&key &allow-other-keys`
⇒ *new-instance, slots*

Purpose
Create a duplicate instance of *instance*, changing its class to *new-class* in the process.

Method signatures
make-duplicate-instance-changing-class (*instance* `standard-object`) (*new-class* `class`) *unduplicated-slot-names* `&rest` *initargs* ⇒ *new-instance, slots*

make-duplicate-instance-changing-class (*instance* `standard-object`) (*new-class* `standard-space-class`) *unduplicated-slot-names* `&rest` *initargs* ⇒ *new-space-instance, slots*

make-duplicate-instance-changing-class (*instance* `standard-object`) (*new-class* `standard-unit-class`) *unduplicated-slot-names* `&rest` *initargs* ⇒ *new-unit-instance, slots*

make-duplicate-instance-changing-class (*instance* `standard-object`) (*new-class* `symbol`) *unduplicated-slot-names* `&rest` *initargs* ⇒ *new-instance, slots*

make-duplicate-instance-changing-class `:around` (*instance* `standard-unit-instance`) (*new-class* `standard-unit-class`) *unduplicated-slot-names* `&rest` *initargs* ⇒ *new-unit-instance, slots*

Package
`:gbbopen-tools` (re-exported by `:gbbopen`)

Module
`:gbbopen-tools` (the unit-instance and space-instance methods are added by `:gbbopen-core`)

Arguments
instance	A `standard-object` instance
new-class	A class designator
unduplicated-slot-names	A list of slot names
initargs	An initialization argument list
new-instance	A `standard-object` instance
slots	A proper list of slot objects

Returns
Two values: the (newly created) duplicate instance and a list of the slots that were duplicated or explicitly initialized.

Events
When a unit instance is duplicated, events are signaled in the following sequence:

1. An `nonlink-slot-updated-event` or `link-event` is signaled for each initialized slot in the duplicated unit instance. A `link-event` is also signaled for each inverse pointer from an existing unit instance to the (newly created) duplicate unit instance.

2. An `instance-added-to-space-instance-event` is signaled for each space instance on which the duplicate unit instance is added.
3. A `instance-created-event` is signaled.

Errors

Use of an initialization argument that has not been declared as valid.

Description

Slot initialization during instance duplication behaves as follows, regardless of whether the slots are local or shared:

- If an initarg in *initargs* specifies a value for a slot, that value is stored into the slot in the duplicated instance.
- All slots in the duplicated instance that are unbound at this point and that are not named in *unduplicated-slot-names* or in the list of slot-names returned from calling **unduplicated-slot-names** on *instance* are set to the value of the slot in *instance*.
- Any slots that remain unbound at this point and that are defined for *new-class* instances but not for *instance* or are named in *unduplicated-slot-names* or in the list of slot-names returned from calling **unduplicated-slot-names** on *instance* are initialized according to their `:initform` forms.

When creating (duplicating) a new unit instance, specifying a `:instance-name` initialization argument causes that value to be used as the instance name of the newly created unit instance instead of the instance-name counter value associated with the unit class (if the `:use-global-instance-name-counter` class option is `nil` or was not specified for the unit class) or the global instance-name counter value (if the `:use-global-instance-name-counter` class option is true for the unit class).

A `:space-instances` initialization argument must be provided when creating a new a space instance.

See also

change-class (page 325)
delete-instance (page 334)
describe-instance (page 338)
initial-class-instance-number (page 352)
instance-name-of (page 357)
make-duplicate-instance (page 358)
make-instance (page 364)
make-space-instance (page 455)
next-class-instance-number (page 366)
unduplicated-slot-names (page 371)

Examples

Create some simple duplicate instances:

```
> (define-class foo ()
      ((a :initform 1)
       (b :initform 2)))
#<standard-class foo>
> (define-unit-class bar ()
```

```
      ((a :initform 1)
       (b :initform 2)))
#<standard-class bar>
> (defparameter *x* (make-instance 'foo :a 11 :b 12))
*x*
> :ds *x*
#<foo #x110fe6ea> is an instance of #<standard-class foo>:
 The following slots have :instance allocation:
  a   11
  b   12
> :ds (make-duplicate-instance-changing-class *x* 'foo nil :a -1)
#<foo #x110ff93a> is an instance of #<standard-class foo>:
 The following slots have :instance allocation:
  a   -1
  b   12
> :ds (make-duplicate-instance-changing-class *x* 'bar nil :b 2)
#<bar #x11100032> is an instance of #<standard-unit-class bar>:
 The following slots have :instance allocation:
  a   11
  b   2
> :ds (make-duplicate-instance-changing-class *x* 'bar '(b) :a -1)
#<bar #x11104542> is an instance of #<standard-unit-class bar>:
 The following slots have :instance allocation:
  a   -1
  b   2
> :ds (make-duplicate-instance-changing-class *x* 'bar '(b))
#<bar #x11108c82> is an instance of #<standard-unit-class foo>:
 The following slots have :instance allocation:
  a   11
  b   2
> (make-duplicate-instance-changing-class *x* 'bar nil)
#<bar @ #x1110a0e2>
(#<standard-effective-slot-definition b>
 #<standard-effective-slot-definition a>)
>
```

Create a duplicate of the `hyp 419` unit instance as a `probable-hyp`:'

```
> (make-duplicate-instance-changing-class
    (find-instance-by-name 19 'hyp)
    'possible-hyp nil)
#<possible-hyp 681 (1835 4791) 0.85 [5..35]>
>
```

make-duplicate-instance-changing-class

make-instance *class* `&rest` *initargs* `&key &allow-other-keys` ⇒ *instance* [*Generic Function*]

Purpose

Create a new instance of *class*, such as a new unit instance.

Method signatures

make-instance (*class* `standard-class`) `&rest` *initargs* ⇒ *instance*

make-instance (*class* `symbol`) `&rest` *initargs* ⇒ *instance*

Package `:gbbopen`

Module `:gbbopen-core`

Arguments

class A class designator

initargs An initialization argument list

instance A `standard-object` instance

Returns

The newly created instance of *class*.

Events

When a unit instance is created, events are signaled in the following sequence:

1. An `nonlink-slot-updated-event` or `link-event` is signaled for each initialized slot in the newly created unit instance. A `link-event` is also signaled for each inverse pointer from an existing unit instance to the newly created unit instance.
2. An `instance-added-to-space-instance-event` is signaled for each space instance on which the newly created unit instance is added.
3. A `instance-created-event` is signaled.

Errors

Use of an initialization argument that has not been declared as valid.

If *class* is a unit class and the supplied or generated instance name is identical to the instance name of an existing unit instance of *class*.

Description

Specifying a `:space-instances` initialization argument causes that value to be used instead of the `:initial-space-instances` specification associated with the unit class. Similarly, specifying a `:instance-name` initialization argument causes that value to be used as the instance name of the newly created unit instance instead of the instance-name counter value associated with the unit class (if the `:use-global-instance-name-counter` class option is `nil` or was not specified for the unit class) or the global instance-name counter value (if the `:use-global-instance-name-counter` class option is true for the unit class).

See also

change-class (page 325)
define-event-class (page 394)
define-unit-class (page 330)
define-space-class (page 438)
delete-instance (page 334)
describe-instance (page 338)
initial-class-instance-number (page 352)
instance-name-of (page 357)
make-duplicate-instance (page 358)
make-duplicate-instance-changing-class (page 361)
make-space-instance (page 455)
next-class-instance-number (page 366)

Example

Create a new `hyp` unit instance:

```
> (make-instance 'hyp
      :location (list x y)
      :classification '(:car :truck)
      :color ':gray
      :belief .85
      :velocity-range '(5 35)
      :supporting-hyps supporting-hyps)
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

Note

The function **make-space-instance** provides a clear and convenient shorthand for creating space instances.

---

make-instance

next-class-instance-number *unit-class-name-or-instance* ⇒ *integer* [*Generic Function*]

Purpose
Increment and return the instance-name counter value associated with a unit class.

Method signatures
next-class-instance-number (*unit-class-name* `symbol`) ⇒ *integer*
next-class-instance-number (*unit-instance* `standard-unit-instance`) ⇒ *integer*

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-class-name-or-instance A unit instance or a symbol naming a unit class
integer An integer

Returns
The incremented instance-name counter value associated with the unit class.

See also

change-class (page 325)
class-instances-count (page 329)
initial-class-instance-number (page 352)
make-instance (page 364)

Examples
Increment and return the next instance-name number of unit instances of `ksa`:

```
> (next-class-instance-number 'ksa)
8
>
```

Change the class of unit instance `hyp` from `probable-hyp` to `rejected-hyp`:

```
> (change-class hyp 'rejected-hyp
    :instance-name (next-class-instance-number 'rejected-hyp))
#<rejected-hyp 1409 (896 388) .68>
>
```

Note that the `:instance-name` initarg can be eliminated if both the old and new classes for *hyp* are using the global instance-name counter.

Note
The **next-class-instance-number** function increments the global instance-name counter value if the `:use-global-instance-name-counter` class option is true for the unit class.

reset-unit-class *unit-class-or-name* [*Generic Function*]

Purpose
Resets the unit-class instance-name counter of *unit-class* to its initial value.

Method signatures
reset-unit-class (*unit-class-name* `symbol`)

reset-unit-class (*unit-class-specifier* `cons`)

reset-unit-class (*unit-class* `standard-unit-instance`)

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-class A unit class

Description
The unit-class instance-name counter of *unit-class* is reset to the value returned by calling **initial-class-instance-number** with *unit-class*, but only if the unit class is not abstract, if the `:use-global-instance-name-counter` class option is `nil` or was not specified for the unit class, and if there are no existing instances of that class. If instances do exist and the global instance-name counter is not being used for the unit class, a warning is issued.

See also
delete-blackboard-repository (page 442)
initial-class-instance-number (page 352)
reset-gbbopen (page 461)

Example
Reset the instance-name counters of all unit classes:

```
> (reset-unit-class 't)
;; Warning: Unit class standard-space-instance has 4 instances; not reset
>
```

Note that a warning was issued when resetting **standard-space-instance** because 4 space instances of that class exist.

space-instances-of *unit-instance* ⇒ *space-instances* [*Function*]

Purpose
Obtain the space instances on which a unit instance resides.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-instance A unit instance
space-instances A proper list

Returns
The list of space instances on which *unit-instance* resides.

Example
Return the space instances on which the unit instance, `unit-instance`, resides:

```
> (space-instances-of unit-instance)
(#<standard-space-instance (bb hyps)>)
>
```

Note
The returned list of space instances should not be destructively altered.

standard-unit-class [*Unit Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **standard-unit-class** is the default class of classes defined by **define-unit-class**. It is a subclass of `standard-class`.

See also

define-unit-class (page 330)
deleted-unit-instance (page 337)
standard-unit-instance (page 370)
standard-space-class (page 463)

standard-unit-instance [*Unit Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **standard-unit-instance** is an instance of **standard-unit-class** and is a superclass of every unit class that is an instance of **standard-unit-class** except itself. It is a subclass of **standard-gbbopen-instance**.

A space instance is also a unit instance, so **standard-space-instance** is a superclass of **standard-unit-instance**.

See also

deleted-unit-instance (page 337)
print-instance-slots (page 107)
standard-gbbopen-instance (page 126)
standard-space-instance (page 464)
standard-unit-class (page 369)

unduplicated-slot-names *instance* ⇒ *unduplicated-slot-names* [*Generic Function*]

Purpose

Add slot names of a class to the list of slots that are not duplicated by **make-duplicate-instance** and **make-duplicate-instance-changing-class**.

Method signatures

unduplicated-slot-names (*instance* `standard-object`) ⇒ `nil`

unduplicated-slot-names (*instance* `standard-space-instance`) ⇒ *unduplicated-slot-names*

unduplicated-slot-names (*instance* `standard-unit-instance`) ⇒ *unduplicated-slot-names*

Package

`:gbbopen-tools` (re-exported by `:gbbopen`)

Module

`:gbbopen-tools` (the unit-instance and space-instance methods are added by `:gbbopen-core`)

Arguments

instance	A `standard-object` instance
unduplicated-slot-names	A proper list

Returns

A list of unduplicated slot names for the class of *instance*.

See also

make-duplicate-instance (page 358)

make-duplicate-instance-changing-class (page 361)

Example

Specify that slot `complex-unsaved-slot` in `my-unit-instance` should be added to the list of slots that are not duplicated by **make-duplicate-instance** and **make-duplicate-instance-changing-class**:

```
(defmethod unduplicated-slot-names ((instance my-unit-instance))
  (cons 'complex-unsaved-slot (call-next-method)))
```

with-changing-dimension-values (*unit-instance* [*dimension-name**]) *declaration** *form** ⇒ *result** [*Macro*]

Purpose

Inform GBBopen that the dimension values of a unit instance potentially will be changed by the evaluation of *forms*.

Package

:gbbopen

Module

:gbbopen-core

Arguments

unit-instance	A unit instance
dimension-name	A symbol specifying a dimension of *unit-instance*
declaration	A declare expression (not evaluated)
forms	An implicit **progn** of forms to be evaluated
results	The values returned by evaluating the last *form*

Returns

The values returned by evaluating the last *form*.

Description

The locators for *unit-instance* are updated following the evaluation of the last *form*. Any retrieval operations performed during the evaluation of *forms* will use the *unit-instance* locators as they existed before evaluation of the first *form*. Retrievals performed by separate threads also should be synchronized when using **with-changing-dimension-values**.

If *dimension-names* are specified, only the locators for those dimensions of *unit-instance* will be checked and updated as needed. If no *dimension-names* are specified, the values of any or all dimensions of *unit-instance* are assumed to have been potentially changed by *forms*.

See also

check-instance-locators (page 328)

Examples

Notify GBBopen that the `x`, `y`, and `belief` dimension values of `hyp` might be changed:

```
> (with-changing-dimension-values (hyp x y belief)
    (setf (location-of hyp) '(30 40))
    (setf (belief-of hyp) 0.78))
0.78
>
```

Notify GBBopen that *some* dimension values of `hyp` might be changed (less efficient than the above, if `hyp` unit instances have many dimensions):

```
> (with-changing-dimension-values (hyp)
    (setf (location-of hyp) '(36 52))
    (incf (belief-of hyp) 0.05))
0.83
>
```

with-changing-dimension-values

5.1 Links

This section contains :`gbbopen-core` entities that pertain to links. A link represents a bi-directional relationship between two unit instances using an outgoing pointer at each unit instance pointing to the other. GBBopen's link operators maintain the bi-directional consistency of link pointers.

GBBopen also allows a link-pointer object to be used as an outgoing link pointer. An application developer can define a link-pointer object to hold attributes (such as "confidence" or "applicability conditions") that are associated with the outgoing pointer.

Attributes of a link relationship that are not directional (attributes that are to be the same when obtained from either side of the link), are best represented by representing the "link" as a first-class unit instance rather than by maintaining a link-pointer object on each end of the link that have consistent values.

check-all-instance-links `&optional` *silentp errorp* ⇒ *problem-count* [*Function*]

Purpose
Check the link slots of all unit instances for bi-directional consistency.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
silentp A generalized boolean (default is `nil`)
errorp A generalized boolean (default is `nil`)
problem-count An integer

Returns
The number of inconsistent links encountered.

Description
GBBopen automatically maintains bidirectional relationship consistency all link-slot definitions are consistent. (Definitional consistency can be checked with **check-link-definitions**). Unintended destructive list modification of the value of a link slot (for example, using `sort` without a protective `copy-list`) can break this consistency. **Check-all-instance-links** detects link inconsistencies and, when *errorp* is true, assist in repairing them.

If a link inconsistency is found, details of the inconsistency are printed to `*standard-output*`.

If *silentp* is true, warning and summary messages are not printed. If *silentp* is `nil` and *errorp* is true, a correctable error is signaled if a link inconsistency is found.

See also
check-link-definitions (page 377)

Examples
Check that all instance links are consistent:

```
> (check-all-instance-links)
;; All instance links are consistent.
0
>
```

Check again, silently:

```
> (check-all-instance-links 't)
0
>
```

The result of checking when something has set the `next-location` link slot in `location` 20 to `nil`:

```
> (check-all-instance-links)
Warning: Inverse link back to #<location 21 (32 37)> (link-slot
previous-location)
```

```
          from #<location 20 (28 28)> is missing in link-slot next-location
  ;; 1 problem was found.
  1
  >
```

Repeat, but with *errorp* true to prompt for automated repair of each problem:

```
  > (check-all-instance-links nil 't)
  Error: Inverse link back to #<location 21 (32 37)> (link-slot
previous-location)
         from #<location 20 (28 28)> is missing in link-slot next-location
  Restart actions (select using :c n):
    0: Fix the inconsistency.
  >> :c 0
  ;; Link to #<location 21 (32 37)> added.
  ;; 1 problem was found.
  ;; 1 repair was made.
  1
  > (check-all-instance-links)
  ;; All instance links are consistent.
  0
  >
```

---

check-all-instance-links

check-link-definitions `&optional` *silentp errorp* ⇒ *boolean* [*Function*]

Purpose
Check for consistency among link-slot definitions of unit classes.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
silentp A generalized boolean (default is `nil`)
errorp A generalized boolean (default is `nil`)
boolean A generalized boolean

Returns
True if all link-slot definitions are consistent; `nil` otherwise.

Description
If a link inconsistency is found, details of the inconsistency are printed to `*standard-output*`. For clarity, only the first inconsistency is displayed. After the inconsistency has been repaired, **check-link-definitions** should be used again to check for additional inconsistencies.

If *silentp* is true, warning and success printing or error signaling is supressed. If *silentp* is `nil` and *errorp* is true, an error is signaled if a link inconsistency is found.

See also
define-unit-class (page 330)
check-all-instance-links (page 375)

Examples
Check for consistency in link-slot definitions in all unit classes:

```
> (check-link-definitions)
;; All link definitions are consistent.
t
>
```

Create a link-slot inconsistency:

```
> (define-unit-class bad ()
     ((mismatched-link :link (missing inverse))))
bad
> (check-link-definitions)
;; Warning: The inverse of link MISMATCHED-LINK in unit-class BAD refers
;;          to unit-class MISSING, which is not defined.
nil
>
```

Check again, generating an error on the inconsistency:

```
> (check-link-definitions nil 't)
Error: The inverse of link MISMATCHED-LINK in unit-class BAD refers
       to unit-class MISSING, which is not defined.
>>
```

Check again, silently:

```
> (check-link-definitions 't)
nil
>
```

Define the missing unit class, but incorrectly:

```
> (define-unit-class missing () ((mismatched-link :link (missing bad))))
missing
> (check-link-definitions)
;; Warning: The inverse of link MISMATCHED-LINK in unit-class MISSING
;;          refers to link BAD which is not present in unit-class MISSING.
nil
>
```

Fix the definition and check again:

```
> (define-unit-class missing () ((inverse :link (bad mismatched-link))))
#<standard-unit-class missing>
> (check-link-definitions)
;; All link definitions are consistent.
t
>
```

Check again, silently:

```
> (check-link-definitions 't)
t
>
```

---

check-link-definitions

direct-link-definition [*Metaobject Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **direct-link-definition** is the default direct link-definition metaobject class of unit classes created by **define-unit-class**. **Direct-link-definition** is a subclass of **gbbopen-direct-slot-definition**.

See also

effective-link-definition (page 380)
direct-nonlink-slot-definition (page 346)
gbbopen-direct-slot-definition (page 348)

effective-link-definition [*Metaobject Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **effective-link-definition** is the default effective link-definition metaobject class of unit classes created by **define-unit-class**. **Effective-link-definition** is a subclass of **gbbopen-effective-slot-definition**.

See also

direct-link-definition (page 379)
effective-nonlink-slot-definition (page 347)
gbbopen-effective-slot-definition (page 349)

link-instance-of *object* ⇒ *instance* [*Generic Accessor*]

Purpose
Return the unit instance in an object that can be used as a link pointer.

Setf syntax
(setf (link-instance-of *object*) *unit-instance*) ⇒ *unit-instance*

Method signatures
link-instance-of (*object* `deleted/non-deleted-unit-instance`) ⇒ *object*
link-instance-of (*object* `standard-link-pointer`) ⇒ *unit-instance*

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
object	An object
unit-instance	A unit instance

Returns
The link pointer unit-instance.

Description
The **link-instance-of** method on a unit instance returns its argument unit-instance object.

See also

linkf (page 382)
link-setf (page 384)
standard-link-pointer (page 388)
unlinkf (page 389)

Example
Define a link-pointer object class that can be used to associate a value with a link pointer:

```
(define-class link-ptr-with-value (standard-link-pointer)
   ((value :initform nil)))
```

and an informative **print-instance-slots** method:

```
(defmethod print-instance-slots ((obj link-ptr-with-value) stream)
  (call-next-method)
  (print-instance-slot-value obj 'value stream))
```

linkf *link-slot-place unit-instance-or-instances* ⇒ *unit-instance-or-instances* [*Macro*]

Purpose
Add a link between a unit instance and one or more unit instances.

Package :gbbopen

Module :gbbopen-core

Arguments
link-slot-place	A form which is suitable for use as a generalized reference to a link slot
unit-instance-or-instances	A unit instance, a link-pointer object, or a list of unit instances and link-pointer objects

Returns
The supplied *unit-instance-or-instances*.

Events
A `link-event` is signaled for:

- all pointers that are added to the specified link-slot-place
- each inverse pointer of the link that is added to another unit instance

Description
Adding a link from a unit instance to another unit instance that is already linked to that same unit instance is ignored.

See also

link-instance-of (page 381)
link-setf (page 384)
standard-link-pointer (page 388)
unlinkf (page 389)
unlinkf-all (page 390)

Examples
Add `support-hyp` to the `supporting-hyps` link slot of the `hyp` unit instance `unit-instance`:

```
> (linkf (supporting-hyps-of unit-instance) support-hyp)
#<hyp 231 (1488 7405) 0.63 [0..8]>
>
```

Note that, when a link pointer to `hyp` 231 is already present in a link slot, adding a link to that same `hyp` unit instance—even as a link-pointer object—has no effect on the existing link pointer:

```
> (supporting-hyps-of unit-instance)
(#<hyp 231 (1488 7405) 0.63 [0..8]>)
> (linkf (supporting-hyps-of unit-instance)
         (make-instance 'link-ptr-with-value
             :link-instance support-hyp
```

```
            :value 0.9))
#<link-ptr-with-value #<hyp 231 (1488 7405) 0.63 [0..8]>
> (supporting-hyps-of unit-instance)
(#<hyp 231 (1488 7405) 0.63 [0..8]>)
>
```

This time, add a link-pointer object as a new pointer:

```
> (unlinkf-all (supporting-hyps-of unit-instance))
nil
> (linkf (supporting-hyps-of unit-instance)
         (make-instance 'link-ptr-with-value
             :link-instance support-hyp
             :value 0.9))
#<link-ptr-with-value #<hyp 231 (1488 7405) 0.63 [0..8]>
> (supporting-hyps-of unit-instance)
(#<link-ptr-with-value #<hyp 231 (1488 7405) 0.63 [0..8]>)
>
```

---

linkf

link-setf *link-slot-place unit-instance-or-instances* ⇒ *unit-instance-or-instances* [*Macro*]

Purpose

Set *link-slot-place* to be precisely *unit-instance-or-instances* links between unit instance and *unit-instance-or-instances*.

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

link-slot-place	A form which is suitable for use as a generalized reference to a link slot
unit-instance-or-instances	A unit instance, a link-pointer object, or a list of unit instances and link-pointer objects

Returns

The supplied *unit-instance-or-instances*.

Events

An `unlink-event` is signaled for:

- all pointers that are removed from the specified link-slot-place
- each inverse pointer of the link that is removed-from another unit instance

A `link-event` is signaled for:

- all pointers that are added to the specified link-slot-place (unlike **linkf**, this event is signaled even if no new pointers were added to link-slot-place)
- each inverse pointer of the link that is added to another unit instance

Description

Any existing links in *link-slot-place* that do not involve *unit-instance-or-instances* are unlinked. Existing links that are also specified as link-pointer objects in *unit-instance-or-instances* are replaced with the new link-pointer objects. Then links to any additional unit instances in *unit-instance-or-instances* are added.

The order of the specified *unit-instance-or-instances* is maintained when settting the value of *link-slot-place*, if no `:sort-function` was specified as a *link-slot-option* in the unit-class definition for that link slot. Any duplicates in *unit-instance-or-instances* are removed, but the order of the remaining elements will be the same as the order in which they appeared in *unit-instance-or-instances*.

See also

link-instance-of (page 381)
linkf (page 382)
standard-link-pointer (page 388)
unlinkf (page 389)
unlinkf-all (page 390)

Examples

Set the `supporting-hyps` link slot of the `hyp` unit instance to the unit instances in `supporting-hyps`:

```
> (link-setf (supporting-hyps-of unit-instance) supporting-hyps)
#<hyp 231 (1488 7405) 0.63 [0..8]>
>
```

Note that, when a link pointer to `hyp` 231 is already present in a link slot, adding a link-pointer-object link to that same `hyp` unit instance replaces the existing link pointer:

```
> (supporting-hyps-of unit-instance)
(#<hyp 231 (1488 7405) 0.63 [0..8]>)
> (link-setf (supporting-hyps-of unit-instance)
          (make-instance 'link-ptr-with-value
              :link-instance support-hyp
              :value 0.9))
#<link-ptr-with-value #<hyp 231 (1488 7405) 0.63 [0..8]>
> (supporting-hyps-of unit-instance)
(#<link-ptr-with-value #<hyp 231 (1488 7405) 0.63 [0..8]>)
> (value-of *)
0.9
>
```

Replace the existing link-pointer-object pointer with one that has a different `value`:

```
> (supporting-hyps-of unit-instance)
(#<link-ptr-with-value #<hyp 231 (1488 7405) 0.63 [0..8]>)
> (link-setf (supporting-hyps-of unit-instance)
          (make-instance 'link-ptr-with-value
              :link-instance support-hyp
              :value 0.94))
#<link-ptr-with-value #<hyp 231 (1488 7405) 0.63 [0..8]>
> (supporting-hyps-of unit-instance)
(#<link-ptr-with-value #<hyp 231 (1488 7405) 0.63 [0..8]>)
> (value-of *)
0.94
>
```

Of course, a `value` change can also be done directly in an existing link-pointer object:

```
> (supporting-hyps-of unit-instance)
(#<link-ptr-with-value #<hyp 231 (1488 7405) 0.63 [0..8]>)
> (setf (value-of *) 0.96)
0.96
> (supporting-hyps-of unit-instance)
(#<link-ptr-with-value #<hyp 231 (1488 7405) 0.63 [0..8]>)
> (value-of *)
0.96
>
```

Note

The form `(link-setf` *link-slot-place* `nil)` is semantically equivalent to `(unlinkf-all` *link-slot-place*`)`. However, using **unlinkf-all** is preferable stylistically and slightly faster.

---

link-setf

link-slot-p *slot* ⇒ *slot-or-nil* [*Generic Function*]

Purpose
Determine if a slot meta-object is a link slot.

Method signatures
link-slot-p (*slot* `direct-link-definition`) ⇒ *slot*
link-slot-p (*slot* `effective-link-definition`) ⇒ *slot*
link-slot-p (*slot* `slot-definition`) ⇒ `nil`

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
slot A slot meta object
slot-or-nil A slot meta object or `nil`

Returns
The *slot* if it is a link slot; nil otherwise.

See also
define-unit-class (page 330)

Example
Return the names of the link slots of the `hyp` unit class:

```
> (loop for slot in (class-slots (find-class 'hyp))
      if (link-slot-p slot) collect (slot-definition-name slot))
(supporting-hyps supported-hyps)
>
```

standard-link-pointer [*Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **standard-link-pointer** is a subclass of **standard-gbbopen-instance** with one direct slot, `link-instance`, and the slot accessor, **link-instance-of**. **Standard-link-pointer** is a useful superclass for defining objects that can be used as a link pointer.

See also

linkf (page 382)
link-setf (page 384)
link-instance-of (page 381)
standard-gbbopen-instance (page 126)
unlinkf (page 389)

Example

Define a link-pointer object class that can be used to associate a value with a link pointer:

```
(define-class link-ptr-with-value (standard-link-pointer)
   ((value :initform nil)))
```

and an informative **print-instance-slots** method:

```
(defmethod print-instance-slots ((obj link-ptr-with-value) stream)
  (call-next-method)
  (print-instance-slot-value obj 'value stream))
```

unlinkf *link-slot-place unit-instance-or-instances* ⇒ *unit-instance-or-instances* [*Macro*]

Purpose
Remove a link between a unit instance and one or more unit instances.

Package :gbbopen

Module :gbbopen-core

Arguments
link-slot-place	A form which is suitable for use as a generalized reference to a link slot
unit-instance-or-instances	A unit instance, a link-pointer object, or a list of unit instances and link-pointer objects

Returns
The supplied *unit-instance-or-instances*.

Events
An `unlink-event` is signaled for:

- all pointers that are removed from the specified link-slot-place
- each inverse pointer of the link that is removed from another unit instance

See also
link-instance-of (page 381)
linkf (page 382)
link-setf (page 384)
standard-link-pointer (page 388)
unlinkf-all (page 390)

Examples
Remove `support-hyp` from the `supporting-hyps` link slot of the `hyp` unit instance `unit-instance`:

```
> (unlinkf (supporting-hyps-of unit-instance) support-hyp)
#<hyp 231 (1488 7405) 0.63 [0..8]>
>
```

The above, but this time showing that the `support-hyp` can be a link-pointer object:

```
> (linkf (supporting-hyps-of unit-instance) support-hyp)
#<hyp 231 (1488 7405) 0.63 [0..8]>
> (unlinkf (supporting-hyps-of unit-instance)
           (make-instance 'link-ptr-with-value
               :link-instance support-hyp))
#<link-ptr-with-value #<hyp 231 (1488 7405) 0.63 [0..8]>
> (supporting-hyps-of unit-instance)
nil
>
```

unlinkf-all *link-slot-place* [*Macro*]

Purpose
Remove all the links in the specified link slot.

Package :gbbopen

Module :gbbopen-core

Arguments
link-slot-place A form which is suitable for use as a generalized reference to a link slot

Events
An `unlink-event` is signaled for:

- all pointers that are removed from the specified link-slot-place
- each inverse pointer of the link that is removed from another unit instance

See also
linkf (page 382)
link-setf (page 384)
unlinkf (page 389)

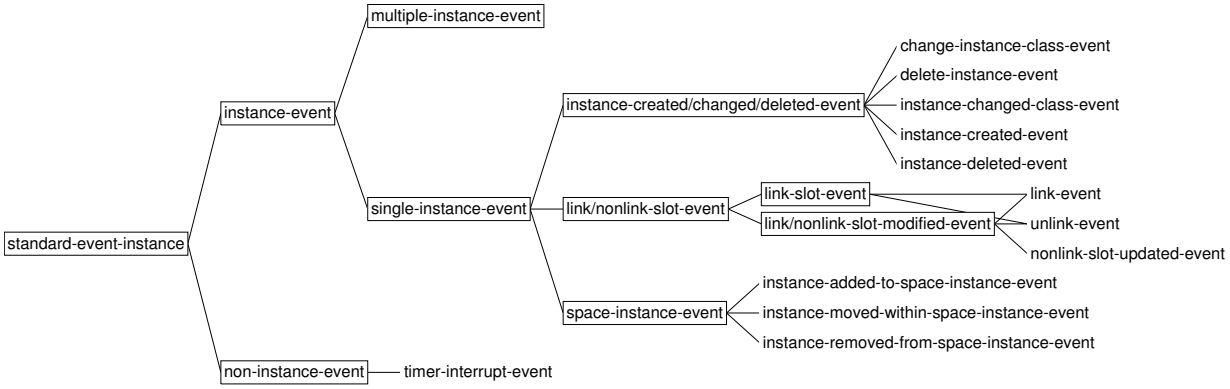
Example
Remove all supporting hypothesis links from the `supporting-hyps` link slot of the `hyp` unit instance `unit-instance`:

```
> (unlinkf-all (supporting-hyps-of unit-instance))
nil
>
```

5.2 Events

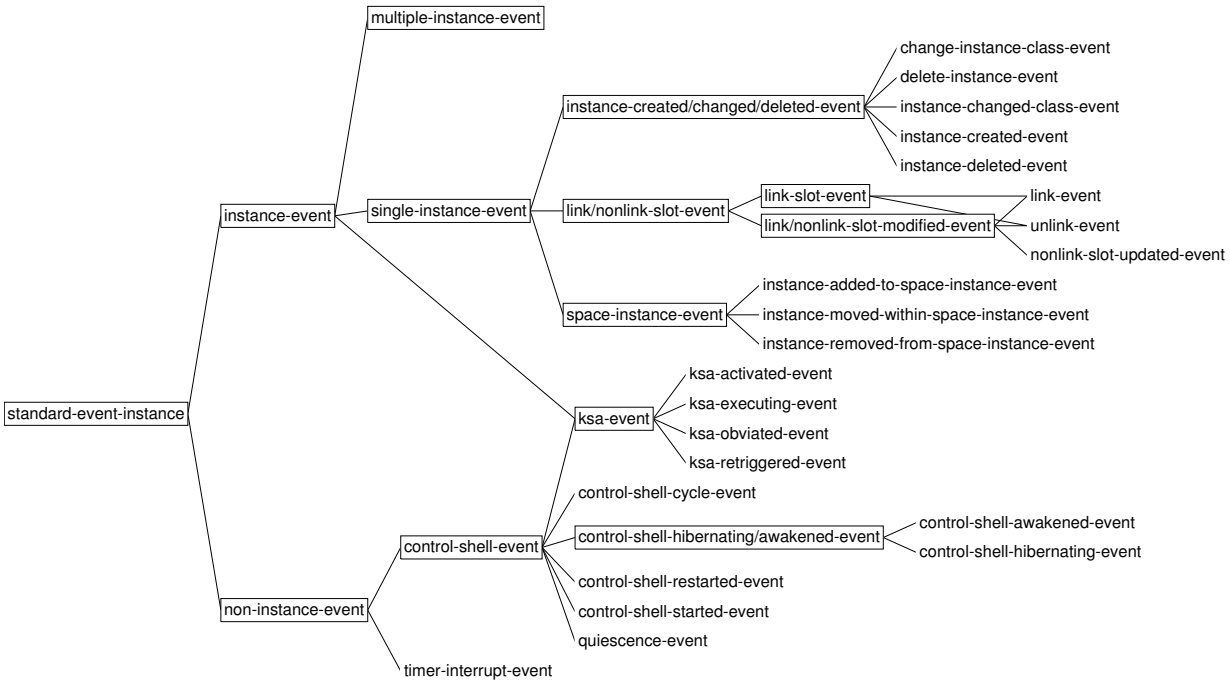
This section contains :gbbopen-core entities that pertain to events, event functions, event printing, and event signaling.

Here are the event subclasses of **standard-event-instance** that are defined in the :gbbopen-core module:



The event classes shown within rectangles are abstract classes that cannot be signaled.

Here are the defined event subclasses when both the :gbbopen-core and :agenda-shell modules have been loaded:



The additional control-shell-event classes are defined and signaled by the Agenda Shell. Again classes shown within rectangles are abstract classes that cannot be signaled.

add-event-function *function* [*event-class-specifier* [*unit-class-or-instance-specifier*]] [*Function*]
`&key` *slot-names paths permanent priority*

Purpose
Add an event function for one or more event classes.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
function	A function designator
event-class-specifier	An extended event-class specification (see below; default is `t`)
unit-class-or-instance-specifier	An extended unit-class or instance specification (see below; default is `t`)
slot-names or slot-name	A slot-name or list of slot-names (default is `t`)
paths or path	A space-instance path regular expression (default is `(*)`)
permanent	A generalized boolean (default is `nil`)
priority	An integer between -127 and 127, inclusive (default is `0`)

Detailed syntax
event-class-specifier ::= *atomic-event-class* | `(`*atomic-event-class subeventing-specifier*`)` | `t`
atomic-event-class ::= *event-class* | *event-class-name*
subeventing-specifier ::= `:plus-subevents` | `:no-subevents` | `+` | `=`

The shorthand + subevents specifier is equivalent to `:plus-subevents` and = to `:no-subevents`.

unit-class-or-instance-specifier ::= *unit-instance* | `(`*unit-instance*`*)` |
atomic-unit-class |
`(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
The specified *function* must accept the arguments associated with every event class to which it is added. In addition, *function* should accept additional arguments that are associated with all subevents of the specified event classes. (This can be achieved by specifying `&allow-other-keys` in the lambda list of *function*.)

The *paths* argument is either the symbol `t` (indicating all space instances) or a list representing a regular expression where the following reserved symbols are interpreted as follows:
- `=` matches one occurrence in a space-instance path
- `?` matches zero or one occurrence in a space-instance path
- `+` matches one or more occurrences in a space-instance path
- `*` matches zero or more occurrences in a space-instance path
- `^` move to parent

See also

remove-event-function (page 405)
remove-all-event-functions (page 403)

Examples

Add the event function evfn-printv to the set of functions to be invoked when `instance-created-event` is signaled on a `hyp` unit instance:

```
(add-event-function 'evfn-printv 'instance-created-event 'hyp)
```

Add the event function evfn-printv to the set of functions to be invoked when `instance-created-event` is signaled on a `hyp` unit instance or its subclasses:

```
(add-event-function 'evfn-printv 'instance-created-event '(hyp
:plus-subclasses))
```

or simply:

```
(add-event-function 'evfn-printv 'instance-created-event '(hyp +))
```

Note

Unit-instance-specific event functions are not yet implemented in GBBopen.

define-event-class *event-class-name* ({*superclass-name*}*) ({*slot-specifier*}*) {*class-option*}* ⇒ *new-event-class* [*Macro*]

Purpose
Define or redefine an event class.

Package
:gbbopen

Module
:gbbopen-core

Arguments
event-class-name	A non-nil, non-keyword symbol that names the event class
superclass-name	A non-nil, non-keyword symbol that specifies a direct superclass of the event class *event-class-name*
slot-specifiers	See below
class-options	See below
new-event-class	A new or modified event class object.

Returns
The newly defined or modified event class object.

Detailed syntax
slot-specifier ::= *slot-name* | (*slot-name* [[*slot-option*]])
slot-option ::= {:accessor *reader-function-name*}* |
{:allocation *allocation-type*} |
{:documentation *string*} |
{:initarg *initarg-name*}* |
{:initform *form*} |
{:reader *reader-function-name*}* |
{:type *type-specifier*} |
{:writer *writer-function-name*}*
class-option ::= (:abstract *boolean*) |
(:default-initargs . *initarg-list*) |
(:documentation *string*) |
(:event-metaclass *event-metaclass-specifier*) |
(:event-printing *event-printing-specifier*/code) |
(:export-accessors *boolean*) |
(:export-class-name *boolean*) |
(:export-slot-names *direct-slots-specifier*) |
(:generate-accessors *direct-slots-specifier*) |
(:generate-accessors-format {:prefix | :suffix} |
(:generate-accessors-prefix {*string* | *symbol*}) |
(:generate-accessors-suffix {*string* | *symbol*}) |
(:generate-initargs *direct-slots-specifier*) |
(:metaclass *class-name*)
event-metaclass-specifier ::= non-instance-event-class | instance-event-class |
space-instance-event-class |
nonlink-slot-event-class | link-slot-event-class

direct-slots-specifier ::= `nil` | `t` | *included-slot-name** |
{`t` `:exclude` *excluded-slot-name**}

Terms

class-name A non-`nil`, non-keyword symbol that names a class
documentation A documentation string
initarg-list An initialization argument list
slot-name A non-`nil`, non-keyword symbol

Description

Each *superclass-name* argument specifies a direct superclass of the new class. If the superclass list is empty, then the direct superclass defaults to the single class **standard-event-instance**.

The `:metaclass` *class-name* class option, if specified, must be a subclass of **standard-event-class**. The default metaclass value is the metaclass of the event superclasses of *event-class-name* if they all have the smae metaclass. If the event superclasses have multiple metaclasses, the metaclass of *event-class-name* must be provided. The following table lists the compatible event-superclass metaclasses for each event metaclass:

	Compatible Event-Superclass Metaclasses				
Event Metaclass	non-instance	instance	space-instance	nonlink-slot	link-slot
non-instance-event-class	**X**				
instance-event-class	**X**	**X**			
space-instance-event-class	**X**	**X**	**X**		
nonlink-slot-event-class	**X**	**X**		**X**	
link-slot-event-class	**X**	**X**			**X**

The table in the documentation for **signal-event** lists the initialization arguments that are required when signaling an event. These required initialization arguments are based on the event metaclass of the event class of the event that is being signaled.

See also

signal-event (page 409)
standard-event-class (page 410)
standard-event-instance (page 411)
with-generate-accessors-format (page 136)

Example

```
> (define-event-class my-event (non-instance-event)
    ((my-event-arg1 :initform nil)
     (my-event-arg2 :initform nil)))
#<non-instance-event-class my-event>
>
```

describe-event-printing [*event-class-specifier* [*unit-class-or-instance-specifier*]] [*Function*]
`&key` *slot-names paths*

Purpose
Describe the printing of events for one or more event classes.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
event-class-specifier	An extended event-class specification (see below; default is `t`)
unit-class-or-instance-specifier	An extended unit-class or instance specification (see below; default is `t`)
slot-names or slot-name	A slot-name or list of slot-names (default is `t`)
paths or path	A space-instance path regular expression (default is `(*)`)

Detailed syntax
event-class-specifier ::= *atomic-event-class* | `(`*atomic-event-class subeventing-specifier*`)` | `t`
atomic-event-class ::= *event-class* | *event-class-name*
subeventing-specifier ::= `:plus-subevents` | `:no-subevents` | `+` | `=`

The shorthand + subevents specifier is equivalent to `:plus-subevents` and = to `:no-subevents`.

unit-class-or-instance-specifier ::= *unit-instance* | `(`*unit-instance*`*)` |
atomic-unit-class |
`(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
The *paths* argument is either the symbol `t` (indicating all space instances) or a list representing a regular expression where the following reserved symbols are interpreted as follows:
- `=` matches one occurrence in a space-instance path
- `?` matches zero or one occurrence in a space-instance path
- `+` matches one or more occurrences in a space-instance path The description is printed to the
- `*` matches zero or more occurrences in a space-instance path
- `^` move to parent

standard-output stream.

See also
disable-event-printing (page 398)
enable-event-printing (page 400)
resume-event-printing (page 407)
suspend-event-printing (page 412)

Example

Describe all event printing:

```
> (describe-event-printing 'instance-event)
instance-event
  standard-unit-instance
  uc-2 [suspended]
  uc-1 [suspended]
  ksa
  ks
  standard-space-instance
>
```

Note

Unit-instance-specific event functions are not yet implemented in GBBopen.

---

describe-event-printing

disable-event-printing [*event-class-specifier* [*unit-class-or-instance-specifier*]] [*Function*]
`&key` *slot-names paths*

Purpose
Disable the printing of events for one or more event classes.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
event-class-specifier	An extended event-class specification (see below; default is `t`)
unit-class-or-instance-specifier	An extended unit-class or instance specification (see below; default is `t`)
slot-names or slot-name	A slot-name or list of slot-names (default is `t`)
paths or path	A space-instance path regular expression (default is `(*)`)

Detailed syntax
event-class-specifier ::= *atomic-event-class* | `(`*atomic-event-class subeventing-specifier*`)` | `t`
atomic-event-class ::= *event-class* | *event-class-name*
subeventing-specifier ::= `:plus-subevents` | `:no-subevents` | `+` | `=`

The shorthand + subevents specifier is equivalent to `:plus-subevents` and = to `:no-subevents`.

unit-class-or-instance-specifier ::= *unit-instance* | `(`*unit-instance*`*)` |
atomic-unit-class |
`(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
The *paths* argument is either the symbol `t` (indicating all space instances) or a list representing a regular expression where the following reserved symbols are interpreted as follows:
- `=` matches one occurrence in a space-instance path
- `?` matches zero or one occurrence in a space-instance path
- `+` matches one or more occurrences in a space-instance path
- `*` matches zero or more occurrences in a space-instance path
- `^` move to parent

See also
describe-event-printing (page 396)
enable-event-printing (page 400)
resume-event-printing (page 407)
suspend-event-printing (page 412)

Example
Disable all event printing:

```
(disable-event-printing)
```

Note

Unit-instance-specific event functions are not yet implemented in GBBopen.

---

disable-event-printing

enable-event-printing [*event-class-specifier* [*unit-class-or-instance-specifier*]] [*Function*]
`&key` *slot-names paths*

Purpose
Enable the printing of events for one or more event classes.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
event-class-specifier	An extended event-class specification (see below; default is `t`)
unit-class-or-instance-specifier	An extended unit-class or instance specification (see below; default is `t`)
slot-names or slot-name	A slot-name or list of slot-names (default is `t`)
paths or path	A space-instance path regular expression (default is `(*)`)

Detailed syntax
event-class-specifier ::= *atomic-event-class* | `(`*atomic-event-class subeventing-specifier*`)` | `t`
atomic-event-class ::= *event-class* | *event-class-name*
subeventing-specifier ::= `:plus-subevents` | `:no-subevents` | `+` | `=`

The shorthand `+` subevents specifier is equivalent to `:plus-subevents` and `=` to `:no-subevents`.

unit-class-or-instance-specifier ::= *unit-instance* | `(`*unit-instance*`*)` |
atomic-unit-class |
`(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand `+` subclasses specifier is equivalent to `:plus-subclasses` and `=` to `:no-subclasses`.

Description
The *paths* argument is either the symbol `t` (indicating all space instances) or a list representing a regular expression where the following reserved symbols are interpreted as follows:
- `=` matches one occurrence in a space-instance path
- `?` matches zero or one occurrence in a space-instance path
- `+` matches one or more occurrences in a space-instance path
- `*` matches zero or more occurrences in a space-instance path
- `^` move to parent

See also
describe-event-printing (page 396)
disable-event-printing (page 398)
resume-event-printing (page 407)
suspend-event-printing (page 412)

Example

Enable event printing on all space-instance events of `hyp` unit instances:

```
(enable-event-printing '(space-instance-event :plus-subevents)
                       '(hyp :plus-subclasses))
```

or simply:

```
(enable-event-printing '(space-instance-event +) '(hyp +))
```

Note

Unit-instance-specific event functions are not yet implemented in GBBopen.

---

enable-event-printing

evfn-printv *event-class* &rest *args* [*Function*]

Purpose
Assist debugging by printing forms and the results of evaluating them to `*trace-output*`.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
forms An implicit **progn** of forms to be evaluated and printed

Returns
The values returned by evaluating the last *form*.

Description
Evaluates *forms*, printing the *form* and the result values of each evaluation to `*trace-output*`. Any*form* that is a string (before evaluation) is simply printed without enclosing double-quote characters.

Examples
Add the event function evfn-printv to the set of functions to be invoked when `instance-created-event` is signaled on a `hyp` unit instance:

```
(add-event-function 'evfn-printv 'instance-created-event 'hyp)
```

remove-all-event-functions [*event-class-specifier* [*unit-class-or-instance-specifier*]] [*Function*]
`&key` *slot-names paths permanent*

Purpose
Remove all event functions for one or more event classes.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
event-class-specifier	An extended event-class specification (see below; default is `t`)
unit-class-or-instance-specifier	An extended unit-class or instance specification (see below; default is `t`)
slot-names or slot-name	A slot-name or list of slot-names (default is `t`)
paths or path	A space-instance path regular expression (default is `(*)`)
permanent	A generalized boolean (default is `nil`)

Detailed syntax
event-class-specifier ::= *atomic-event-class* | `(`*atomic-event-class subeventing-specifier*`)` | `t`
atomic-event-class ::= *event-class* | *event-class-name*
subeventing-specifier ::= `:plus-subevents` | `:no-subevents` | `+` | `=`

The shorthand + subevents specifier is equivalent to `:plus-subevents` and = to `:no-subevents`.

unit-class-or-instance-specifier ::= *unit-instance* | `(`*unit-instance*`*)` |
atomic-unit-class |
`(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

See also
add-event-function (page 392)
remove-event-function (page 405)

Examples
Remove all event functions associated with a `instance-created-event` on a `hyp` unit instance:

```
(remove-all-event-functions 'instance-created-event 'hyp)
```

Remove all event functions associated with a `instance-created-event` on a `hyp` unit instance or its subclasses:

```
(remove-all-event-functions 'instance-created-event '(hyp :plus-subclasses))
```

or simply:

```
(remove-all-event-functions 'instance-created-event '(hyp +))
```

Note

Unit-instance-specific event functions are not yet implemented in GBBopen.

---

remove-all-event-functions

remove-event-function *function* [*event-class-specifier* [*unit-class-or-instance-specifier*]] [*Function*]
`&key` *slot-names paths permanent*

Purpose
Remove an event function for one or more event classes.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
function A function designator
event-class-specifier An extended event-class specification (see below; default is `t`)
unit-class-or-instance-specifier An extended unit-class or instance specification (see below; default is `t`)
slot-names or slot-name A slot-name or list of slot-names (default is `t`)
paths or path A space-instance path regular expression (default is `(*)`)
permanent A generalized boolean (default is `nil`)

Detailed syntax
event-class-specifier ::= *atomic-event-class* | `(`*atomic-event-class subeventing-specifier*`)` | `t`
atomic-event-class ::= *event-class* | *event-class-name*
subeventing-specifier ::= `:plus-subevents` | `:no-subevents` | `+` | `=`

The shorthand + subevents specifier is equivalent to `:plus-subevents` and = to `:no-subevents`.

unit-class-or-instance-specifier ::= *unit-instance* | `(`*unit-instance*`*)` |
atomic-unit-class |
`(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

See also
add-event-function (page 392)
remove-all-event-functions (page 403)

Examples
Remove the event function **evfn-printv** from the set of functions to be invoked when `instance-created-event` is signaled on a `hyp` unit instance:

```
(remove-event-function 'evfn-printv 'instance-created-event 'hyp)
```

Remove the event function **evfn-printv** from the set of functions to be invoked when `instance-created-event` is signaled on a `hyp` unit instance or its subclasses:

```
(remove-event-function 'evfn-printv 'instance-created-event '(hyp
:plus-subclasses))
```

or simply:

```
(remove-event-function 'evfn-printv 'instance-created-event '(hyp +))
```

Note

Unit-instance-specific event functions are not yet implemented in GBBopen.

---

remove-event-function

resume-event-printing [*event-class-specifier* [*unit-class-or-instance-specifier*]] [*Function*]
`&key` *slot-names paths*

Purpose
Resume the printing of printing-enabled events for one or more event classes.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
event-class-specifier	An extended event-class specification (see below; default is `t`)
unit-class-or-instance-specifier	An extended unit-class or instance specification (see below; default is `t`)
slot-names or slot-name	A slot-name or list of slot-names (default is `t`)
paths or path	A space-instance path regular expression (default is `(*)`)

Detailed syntax
event-class-specifier ::= *atomic-event-class* | `(`*atomic-event-class subeventing-specifier*`)` | `t`
atomic-event-class ::= *event-class* | *event-class-name*
subeventing-specifier ::= `:plus-subevents` | `:no-subevents` | `+` | `=`

The shorthand + subevents specifier is equivalent to `:plus-subevents` and = to `:no-subevents`.

unit-class-or-instance-specifier ::= *unit-instance* | `(`*unit-instance*`*)` |
atomic-unit-class |
`(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
The *paths* argument is either the symbol `t` (indicating all space instances) or a list representing a regular expression where the following reserved symbols are interpreted as follows:
- `=` matches one occurrence in a space-instance path
- `?` matches zero or one occurrence in a space-instance path
- `+` matches one or more occurrences in a space-instance path
- `*` matches zero or more occurrences in a space-instance path
- `^` move to parent

See also
describe-event-printing (page 396)
disable-event-printing (page 398)
enable-event-printing (page 400)
suspend-event-printing (page 412)

Example
Resume all suspended event printing:

```
(resume-event-printing)
```

Note

Resuming event printing does not enable event printing that is disabled.

Unit-instance-specific event functions are not yet implemented in GBBopen.

---

resume-event-printing

signal-event *event-class* `&rest` *initargs* [*Function*]

Purpose
Signal an event.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
event-class An event class or a non-`nil`, non-keyword symbol that names an event class
initargs An initialization argument list

Description
The following table lists the initialization arguments that are required for specific event metaclasses:

Event metaclass	**Required initargs**
`non-instance-event-class`	None
`instance-event-class`	`:instance` *unit-instance*
`space-instance-event-class`	`:instance` *unit-instance*
	`:space-instance` *space-instance*
`nonlink-slot-event-class`	`:instance` *unit-instance*
	`:slot` *effective-nonlink-slot-definition*
`link-slot-event-class`	`:instance` *unit-instance*
	`:slot` *effective-link-definition*

See also
define-event-class (page 394)
with-events-disabled (page 414)
with-events-enabled (page 415)

Example
```
(signal-event 'my-event :my-event-arg1 3)
```

standard-event-class [*Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **standard-event-class** is the superclass of classes defined by **define-event-class**. It is a subclass of `standard-class`.

See also

define-event-class (page 394)
standard-event-instance (page 411)

standard-event-instance [*Event Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **standard-event-instance** is an instance of **standard-event-class** and is a superclass of every event class that is an instance of **standard-event-class** except itself. It is a subclass of **standard-gbbopen-instance**.

See also

print-instance-slots (page 107)
standard-gbbopen-instance (page 126)
standard-event-class (page 410)

suspend-event-printing [*event-class-specifier* [*unit-class-or-instance-specifier*]] [*Function*]
`&key` *slot-names paths*

Purpose
Suspend the printing of printing-enabled events for one or more event classes.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
event-class-specifier	An extended event-class specification (see below; default is `t`)
unit-class-or-instance-specifier	An extended unit-class or instance specification (see below; default is `t`)
slot-names or slot-name	A slot-name or list of slot-names (default is `t`)
paths or path	A space-instance path regular expression (default is `(*)`)

Detailed syntax
event-class-specifier ::= *atomic-event-class* | `(`*atomic-event-class subeventing-specifier*`)` | `t`
atomic-event-class ::= *event-class* | *event-class-name*
subeventing-specifier ::= `:plus-subevents` | `:no-subevents` | `+` | `=`

The shorthand + subevents specifier is equivalent to `:plus-subevents` and = to `:no-subevents`.

unit-class-or-instance-specifier ::= *unit-instance* | `(`*unit-instance*`*)` |
atomic-unit-class |
`(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
Suspending event printing is a convenient way of switching off event printing without losing event-printing enabled/disabled settings. Disabled event printing remains disabled if event printing is resumed (by using **resume-event-printing**).

The *paths* argument is either the symbol `t` (indicating all space instances) or a list representing a regular expression where the following reserved symbols are interpreted as follows:
- `=` matches one occurrence in a space-instance path
- `?` matches zero or one occurrence in a space-instance path
- `+` matches one or more occurrences in a space-instance path
- `*` matches zero or more occurrences in a space-instance path
- `^` move to parent

See also
describe-event-printing (page 396)
disable-event-printing (page 398)
enable-event-printing (page 400)
resume-event-printing (page 407)

Example

Suspend all event printing associated with `possible-hyp` unit instances:

```
(suspend-event-printing 't 'possible-hyp)
```

Note

Unit-instance-specific event functions are not yet implemented in GBBopen.

---

suspend-event-printing

with-events-disabled (*option**) *declaration** *form** ⇒ *result** [*Macro*]

Purpose
Disable event signaling during evaluation of *forms*.

Package :gbbopen

Module :gbbopen-core

Arguments
option No options are currently supported
declaration A declare expression (not evaluated)
forms An implicit **progn** of forms to be evaluated
results The values returned by evaluating the last *form*

Returns
The values returned by evaluating the last *form*.

See also
signal-event (page 409)
with-events-enabled (page 415)

Example
Create a `hyp` without signaling any events:

```
> (with-events-disabled ()
    (make-instance 'hyp
      :location (list x y)
      :classification '(:car :truck)
      :color ':red
      :belief .85
      :velocity-range '(5 35)
      :supporting-hyps supporting-hyps))
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

with-events-enabled (*option**) *declaration** *form** ⇒ *result** [*Macro*]

Purpose
Restore event signaling during evaluation of *forms*.

Package :gbbopen

Module :gbbopen-core

Arguments
option No options are currently supported
declaration A declare expression (not evaluated)
forms An implicit **progn** of forms to be evaluated
results The values returned by evaluating the last *form*

Returns
The values returned by evaluating the last *form*.

See also
signal-event (page 409)
with-events-disabled (page 414)

Example
Create a hyp without signaling any events, then add supporting-hypothesis links with events enabled:

```
> (with-events-disabled
     (let ((hyp (make-instance 'hyp
                  :location (list x y)
                  :classification '(:car :truck)
                  :color ':red
                  :belief .85
                  :velocity-range '(5 35))))
        (with-events-enabled ()
           (linkf (supporting-hyps-of hyp) supporting-hyps))
        hyp))
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

5.3 Intervals

This section contains `:gbbopen-core` entities that pertain to intervals. An interval $[a, b]$ is the set of real numbers between the start value of the interval, a, and the end value, b, inclusive. The interval $[x, x]$ represents the single point x.

An interval is represented as either a cons, a two-element list, or a two-element array containing the start and end values of the interval. So, a representation for the interval $[0, 100]$ can be created as any of the following:

- `(cons 0 100)`
- `(list 0 100)`
- `(vector 0 100)`

The function **make-interval** is provided for stylistic clarity in creating an interval.

Intervals also include the unbounded intervals:

- $(-\infty, \infty)$ (provided as the constant **infinite-interval**)
- $(-\infty, x]$ (for example, **(make-interval x infinity)**)
- $[x, \infty)$ (for example, **(make-interval -infinity x)**)

It is an error for the start value of an interval to be greater than the end value.

coerce-contracted-interval-rationals-to-floats [*Variable*]

Purpose

Control automatic coercion of non-integer rationals to floats when an interval is contracted into a non-integral point range by **expand-interval** and **nexpand-interval**.

Package

`:gbbopen`

Module

`:gbbopen-core`

Value type

A generalized boolean

Initial value

`nil`

See also

expand-interval (page 419)
nexpand-interval (page 426)

Examples

```
> (let ((*coerce-contracted-interval-rationals-to-floats* 't))
    (expand-interval '(2 . 5) -3))
(3.5 . 3.5)
> (let ((*coerce-contracted-interval-rationals-to-floats* nil))
    (expand-interval '(2 . 5) -3))
(7/2 . 7/2)
>
```

copy-interval *interval* ⇒ *new-interval* [*Function*]

Purpose
Create a new interval by copying *interval*.

Package :gbbopen

Module :gbbopen-core

Arguments
interval An interval
new-interval An interval

Returns
The new interval.

Description
The structure of the original *interval* (cons, two-element list, or two-element array) is maintained in the newly allocated *new-interval*.

See also
expand-interval (page 419)
infinite-interval (page 421)
interval-start (page 423)
interval-end (page 422)
make-interval (page 425)
nexpand-interval (page 426)
nshift-interval (page 427)
shift-interval (page 428)

Examples
```
> (copy-interval '(2 5))
(2 5)
> (copy-interval '(2 . 5))
(2 . 5)
> (expand-interval #(2 5))
#(2 5)
>
```

expand-interval *interval amount* ⇒ *new-interval* [*Function*]

Purpose
Create a new interval by expanding *interval* by *amount*.

Package :gbbopen

Module :gbbopen-core

Arguments
interval An interval
amount A numbe
new-interval An interval

Returns
A new, expanded interval.

Description
The structure of the original *interval* (cons, two-element list, or two-element array) is maintained in the newly allocated, expanded *new-interval*.

An interval that is contracted (expanded negatively) by an amount greater than one-half of its width will result in a zero-width *new-interval* at the center point of the original *interval*.

See also

coerce-contracted-interval-rationals-to-floats (page 417)
copy-interval (page 418)
infinite-interval (page 421)
interval-start (page 423)
interval-end (page 422)
interval-values (page 424)
make-interval (page 425)
nexpand-interval (page 426)
nshift-interval (page 427)
shift-interval (page 428)

Examples
```
> (expand-interval '(2 5) 2)
(0 7)
> (expand-interval '(2 . 5) -1)
(3 . 4)
> (expand-interval #(2 5) .5)
#(1.5 5.5)
> (expand-interval '(2 . 5) -3)
(3.5 . 3.5)
>
```

expand-point *point amount* `&optional` *type-specifier* ⇒ *new-interval* [*Function*]

Purpose
Create a new interval by expanding *point* by *amount*.

Package :gbbopen

Module :gbbopen-core

Arguments
point	A number
amount	A number
type-specifier	One of: `cons`, `list`, or `array`. (Default is `cons`.)
new-interval	An interval

Returns
The new interval.

See also
expand-interval (page 419)
interval-start (page 423)
interval-end (page 422)
interval-values (page 424)
make-interval (page 425)
nexpand-interval (page 426)
nshift-interval (page 427)
shift-interval (page 428)

Examples
```
> (expand-point 3 2)
(1 . 5)
> (expand-point 3 2 'cons)
(1 . 5)
> (expand-point 3 2 'list)
(1 5)
> (expand-point 3 2 'array)
#(1 5)
>
```

Note
Declared numeric (see page 143) and pseudo probability (see page 149) versions of **expand-point** are also provided: **expand-point&**, **expand-point$&**, **expand-point$**, **expand-point$$**, **expand-point$$$**, and **expand-point%**.

infinite-interval [*Constant*]

Purpose
An interval (represented as a cons) from `-infinity` to `infinity`.

Package
`:gbbopen`

Module
`:gbbopen-core`

Value type
A cons

Value
`(-infinity . infinity)`

See also

copy-interval (page 418)
expand-interval (page 419)
interval-end (page 422)
interval-start (page 423)
interval-values (page 424)
make-interval (page 425)
nexpand-interval (page 426)
nshift-interval (page 427)
shift-interval (page 428)

Example
Define a unit class, `temporal-duration-mixin`, that contains a `temporal-duration` slot and dimension value declaration:

```
> (define-unit-class temporal-duration-mixin ()
    ((temporal-duration
       ;; Copy the interval to allow destructive changes by
       ;; GBBopen's interval operators:
       :initform (copy-interval infinite-interval)))
    (:dimensional-values
     (temporal-duration :interval temporal-duration)))
#<standard-unit-class temporal-duration-mixin>
>
```

interval-end *interval* ⇒ *end-value* [*Function*]

Purpose
Obtain the end value of an interval.

Setf syntax
(setf (interval-end *interval*) *end-value*) ⇒ *end-value*

Package :gbbopen

Module :gbbopen-core

Arguments
interval An interval
end-value A number

Returns
The end value of the interval.

See also
interval-start (page 423)
interval-values (page 424)

Examples
```
> (interval-end '(1 2))
2
> (interval-end '(1 . 2))
2
> (interval-end #(1  2))
2
> (defparameter *x* (make-interval 1 2))
*x*
> *x*
(1 . 2)
> (setf (interval-end *x*) 4)
4
> *x*
(1 . 4)
>
```

interval-start *interval* ⇒ *start-value* [*Function*]

Purpose
Obtain the start value of an interval.

Setf syntax
(setf (interval-start *interval*) *start-value*) ⇒ *start-value*

Package :gbbopen

Module :gbbopen-core

Arguments
interval An interval
start-value A number

Returns
The start value of the interval.

See also
interval-end (page 422)
interval-values (page 424)

Examples
```
> (interval-start '(1 2))
1
> (interval-start '(1 . 2))
1
> (interval-start #(1 2))
1
> (defparameter *x* (make-interval 1 2))
*x*
> *x*
(1 . 2)
> (setf (interval-start *x*) -1)
-1
> *x*
(-1 . 4)
>
```

interval-values *interval* ⇒ *start-value, end-value* [*Function*]

Purpose
Obtain the start and end values of an interval.

Setf syntax
(setf (interval-values *interval*) *source-interval*) ⇒ *source-interval*

Package :gbbopen

Module :gbbopen-core

Arguments
interval	An interval
source-interval	An interval
start-value	A number
end-value	A number

Returns
Two values: the start value and the end value of the interval.

See also
interval-end (page 422)
interval-start (page 423)

Examples
```
> (interval-values '(1 2))
1
2
> (interval-values '(1 . 2))
1
2
> (interval-values #(1  2))
1
2
> (defparameter *x* (make-interval 1 2))
*x*
> *x*
(1 . 2)
> (setf (interval-values *x*) #(3 4))
#(3 4)
> *x*
(3 . 4)
>
```

make-interval *start end* `&optional` *type-specifier* ⇒ *new-interval* [*Function*]

Purpose
Create a new interval of type *type-specifier*.

Package :gbbopen

Module :gbbopen-core

Arguments
start	A number
end	A number
type-specifier	One of: `cons`, `list`, or `array`. (Default is `cons`.)
new-interval	An interval

Returns
The new interval.

See also
copy-interval (page 418)
expand-interval (page 419)
expand-point (page 420)
infinite-interval (page 421)
interval-start (page 423)
interval-end (page 422)
nexpand-interval (page 426)
nshift-interval (page 427)
shift-interval (page 428)

Examples
```
> (make-interval 2 5)
(2 . 5)
> (make-interval 2 5 'list)
(2 5)
> (make-interval 2 5 'cons)
(2 . 5)
> (make-interval 2 5 'array)
#(2 5)
>
```

nexpand-interval *interval amount* ⇒ *interval* [*Function*]

Purpose
Expand an interval by *amount*.

Package :gbbopen

Module :gbbopen-core

Arguments
interval An interval
amount A number

Returns
The expanded *interval*.

Description
An interval that is contracted (expanded negatively) by an amount greater than one-half of its width will result in a zero-width *interval* at the center point of the original *interval*.

See also

coerce-contracted-interval-rationals-to-floats (page 417)
copy-interval (page 418)
expand-interval (page 419)
expand-point (page 420)
infinite-interval (page 421)
interval-start (page 423)
interval-values (page 424)
interval-end (page 422)
make-interval (page 425)
nshift-interval (page 427)
shift-interval (page 428)

Examples
```
> (nexpand-interval '(2 5) 2)
(0 7)
> (nexpand-interval '(2 . 5) -1)
(3 . 4)
> (nexpand-interval #(2 5) .5)
#(1.5 5.5)
> (nexpand-interval '(2 . 5) -3)
(3.5 . 3.5)
>
```

nshift-interval *interval amount* ⇒ *interval* [*Function*]

Purpose
Shift an interval by *amount*.

Package :gbbopen

Module :gbbopen-core

Arguments
interval An interval
amount A number

Returns
The shifted *interval*.

See also
copy-interval (page 418)
expand-interval (page 419)
expand-point (page 420)
infinite-interval (page 421)
interval-end (page 422)
interval-start (page 423)
interval-values (page 424)
make-interval (page 425)
nexpand-interval (page 426)
shift-interval (page 428)

Examples
```
> (nshift-interval '(2 5) 2)
(4 7)
> (nshift-interval '(2 . 5) -1)
(1 . 4)
> (nshift-interval #(2 5) .5)
#(2.5 5.5)
>
```

shift-interval *interval amount* ⇒ *new-interval* [*Function*]

Purpose
Create a new interval by shifting *interval* by *amount*.

Package :gbbopen

Module :gbbopen-core

Arguments
interval An interval
amount A number
new-interval An interval

Returns
A new, shifted interval.

Description
The structure of the original *interval* (cons, two-element list, or two-element array) is maintained in the newly allocated, shifted *new-interval*.

See also
copy-interval (page 418)
expand-interval (page 419)
expand-point (page 420)
infinite-interval (page 421)
interval-end (page 422)
interval-start (page 423)
interval-values (page 424)
make-interval (page 425)
nexpand-interval (page 426)
nshift-interval (page 427)

Examples
```
> (shift-interval '(2 5) 2)
(4 7)
> (shift-interval '(2 . 5) -1)
(1 . 4)
> (shift-interval #(2 5) .5)
#(2.5 5.5)
>
```

5.4 Blackboard Repository

This section contains :`gbbopen-core` entities that pertain to space instances and the blackboard repository.

add-instance-to-space-instance *unit-instance space-instance-or-path* ⇒ *unit-instance* [*Generic Function*]

Purpose
Add a unit instance to a space instance.

Method signatures
add-instance-to-space-instance (*unit-instance* `standard-unit-instance`)(*space-instance-path* `cons`)⇒ *unit-instance*

add-instance-to-space-instance (*unit-instance* `standard-unit-instance`)(*space-instance* `standard-space-instance`)⇒ *unit-instance*

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-instance The unit instance to be added

space-instance-or-path The space instance or space-instance path to which the unit instance is to be added

Returns
The supplied *unit-instance*

Events
An `instance-added-to-space-instance-event` is signaled.

See also

define-unit-class (page 330)
make-instance (page 364)
make-space-instance (page 455)
remove-instance-from-space-instance (page 460)

Examples
Add a highly plausible hypothesis unit instance, `good-hyp`, to the `hyps` space instance:

```
> (add-instance-to-space-instance
    good-hyp (find-space-instance-by-path '(bb hyps)))
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

or

```
> (add-instance-to-space-instance good-hyp '(bb hyps))
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

allowed-unit-classes-of *space-instance* [*Generic Function*]
⇒ *extended-unit-classes-specification-list*

Purpose

Return the extended unit-classes specifications of unit classes whose instances are allowed on a space instance.

Method signatures

allowed-unit-classes-of (*space-instance* `standard-space-instance`) ⇒
extended-unit-class-specification-list

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

space-instance	A space instance
extended-unit-classes-specification-list	A proper list

Returns

A list of extended unit-classes specifications; `t`, if instances of any unit class are allowed on the space instance; or `nil`, if no unit instances are allowed on the space instance

See also

make-space-instance (page 455)

Example

Return the extended unit-classes specifications describing the allowed classes that can have their unit instances stored on the `(bb hyps)` space instance:

```
> (allowed-unit-classes-of '(bb hyps))
((hyp :plus-subclasses))
>
```

change-space-instance *space-instance* `&key` *allowed-unit-classes dimensions storage* [*Function*]
⇒ *space-instance*

Purpose
Change the dimensions, allowed unit classes, and storage of a space instance.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
space-instance-or-path The space instance or space-instance path to be changed

allowed-unit-classes An extended unit-classes specification or `nil` (see below; default is `t`)

dimensions A list of (*dimension-name dimension-type-specifier*) pairs (default is `nil`)

storage A storage specification (see below; default is `(t t unstructured)` or `nil` if *allowed-unit-classes* is `nil`)

space-instance The space instance

Returns
The space instance that was changed.

Events
When the allowed unit classes of *space-instance* are made more restrictive, unit instances that are no longer allowed on the space instance are removed from *space-instance*, signaling a `instance-removed-from-space-instance-event` for each removed unit instance.

Detailed syntax
allowed-unit-classes ::= *unit-classes-specifier* | `nil`

dimension-type-specifier ::= `:ordered` | (`:ordered` [*ordered-comparison-type*]) |
`:enumerated` | (`:enumerated` [*enumerated-comparison-type*]) |
`:boolean` | (`:boolean` [*boolean-comparison-type*])

ordered-comparison-type ::= `number` | `fixnum` | `short-float` | `single-float` |
`double-float` | `long-float` |
`pseudo-probability`

enumerated-comparison-type ::= `eq` | `eql` | `equal` | `equalp`

boolean-comparison-type ::= `t`

unit-classes-specifier ::= `t` | *single-unit-class-specifier* | (*single-unit-class-specifier*$^+$)

single-unit-class-specifier ::= *atomic-unit-class* | (*atomic-unit-class subclassing-specifier*)

atomic-unit-class ::= *unit-class* | *unit-class-name*

storage ::= (*unit-class-specifier dimension-names storage-specification*)*

dimension-names ::= *dimension-name* | (*dimension-name*$^+$) | `t`

storage-specification ::= `unstructured` |
`boolean` |
`uniform-buckets` `:layout` *dimension-buckets-specification*$^+$ |
`hashed` [`:size` *integer*]

dimension-buckets-specification ::= (*start-value end-value bucket-width*)

The default *ordered-comparison-type*, if unspecified, is `number`. The default *enumerated-comparison-type*, if unspecified, is `eql`. The default *boolean-comparison-type* is `t`.

Terms

dimension-name A symbol specifying a dimension

See also

make-space-instance (page 455)

Examples

Change the storage of space instance `(bb hyp)` to the default unstructured storage for all unit instances:

```
> (change-space-instance '(bb hyps) :storage nil)
#<standard-space-instance (bb hyps)>
>
```

Now change it to store `hyp` unit instances with uniform-bucket storage for indexing in the `x` and `y` dimensions and hashed storage in the `classification` dimension:

```
> (change-space-instance '(bb hyps)
     :storage '(((hyp :plus-subclasses) (x y)
                  uniform-buckets :layout ((0 10000 100)
                                           (0 10000 100)))
                ((hyp :plus-subclasses) (classification)
                  hashed)))
#<standard-space-instance (bb hyps)>
>
```

Now change it to store only `hyp` unit instances (no subclasses) with uniform-bucket storage for indexing in the `x` and `y` dimensions and hashed storage in the `classification` dimension:

```
> (change-space-instance '(bb hyps)
     :allowed-unit-classes 'hyp
     :storage '((hyp (x y)
                 uniform-buckets :layout ((0 10000 100)
                                          (0 10000 100)))
                (hyp (classification) hashed)))
#<standard-space-instance (bb hyps)>
>
```

Any unit instances that are subclasses of `hyp` are removed from the `(bb hyps)` space instance by the above change in allowed unit classes.

children-of *space-instance* ⇒ *space-instances* [*Generic Function*]

Purpose
Return the child space instances of a space instance.

Method signatures
children-of (*space-instance* `standard-space-instance`) ⇒ *space-instances*

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
space-instance A space instance
space-instances A proper list

Returns
A list of the child space instances.

See also
make-space-instance (page 455)
parent-of (page 459)

Example
Return the child space instances of the `(bb)` space instance:

```
> (children-of (find-space-instance-by-path '(bb))
(#<standard-space-instance (bb hyps)>
 #<standard-space-instance (bb probable-hyps)>
 #<standard-space-instance (bb rejected-hyps)>)
>
```

Note
The returned list of child space instances should not be destructively altered.

clear-space-instances *space-instances* [*Function*]

Purpose

Remove (but not delete) all unit instances from space instances.

Package :gbbopen

Module :gbbopen-core

Arguments

space-instances A space instance, a list of space instances, a space-instance path regular expression, or `t` (indicating all space instances)

Events

A `instance-removed-from-space-instance-event` is signaled for each unit instance that is removed from a space instance.

See also

do-instances-on-space-instances (page 472)
map-instances-on-space-instances (page 493)

Examples

Remove all the unit instances that reside on the `(bb probable-hyps)` space instance:

```
(clear-space-instances
  (find-space-instance-by-path '(bb probable-hyps)))
```

or

```
(clear-space-instances '(bb probable-hyps))
```

or

```
(clear-space-instances
  (find-space-instances '(bb probable-hyps)))
```

confirm-if-blackboard-repository-not-empty-p `&key` *describe-non-empty-repository* *pending-action* ⇒ *boolean* [*Function*]

Purpose
Obtain confirmation if the blackboard repository is not empty.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
describe-non-empty-repository	A generalized boolean (default is `nil`)
pending-action	A string (default is `"deleted"`)
boolean	A generalized boolean

Returns
True if the blackboard repository is empty or the user has confirmed positively; otherwise `nil`.

Description
If the blackboard repository is not empty and *describe-non-empty-repository* is true, the blackboard repository is described (using **describe-blackboard-repository**) when asking for confirmation.

The *pending-action* string is used to indicate to the user what is to happen if confirmed positively.

See also
empty-blackboard-repository-p (page 451)

Examples
```
> (confirm-if-blackboard-repository-not-empty-p)
The blackboard repository is not empty.
Continue anyway (the current contents will be deleted) [y or n]? n
nil
> (confirm-if-blackboard-repository-not-empty-p
     describe-non-empty-repository 't)

Space Instance                      Contents
--------------                      --------
known-world                         53 instances (52 location; 1 path)

Unit Class                          Instances
----------                          ---------
control-shell                              1 *
ks                                         4 +
ksa-queue                                  2 +
location                                  52
ordered-ksa-queue                          1 +
path                                       1
standard-space-instance                    1
                                    ---------
```

```
                                                62 instances
The above blackboard repository is not empty.
Continue anyway (the current contents will be deleted) [y or n]? n
nil
> (confirm-if-blackboard-repository-not-empty-p :pending-action "happy")
The blackboard repository is not empty.
Continue anyway (the current contents will be happy) [y or n]? y
t
>
```

---

confirm-if-blackboard-repository-not-empty-p

define-space-class *space-class-name* (*{superclass-name}**) (*{slot-specifier}**) [*Macro*]
*{class-option}** ⇒ *new-space-class*

Purpose
Define or redefine a space class.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
space-class-name	A non-`nil`, non-keyword symbol that names the space class
superclass-name	A non-`nil`, non-keyword symbol that specifies a direct superclass of the space class *space-class-name*
slot-specifiers	See below
class-options	See below
new-space-class	A new or modified space class object

Returns
The newly defined or modified space class object.

Errors
The specified *superclass-names* do not include at least one space class name. This error is signaled on class finalization.

Detailed syntax
(Syntax shown in gray is not supported in GBBopen Version 1.5, but will become available in a future release.)

slot-specifier ::= *slot-name* |
(*nonlink-slot-name* [[*nonlink-slot-option*]]) |
(*link-slot-name* [[*link-slot-option*]])
nonlink-slot-name ::= *slot-name*
link-slot-name ::= *slot-name*
link-slot-option ::= *slot-option* |
{`:link` *inverse-link-slot-specifier*} |
{`:singular` *boolean*} |
{`:sort-function` *function*} |
{`:sort-key` *function*}
inverse-link-slot-specifier ::= (*unit-class-name link-slot-name* [`:singular` *boolean*]) |
`:reflexive`
nonlink-slot-option ::= *slot-option* |
{`:reader` *reader-function-name*}* |
{`:writer` *writer-function-name*}*
slot-option ::= {`:accessor` *reader-function-name*}* |
{`:allocation` *allocation-type*} |
{`:documentation` *string*} |
{`:initarg` *initarg-name*}* |
{`:initform` *form*} |
{`:type` *type-specifier*}

class-option ::= `(:abstract` *boolean*`)` |
`(:default-initargs .` *initarg-list*`)` |
`(:dimensional-values` *dimension-value-specifier*`*)` |
`(:documentation` *string*`)` |
`(:estimated-instances` *integer*`)` |
`(:export-accessors` *boolean*`)` |
`(:export-class-name` *boolean*`)` |
`(:export-slot-names` *direct-slots-specifier*`)` |
`(:generate-accessors` *direct-slots-specifier*`)` |
`(:generate-accessors-format` {`:prefix` | `:suffix`}`)` |
`(:generate-accessors-prefix` {*string* | *symbol*}`)` |
`(:generate-accessors-suffix` {*string* | *symbol*}`)` |
`(:generate-initargs` *direct-slots-specifier*`)` |
`(:initial-space-instances` *initial-space-instance-specifier*`)` |
`(:instance-name-comparison-test` *instance-name-comparison-test*`)` |
`(:metaclass` *class-name*`)` |
`(:retain` {*boolean* | `:propagate`}`)` |
`(:use-global-instance-name-counter` *boolean*`)`

initial-space-instance-specifier ::= {*space-instance-path*`+` | *function*}

dimension-value-specifier ::= *incomposite-dv-specifier* | *composite-dv-specifier*

incomposite-dv-specifier ::= `(`*dimension-name dimension-value-spec dimension-value-place*`)`

composite-dv-specifier ::= `(`*dimension-name dimension-value-specifier*
composite-type dimension-value-place`)`

composite-type ::= `:set` | `:sequence` |
{`:ascending-series` *ordering-dimension-name*} |
{`:descending-series` *ordering-dimension-name*}

dimension-value-specifier ::= *dimension-value-type* |
`(`*ordered-dimension-value-type* [*ordered-comparison-type*]`)` |
`(`*enumerated-dimension-value-type* [*enumerated-comparison-type*]`)` |
`(`*boolean-dimension-value-type* [*boolean-comparison-type*]`)`

dimension-value-type ::= *ordered-dimension-value-type* |
enumerated-dimension-value-type |
boolean-dimension-value-type

ordered-dimension-value-type ::= `:point` | `:interval` | `:mixed`

enumerated-dimension-value-type ::= `:element`

boolean-dimension-value-type ::= `:boolean`

ordered-comparison-type ::= `number` | `fixnum` | `short-float` | `single-float` |
`double-float` | `long-float` |
`pseudo-probability`

enumerated-comparison-type ::= `eq` | `eql` | `equal` | `equalp`

boolean-comparison-type ::= `t`

dimension-value-place ::= {*slot-name* [*slot-name*]} | {*function* [*slot-name*]}

direct-slots-specifier ::= `nil` | `t` | *included-slot-name*`*` |
{`t :exclude` *excluded-slot-name*`*`}

The default *ordered-comparison-type*, if unspecified, is `number`. The default *enumerated-comparison-type*, if unspecified, is `eql`. The default *boolean-comparison-type* is `t`.

A *dimension-value-place* with two *slot-names* is allowed only for an `:interval` dimension-value specification.

Terms

class-name	A non-`nil`, non-keyword symbol that names a class
documentation	A documentation string
initarg-list	An initialization argument list
slot-name	A non-`nil`, non-keyword symbol
instance-name-comparison-test	One of the four standardized hash table test function names: `eq`, `eql`, `equal`, or `equalp` (default for classes of metaclass **standard-unit-class** is `eql`)

Description

A *dimension-value-place* with two *slot-names* can be specified only for `:interval` dimension-value types.

Each *superclass-name* argument specifies a direct superclass of the new class. If the superclass list is empty, then the direct superclass defaults to the single class **standard-space-instance**.

The `:metaclass` *class-name* class option, if specified, must be a subclass of **standard-space-class**. The default metaclass value is **standard-space-class**.

Inheritance of class options

The set of *dimensional-values* for a unit class is the union of the sets specified in the *dimensional-values* options of the class and its superclasses. When more than one dimension-value specification is supplied for a given dimension, the one supplied by the most specific class is used.

The effective *initial-space-instances* value for a unit class is the value specified in the definition of the most specific unit class. (No additive inheritance of *initial-space-instances* is performed.) If no definitions specify an *initial-space-instances* value, `nil` is used.

The *instance-name-comparison-test* value is not inherited. If no value is specified in the unit-class definition, the default initialization value associated with the metaclass is used.

If a *retain* value is not specified, a value of `:propagate` is used as the default if any parent unit classes have a `:propagate` retention value; otherwise `nil` is used as the default value.

The *use-global-instance-name-counter* value is not inherited. If no value is specified in the unit-class definition, the default initialization value associated with the metaclass is used.

See also

define-unit-class (page 330)
delete-blackboard-repository (page 442)
make-space-instance (page 455)
standard-space-class (page 463)
standard-space-instance (page 464)
with-generate-accessors-format (page 136)

Example

Define a space class, `space-instance-with-lock`, that has an additional slot containing a lock that can be used to synchronize operations on each space instance of that class. Then, create one instance of the `space-instance-with-lock` space class.

```
> (define-space-class space-with-lock ()
    ((lock :initform (make-lock :name "Space-Instance Lock"))))
#<standard-space-class space-with-lock>
> (make-space-instance '(bb hyps)
```

```
    :class 'space-with-lock)
#<space-with-lock (bb hyps)>
>
```

define-space-class

delete-blackboard-repository `&key` *all-classes disable-events retain-classes* [*Function*]

Purpose
Delete all unit and space instances.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
all-classes A generalized boolean (default is `nil`)

disable-events A generalized boolean (default is `t`)

retain-classes An extended unit-classes specification (see below)

Events
If *disable-events* is `nil`, the following events may be signaled as unit instances and space instances are deleted:

- `delete-instance-event`
- `unlink-event`
- `instance-removed-from-space-instance-event`
- `instance-deleted-event`

Detailed syntax
unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*$^{+}$`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*

Description
Calling **delete-blackboard-repository** deletes all unit instances and space instances that have not been defined with a `:retain` class option (unless overridden by *all-classes*). All unit instances of unit classes specified by *retain-classes* are also retained. If both *all-classes* and *retain-classes* are specified, the classes specified *retain-classes* are retained, but all other unit instances are deleted. The instance-name counters of all non-retained unit classes are reset to their initial values.

Delete-blackboard-repository does not undefine any class definitions, functions, methods, etc.

See also
confirm-if-blackboard-repository-not-empty-p (page 436)
delete-instance (page 334)
delete-all-space-instances (page 444)
delete-space-instance (page 445)
initial-class-instance-number (page 352)

Examples

Delete all unit instances and space instances (except for the unit instances of unit classes that have been defined to be retained by default):

```
(delete-blackboard-repository)
```

As above, but also delete all unit instances of unit classes that have been defined to be retained by default:

```
(delete-blackboard-repository :all-classes 't)
```

Note

This function and **reset-gbbopen** are the only GBBopen functions that disable event signaling by default. This conflicts with the normal use of **with-events-disabled** and **with-events-enabled** macros for controlling event signaling, but having events disabled is the desired behavior in almost every `delete-blackboard-repository` situation.

---

delete-blackboard-repository

delete-all-space-instances <no arguments> [*Function*]

Purpose

Delete all space instances.

Package

`:gbbopen`

Module

`:gbbopen-core`

Events

A `delete-instance-event` is signaled at the start of the deletion process of each space instance and an `instance-deleted-event` is signaled when the deletion of each space instance has been completed. The following events may also be signaled if a *space-instance* is, itself, on a space instance or is linked to other unit instances:

- `unlink-event`
- `instance-removed-from-space-instance-event`

See also

delete-all-space-instances (page 444)
delete-blackboard-repository (page 442)
delete-space-instance (page 445)
make-space-instance (page 455)
reset-gbbopen (page 461)
reset-unit-class (page 367)

Example

Delete every space instance:

```
(delete-all-space-instances)
```

delete-space-instance *space-instance-or-path* ⇒ *deleted-space-instance* [*Generic Function*]

Purpose
Delete a space instance.

Method signatures
delete-space-instance (*space-instance-path* `cons`) ⇒ *deleted-space-instance*
delete-space-instance (*space-instance* `standard-space-instance`) ⇒ *deleted-space-instance*

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
space-instance-or-path The space instance or space-instance path to be deleted
deleted-space-instance A space instance

Returns
The deleted space instance.

Events
A `delete-instance-event` is signaled at the start of the deletion process and an `instance-deleted-event` is signaled when the deletion has been completed. The following events may also be signaled if the *space-instance* is, itself, on a space instance or is linked to other unit instances:

- `unlink-event`
- `instance-removed-from-space-instance-event`

See also

delete-all-space-instances (page 444)
delete-blackboard-repository (page 442)
delete-instance (page 334)
make-space-instance (page 455)
reset-gbbopen (page 461)
reset-unit-class (page 367)

Examples
Delete the `(bb hyps)` space instance:

```
> (delete-space-instance (find-space-instance-by-path '(bb hyps))
#<deleted-unit-instance standard-space-instance (bb hyps)>
>
```

or simply

```
> (delete-space-instance '(bb hyps)
#<deleted-unit-instance standard-space-instance (bb hyps)>
>
```

delete-space-instance

describe-blackboard-repository <no arguments> [*Function*]

Purpose

Print information about the unit and space instances in the blackboard repository.

Package :gbbopen

Module :gbbopen-core

Description

Information is printed about the space instances in the blackboard repository and their contents. The total count of the unit instances of each unit class (including ones that do not reside on any space instance) is also printed, as well as a character that indicates if the unit class has been defined to be *retained* by **delete-blackboard-repository**. A plus sign indicates that the retention status will be propagated to subclasses of the unit class, while an asterisk (*) indicates a retain, But not propagated, status for the unit class.

The description is printed to the ***standard-output*** stream.

Example

```
> (describe-blackboard-repository)

Space Instance              Contents
--------------              --------
bb
   hyps                     15223 Instances (hyp 1479; sensor-report 13744)
   probable-hyps            Empty
   rejected-hyps            216 Instances (hyp 216)

Unit Class                  Instances
----------                  ---------
control-shell                       1 *
hyp                              1695
ks                                 13 +
ksa                               891 +
ksa-queue                           2 +
ordered-ksa-queue                   1 +
sensor-report                   13744
standard-space-instance             3
                            ---------
                                16350 instances
>
```

REPL Note

Describe-blackboard-repository can be invoked using the REPL command :dsbb.

describe-space-instance *space-instance-or-path* [*Generic Function*]

Purpose
Describe a space instance.

Method signatures
describe-space-instance (*space-instance-path* `cons`)

describe-space-instance (*space-instance* `standard-space-instance`)

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
space-instance-or-path A space instance or a space-instance path

Description
The description is printed to the ***standard-output*** stream.

See also

describe-instance (page 338)
describe-space-instance-storage (page 449)
make-space-instance (page 455)

Example
Describe the `hyps` space instance:

```
> (describe-space-instance '(bb hyps))
  Standard-space-instance #<standard-space-instance (bb hyps)>
    Path: (bb hyps)
    Allowed unit classes:
      (hyp :plus-subclasses)
    Dimensions:
      (belief (:ordered number))
      (velocity-range (:ordered number))
      (color (:enumerated eq))
      (classification (:enumerated eq))
      (x (:ordered fixnum))
      (y (:ordered fixnum))
>
```

REPL Note
Describe-space-instance can be invoked using the REPL command:

```
:dsi {space-instance | space-instance-path}
```

which also sets = to the described space instance.

describe-space-instance-storage *space-instance-or-path* [*Generic Function*]

Purpose
Describe the storage structure of a space instance.

Method signatures
describe-space-instance-storage (*space-instance-path* `cons`)

describe-space-instance-storage (*space-instance* `standard-space-instance`)

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
space-instance-or-path A space instance or a space-instance path

Description
The description is printed to the ***standard-output*** stream.

See also
describe-space-instance (page 448)
describe-instance (page 338)
make-space-instance (page 455)

Example
Describe the storage structure of the `hyps` space instance:

```
> (describe-space-instance-storage '(bb hyps))
  Standard-space-instance #<standard-space-instance (bb hyps)>
  2d-Uniform-Buckets (hyp+) (x y) 1.4 (857/611)
     hyp             479
     sub-hyp         132
  Unstructured-Storage (t) t N/A
>
```

REPL Note
Describe-space-instance-storage can be invoked using the REPL command:

```
:dsis {space-instance | space-instance-path}
```

which also sets = to the described space instance.

do-space-instances (*var space-instance-regexp*) {*tag* | *form*}* [*Macro*]

Purpose

Iterate over each space instance that matches a path-expression pattern.

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

var	A variable symbol
space-instance-regexp	A space-instance path regular expression specifying the space instances to be mapped over
declaration	A declare expression (not evaluated)
tag	A `go` tag (not evaluated)
form	A form

Description

The *space-instance-regexp* argument is either the symbol `t` (indicating all space instances) or a list representing a regular expression where the following reserved symbols are interpreted as follows:

- `=` matches one occurrence in a space-instance path
- `?` matches zero or one occurrence in a space-instance path
- `+` matches one or more occurrences in a space-instance path
- `*` matches zero or more occurrences in a space-instance path
- `^` move to parent

A *space-instance-regexp* value consisting of a list of space instances mapped over as supplied.

See also

find-space-instances (page 453)
map-space-instances (page 458)

Example

Remove all `hyp` unit instances from space instances that are rooted at `(bb)`:

```
(do-space-instances (space-instance '(bb +))
  (do-instances-on-space-instances (unit-instance 'hyp space-instance)
    (remove-instance-from-space-instance unit-instance space-instance)))
```

empty-blackboard-repository-p ⇒ *boolean* [*Function*]

Purpose
Determine if the blackboard repository is empty.

Package :gbbopen

Module :gbbopen-core

Arguments
boolean A generalized boolean

Returns
True if the blackboard repository is empty; otherwise `nil`.

See also
confirm-if-blackboard-repository-not-empty-p (page 436)

Examples
```
> (empty-blackboard-repository-p)
nil
> (delete-blackboard-repository :all-classes 't)
t
> (empty-blackboard-repository-p)
t
>
```

find-space-instance-by-path *space-instance-path* ⇒ *space-instance* or `nil` [*Function*]

Purpose
Return the space instance with the specified space-instance path.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
space-instance-path A space-instance path specifying the space instance to be returned
space-instance A space instance or `nil`

Returns
The specified space instance if it exists; `nil` otherwise.

See also
find-space-instances (page 453)

Example
Find the space instance with path `(bb hyps)`:

```
> (find-space-instance-by-path '(bb hyps))
#<standard-space-instance (bb hyps)>
>
```

REPL Note
Find-space-instance-by-path can be invoked using the REPL command:

`:fsi {`*space-instance* | *space-instance-path*`}`

which sets = to the found space instance.

find-space-instances *space-instance-regexp* ⇒ *space-instances* [*Function*]

Purpose
Return the space instances that match a path-expression pattern.

Package :gbbopen

Module :gbbopen-core

Arguments
space-instance-regexp A space-instance path regular expression specifying the space instances to be returned
space-instances A proper list

Returns
The specified space instances.

Description
The *space-instance-regexp* argument is either the symbol `t` (indicating all space instances) or a list representing a regular expression where the following reserved symbols are interpreted as follows:

= matches one occurrence in a space-instance path
? matches zero or one occurrence in a space-instance path
+ matches one or more occurrences in a space-instance path Thus both
* matches zero or more occurrences in a space-instance path
^ move to parent

`(find-space-instances '(*))` and `(find-space-instances 't)` return all space instances.

A *space-instance-regexp* value consisting of a list of space instances is returned unchanged.

See also
find-space-instance-by-path (page 452)
map-space-instances (page 458)

Examples
Return the space instances that are rooted at `(bb)`:

```
> (find-space-instances '(bb +))
(#<standard-space-instance (bb hyps)>
 #<standard-space-instance (bb probable-hyps)>)
 #<standard-space-instance (bb rejected-hyps)>)
>
```

Return all the space instances `(bb)` and below:

```
> (find-space-instances '(bb *))
(#<standard-space-instance (bb hyps)>
 #<standard-space-instance (bb probable-hyps)>)
 #<standard-space-instance (bb rejected-hyps)>
 #<standard-space-instance (bb)>)
>
```

find-space-instances

make-space-instance *path* `&rest` *initargs* `&key` *allowed-unit-classes dimensions storage make-parents class* ⇒ *space-instance* [*Function*]

Purpose
Create a new space instance.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
path	A space-instance path specifying the location in the blackboard repository where the new space instance is to be created
initargs	An initialization argument list
allowed-unit-classes	An extended unit-classes specification or `nil` (see below; default is `t`)
dimensions	A list of `(`*dimension-name dimension-type-specifier*`)` pairs (default is `nil`)
storage	A storage specification (see below; default is `(t t unstructured)` or `nil` if *allowed-unit-classes* is `nil`)
make-parents	A generalized boolean (default is `nil`)
class	The name of the space class for the created space instance (default is `standard-space-instance`)
space-instance	A space instance

Returns
The created space instance.

Events
When a space instance is created, events are signaled in the following sequence:

1. An `nonlink-slot-updated-event` or `link-event` is signaled for each slot in the newly created space instance. A `link-event` is also signaled for each inverse pointer from an existing space instance or unit instance to the newly created space instance.
2. An `instance-added-to-space-instance-event` is signaled for each space instance on which the newly created space instance is added.
3. A `instance-created-event` is signaled.

Errors
Use of an initialization argument that has not been declared as valid.

The supplied or generated instance name is identical to the instance name of an existing unit instance of *class*.

Detailed syntax
allowed-unit-classes ::= *unit-classes-specifier* | `nil`

dimension-type-specifier ::= `:ordered` | `(:ordered` [*ordered-comparison-type*]`)` |
`:enumerated` | `(:enumerated` [*enumerated-comparison-type*]`)` |
`:boolean` | `(:boolean` [*boolean-comparison-type*]`)`

ordered-comparison-type ::= `number` | `fixnum` | `short-float` | `single-float` | `double-float` | `long-float` | `pseudo-probability`
enumerated-comparison-type ::= `eq` | `eql` | `equal` | `equalp`
boolean-comparison-type ::= `t`

unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*$^{+}$`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*

storage ::= `(`*unit-class-specifier dimension-names storage-specification*`)`*
dimension-names ::= *dimension-name* | `(`*dimension-name*$^{+}$`)` | `t`
storage-specification ::= `unstructured` | `boolean` | `uniform-buckets :layout` *dimension-buckets-specification*$^{+}$ | `hashed` [`:size` *integer*]
dimension-buckets-specification ::= `(`*start-value end-value bucket-width*`)`

The default *ordered-comparison-type*, if unspecified, is `number`. The default *enumerated-comparison-type*, if unspecified, is `eql`. The default *boolean-comparison-type* is `t`.

Terms

dimension-name A symbol specifying a dimension

Description

Specifying a `:space-instances` initialization argument causes that value to be used instead of any `:initial-space-instances` specification associated with the space class. However, note that placing a space instance on other space instances is unrelated to the layout of blackboard repository hierarchy, which is specified by the `:paths` initialization argument. Placing a space instance on other space instances is no different from placing any other unit instance on a space instance.

See also

allowed-unit-classes-of (page 431)
change-space-instance (page 432)
children-of (page 434)
define-space-class (page 438)
delete-all-space-instances (page 444)
delete-space-instance (page 445)
describe-instance (page 338)
describe-space-instance (page 448)
describe-space-instance-storage (page 449)
dimensions-of (page 344)
make-instance (page 364)
parent-of (page 459)

Examples

Create a top-level space instance, `bb`, that cannot store any unit instances:

```
> (make-space-instance '(bb)
     :allowed-unit-classes nil)
#<standard-space-instance (bb)>
```

```
>
```

Now create a space instance for `hyp` unit instances, named `hyps`, as a child of `bb`, with uniform, 100-wide, bucket storage for indexing unit instances with dimension values between 0–10,000 in the `x` and `y` dimensions:

```
> (make-space-instance '(bb hyps)
     :dimensions (dimensions-of 'hyp)
     :allowed-unit-classes '((hyp :plus-subclasses))
     :storage '(((hyp :plus-subclasses) (x y)
                 uniform-buckets :layout ((0 10000 100)
                                          (0 10000 100)))))
#<standard-space-instance (bb hyps)>
>
```

Here is an improved space instance for `hyp` unit instances named `hyps`, with both uniform-bucket storage for indexing in the `x` and `y` dimensions and hashed storage to retrieve match candidates via `classification` dimension values:

```
> (make-space-instance '(bb hyps)
     :dimensions (dimensions-of 'hyp)
     :allowed-unit-classes '((hyp :plus-subclasses))
     :storage '(((hyp :plus-subclasses) (x y)
                 uniform-buckets :layout ((0 10000 100)
                                          (0 10000 100)))
                ((hyp :plus-subclasses) (classification)
                 hashed)))
#<standard-space-instance (bb hyps)>
>
```

Note

The **make-space-instance** function is equivalent to using **make-instance** with the desired space class and with the initialization argument `:instance-name` containing the space-instance path. However, using **make-space-instance** provides a convenient shorthand and is preferable stylistically.

map-space-instances *function space-instance-regexp* [*Function*]

Purpose

Apply a function once to each space instance that matches a path-expression pattern.

Package :gbbopen

Module :gbbopen-core

Arguments

function	A function designator specifying a function object of one argument
space-instance-regexp	A space-instance path regular expression specifying the space instances to be mapped over

Description

The *space-instance-regexp* argument is either the symbol `t` (indicating all space instances) or a list representing a regular expression where the following reserved symbols are interpreted as follows:

- `=` matches one occurrence in a space-instance path
- `?` matches zero or one occurrence in a space-instance path
- `+` matches one or more occurrences in a space-instance path
- `*` matches zero or more occurrences in a space-instance path
- `^` move to parent

A *space-instance-regexp* value consisting of a list of space instances mapped over as supplied.

See also

do-space-instances (page 450)
find-space-instances (page 453)

Example

Remove all `hyp` unit instances from space instances that are rooted at `(bb)`:

```
  (map-space-instances
    #'(lambda (space-instance)
        (map-instances-on-space-instances
          #'(lambda (unit-instance)
              (remove-instance-from-space-instance unit-instance
space-instance))
          'hyp
          space-instance))
    '(bb +))
```

parent-of *space-instance* ⇒ *space-instances* [*Generic Function*]

Purpose
Return the parent space instance of a space instance.

Method signatures
parent-of (*space-instance* `standard-space-instance`) ⇒ *space-instance*

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
space-instance A space instance
space-instances A proper list

Returns
The parent space instance or `nil`, if *space-instance* does not have a parent.

See also
children-of (page 434)
make-space-instance (page 455)

Examples
Return the parent space instance of the `(bb hyp)` space instance:

```
> (parent-of (find-space-instance-by-path '(bb hyp))
#<standard-space-instance (bb)>
>
```

Return the parent space instance (there is none) of the `(bb)` space instance:

```
> (parent-of (find-space-instance-by-path '(bb))
nil
>
```

remove-instance-from-space-instance *unit-instance space-instance* [*Generic Function*]

Purpose

Remove a unit instance from a space instance.

Method signatures

remove-instance-from-space-instance (*unit-instance* `standard-unit-instance`)(*space-instance-path* `cons`)

remove-instance-from-space-instance (*unit-instance* `standard-unit-instance`)(*space-instance* `standard-space-instance`)

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

unit-instance The unit instance to be removed

space-instance-or-path The space instance or space-instance path from which the unit instance is to be removed

Events

A `instance-removed-from-space-instance-event` is signaled.

See also

add-instance-to-space-instance (page 430)

Examples

Remove an incorrect hypothesis unit instance, `incorrect-hyp`, from the `hyps` space instance:

```
> (remove-instance-from-space-instance
    incorrect-hyp (find-space-instance-by-path '(bb hyps)))
#<hyp 311 (896 388) 0.68 [0..6]>
>
```

or

```
> (remove-instance-from-space-instance incorrect-hyp '(bb hyps))
#<hyp 311 (896 388) 0.68 [0..6]>
>
```

reset-gbbopen `&key` *disable-events* [*Function*]

Purpose
Unconditionally delete all unit and space instances, remove all event functions, and disable event printing.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
disable-events A generalized boolean (default is `t`)

Events
If *disable-events* is `nil`, the following events may be signaled (in order) as unit instances and space instances are deleted:

- `delete-instance-event`
- `unlink-event`
- `instance-removed-from-space-instance-event`
- `instance-deleted-event`

Description
Calling **reset-gbbopen** deletes all unit instances and space instances, disables all event printing, removes all event functions, and resets all unit-class instance-name counters to their initial values. **Reset-gbbopen** does not undefine any class definitions, functions, methods, etc.

See also

confirm-if-blackboard-repository-not-empty-p (page 436)
delete-all-space-instances (page 444)
delete-instance (page 334)
delete-space-instance (page 445)
define-ks-class (page 591)
define-ksa-class (page 595)
define-space-class (page 438)
define-unit-class (page 330)
initial-class-instance-number (page 352)
reset-gbbopen (page 461)
reset-unit-class (page 367)

Example
Prepare for a new application by resetting GBBopen:

```
> (reset-gbbopen)
t
>
```

Note

This function and **delete-blackboard-repository** are the only GBBopen functions that disable event signaling by default. This conflicts with the normal use of **with-events-disabled** and **with-events-enabled** macros for controlling event signaling, but having events disabled is the desired behavior in almost every reset situation.

---

reset-gbbopen

standard-space-class [*Space Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **standard-space-class** is the default class of space classes defined by **define-space-class**. It is a subclass of **standard-unit-class**.

See also

standard-space-instance (page 464)
standard-unit-class (page 369)

standard-space-instance [*Unit Class*]

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

The class **standard-space-instance** is the default class of instances created by **make-space-instance**. A space instance is also a unit instance, so **standard-space-instance** is a subclass of **standard-unit-instance**.

See also

deleted-unit-instance (page 337)
print-instance-slots (page 107)
standard-space-class (page 463)
standard-unit-instance (page 370)

with-blackboard-repository-locked `(&key` *whostate*`)` *form** ⇒ *result** [*Macro*]

Purpose
After locking the blackboard repository, execute forms and then release the lock.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
whostate A string (default `"With Blackboard Repository Locked"`)
forms An implicit **progn** of forms to be evaluated
results The values returned by evaluating the last *form*

Returns
The values returned by evaluating the last *form*.

See also
thread-whostate (page 259)

Example
Lock the blackboard repository and do some stuff:

```
(with-blackboard-repository-locked ()
  (do-some-stuff))
```

Note
The *whostate* value is ignored by SBCL.

5.5 Instance Retrieval

This section contains `:gbbopen-core` entities that pertain to unit-instance retrieval and unit-instance iteration and mapping.

find-verbose [*Variable*]

Purpose

Controls the printing of retrieval and matching information by **do-instances-on-space-instances**, **filter-instances**, **find-instances**, and **map-instances-on-space-instances** when the keyword argument `:verbose` has not been specified.

Package `:gbbopen`

Module `:gbbopen-core`

Value type A generalized boolean

Initial value `nil`

See also

do-instances-on-space-instances (page 472)
filter-instances (page 476)
find-instances (page 483)
map-instances-on-space-instances (page 493)

use-marking [*Variable*]

Purpose

Controls the use of instance marking versus ESETs for retrieval and matching by **do-instances-on-space-instances**, **find-instances**, and **map-instances-on-space-instances** when the keyword argument `:use-marking` has not been specified.

Package `:gbbopen`

Module `:gbbopen-core`

Value type A generalized boolean

Initial value `t`

See also

do-instances-on-space-instances (page 472)
filter-instances (page 476)
find-instances (page 483)
map-instances-on-space-instances (page 493)

warn-about-unusual-requests [*Variable*]

Purpose

Control warning messages of "unusual" **do-instances-on-space-instances**, **find-instances**, and **map-instances-on-space-instances** requests that are likely to be mistakes.

Package

`:gbbopen`

Module

`:gbbopen-core`

Value type

A generalized boolean

Initial value

True

See also

do-instances-on-space-instances (page 472)
filter-instances (page 476)
find-instances (page 483)
map-instances-on-space-instances (page 493)

Example

Suppress the warning message associated with an unachievable retrieval pattern:

```
> (filter-instances nil '(and (> x 3) (< x 2)))
;; Warning: Pattern (and (> X 3) (< X 2)) can not be satisfied.
nil
> (let ((*warn-about-unusual-requests* nil))
    (filter-instances nil '(and (> x 3) (< x 2))))
nil
>
```

do-instances-of-class (*var unit-classes-specifier) declaration** {*tag* | *form*}* [*Macro*]

Purpose
Iterate over all unit instances of the specified unit classes.

Package :gbbopen

Module :gbbopen-core

Arguments
var	A variable symbol
unit-classes-specifier	An extended unit-classes specification (see below)
declaration	A declare expression (not evaluated)
tag	A `go` tag (not evaluated)
form	A form

Detailed syntax
unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*$^{+}$`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand `+` subclasses specifier is equivalent to `:plus-subclasses` and `=` to `:no-subclasses`.

Description
The iteration over the unit instances of the specified unit classes is performed once for each unit instance, whether or not the instances reside on any space instances.

See also

class-instances-count (page 329)
clear-space-instances (page 435)
do-instances-on-space-instances (page 472)
find-instances-of-class (page 488)
make-instances-of-class-vector (page 490)
map-instances-of-class (page 491)
map-instances-on-space-instances (page 493)
map-sorted-instances-of-class (page 496)

Examples
Delete all unit instances of the unit class `hyp`:

```
(do-instances-of-class (instance 'hyp)
  (delete-instance instance))
```

Delete all unit instances of the unit class `hyp` and instances of subclasses of `hyp`:

```
(do-instances-of-class (instance '(hyp :plus-subclasses))
  (delete-instance instance))
```

or simply:

```
(do-instances-of-class (instance '(hyp +))
  (delete-instance instance))
```

Note

The consequences are unspecified if an attempt is made to add or delete a unit instance while **do-instances-of-class** is in progress. There is one exception to this restriction: the current unit instance in the iteration (bound to *var*) can be deleted, provided that the deletion does not trigger the deletion of any other unit instances. For example, the following form intended to delete all space instances violates this restriction:

```
(do-instances-of-class (space-instance
                        '(standard-space-instance :plus-subclasses))
  (delete-space-instance space-instance))
```

because deletion of a space instance with children automatically deletes those child space instances. The function **delete-all-space-instances** provides an efficient means of deleting all space instances without violating this rule.

REPL Note

The equivalent of `(do-instances-of-class (instance` *arg*`) (print instance))` can be invoked using the REPL command `:pic` [*arg*]. If *arg* is omitted, `t` is assumed.

do-instances-on-space-instances (*var unit-classes-specifier space-instances* `&key` *pattern filter-before filter-after use-marking verbose*) {*tag* | *form*}* [*Macro*]

Purpose

Iterate over each unit instance on space instances, optionally selected by a retrieval pattern.

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

var	A variable symbol
unit-classes-specifier	An extended unit-classes specification (see below)
space-instances	A space instance, a list of space instances, a space-instance path regular expression, or `t` (indicating all space instances)
pattern	A retrieval pattern (see below; default is `t`)
filter-before	A single-argument predicate to be applied before pattern-matching tests occur
filter-after	A single-argument predicate to be applied after pattern-matching tests occur
use-marking	A generalized boolean (default is ***use-marking***)
verbose	A generalized boolean (default is ***find-verbose***)
declaration	A declare expression (not evaluated)
tag	A `go` tag (not evaluated)
form	A form

Detailed syntax

unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*+`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand `+` subclasses specifier is equivalent to `:plus-subclasses` and `=` to `:no-subclasses`.

pattern ::= *subpattern* | `t` | `:all`
subpattern ::= *pattern-element* |
`(not` *subpattern*`)` |
`(and` *subpattern*`*)` |
`(or` *subpattern*`*)`
pattern-element ::= `(`*pattern-op dimension-names pattern-values option*`*)` |
`(`*boolean-dimension-unary-pattern-op dimension-names option*`*)`
pattern-op ::= *ordered-dimension-pattern-op* |
enumerated-dimension-pattern-op |
boolean-dimension-binary-pattern-op
ordered-dimension-pattern-op ::= *ordered-dimension-any-numeric-value-pattern-op* |
ordered-dimension-explicit-type-pattern-op

ordered-dimension-explicit-type-pattern-op ::= *ordered-dimension-fixnum-pattern-op* |
ordered-dimension-short-float-pattern-op |
ordered-dimension-single-float-pattern-op |
ordered-dimension-double-float-pattern-op |
ordered-dimension-long-float-pattern-op |
ordered-dimension-pseudo-probability-pattern-op
ordered-dimension-any-numeric-value-pattern-op ::= `<` | `<=` | `>=` | `>` | `=` | `/=` |
`within` | `covers` | `overlaps` |
`abuts` | `starts` | `ends`
ordered-dimension-fixnum-pattern-op ::= `<&` | `<=&` | `>=&` | `>&` | `=&` | `/=&` |
`within&` | `covers&` | `overlaps&` |
`abuts&` | `starts&` | `ends&`
ordered-dimension-short-float-pattern-op ::= `<$&` | `<=$&` | `>=$&` | `>$&` | `=$&` | `/=$&` |
`within$&` | `covers$&` | `overlaps$&` |
`abuts$&` | `starts$&` | `ends$&`
ordered-dimension-single-float-pattern-op ::= `<$` | `<=$` | `>=$` | `>$` | `=$` | `/=$` |
`within$` | `covers$` | `overlaps$` |
`abuts$` | `starts$` | `ends$`
ordered-dimension-double-float-pattern-op ::= `<$$` | `<=$$` | `>=$$` | `>$$` | `=$$` | `/=$$` |
`within$$` | `covers$$` | `overlaps$$` |
`abuts$$` | `starts$$` | `ends$$`
ordered-dimension-long-float-pattern-op ::= `<$$$` | `<=$$$` | `>=$$$` | `>$$$` | `=$$$` | `/=$$$` |
`within$$$` | `covers$$$` | `overlaps$$$` |
`abuts$$$` | `starts$$$` | `ends$$$`
ordered-dimension-pseudo-probability-pattern-op ::= `<%` | `<=%` | `>=%` | `>%` | `=%` | `/=%` |
`within%` | `covers%` | `overlaps%` |
`abuts%` | `starts%` | `ends%`
enumerated-dimension-pattern-op ::= `is` | *enumerated-dimension-explicit-test-pattern-op*
enumerated-dimension-explicit-test-pattern-op ::= `is-eq` | `is-eql` | `is-equal` | `is-equalp`
boolean-dimension-binary-pattern-op ::= `eqv`
boolean-dimension-unary-pattern-op ::= `true` | `false`
dimension-names ::= *dimension-name* | `(`*dimension-name*$^{+}$`)`
pattern-values ::= *pattern-value* |
`(`*pattern-value*$^{+}$`)` |
`(`*pattern-value*$^{+}$ `.` *pattern-value*`)` |
`#(`*pattern-value*$^{+}$`)`
pattern-value ::= *point* | *interval* | *element* | *set*
interval ::= `(`*start end*`)` | `(`*start* `.` *end*`)` | `#(`*start end*`)`

Terms

point A number, `infinity`, or `-infinity`
start A number or `-infinity`
end A number or `infinity`
element An object

Description

The iteration is performed only once for each selected unit instance, even if the unit instance resides on multiple space instances.

The *pattern* `t` matches all unit instances whose dimension values overlap the dimensional extent of at least one space instance in *space-instances*. The *pattern* `:all` matches every unit instance on a

space instance in *space-instances*, regardless of dimensional overlap.

Declared numeric (see page 143) and pseudo probability (see page 149) pattern operators are also supported, for example: `=&`, `=$&`, `=$`, `=$$`, `=$$$`, and `=%` and `within&`, `within$&`, `within$`, `within$$`, `within$$$`, and `within%`.

See also

find-verbose (page 467)
use-marking (page 468)
warn-about-unusual-requests (page 469)
do-instances-of-class (page 470)
find-instances (page 483)
find-instances-of-class (page 488)
make-instances-of-class-vector (page 490)
map-instances-of-class (page 491)
map-instances-on-space-instances (page 493)
with-find-stats (page 498)
Declared numerics (page 143)

Examples

Remove all the `hyp` unit instances that reside on the `(bb probable-hyps)` space instance, deleting those unit instances that do not reside on any other space instance:

```
(let ((space-instance
         (find-space-instance-by-path '(bb probable-hyps))))
  (do-instances-on-space-instances (instance 'hyp space-instance)
     (if (>= (length (space-instances-of instance) 1))
         (remove-instance-from-space-instance
            instance space-instance)
         (delete-instance instance))))
```

Delete `hyp` unit instances that reside on the `(bb probable-hyps)` space instance that have a belief value less than 0.5:

```
(do-instances-on-space-instances (instance 'hyp '(bb probable-hyps)
                                  :pattern '(< belief .5))
   (delete-instance instance))
```

Note

Fixnum `overlaps` comparisons can result in bignum computations if the combined intervals of the pattern and a candidate unit instance exceeds `most-positive-fixnum`.

---

do-instances-on-space-instances

do-sorted-instances-of-class (*var unit-classes-specifier predicate* &key *key*) {*tag* | *form*}* [*Macro*]

Purpose
Iterate over each unit instance of the specified unit classes, in sorted order.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
var	A variable symbol
unit-classes-specifier	An extended unit-classes specification (see below)
predicate	A function designator specifying a function object of two arguments that returns a generalized boolean
key	A function designator specifying a function object of one argument, or `nil` (default is `nil`)
declaration	A declare expression (not evaluated)
tag	A `go` tag (not evaluated)
form	A form

Detailed syntax
unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*+`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
The iteration is performed once for each unit instance of the specified unit classes, whether or not the instances reside on any space instances.

See also

do-instances-of-class (page 470)
find-instances-of-class (page 488)
make-instances-of-class-vector (page 490)
map-instances-of-class (page 491)
map-instances-on-space-instances (page 493)
map-sorted-instances-of-class (page 496)

Example
Print a list of all `hyp` instance names, in ascending order:

```
(do-sorted-instances-of-class (instance 'hyp #'< :key #'instance-name-of)
   (print (instance-name-of instance)))
```

filter-instances *unit-instances pattern* `&key` *filter-before filter-after verbose* [*Function*]
⇒ *matching-unit-instances*

Purpose
Select matching unit instances from a list of unit instances based on a retrieval pattern.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-instances	A list of unit instances
pattern	A retrieval pattern (see below)
filter-before	A single-argument predicate to be applied before pattern-matching tests occur
filter-after	A single-argument predicate to be applied after pattern-matching tests occur
verbose	A generalized boolean (default is ***find-verbose***)
matching-unit-instances	A proper list of unit instances

Returns
A newly consed list of unit instances that satisfy the specified pattern and predicate filters.

Detailed syntax
pattern ::= *subpattern* | `t`
subpattern ::= *pattern-element* |
`(not` *subpattern*`)` |
`(and` *subpattern*`*)` |
`(or` *subpattern*`*)`
pattern-element ::= `(`*pattern-op dimension-names pattern-values option*`*)` |
`(`*boolean-dimension-unary-pattern-op dimension-names option*`*)`
pattern-op ::= *ordered-dimension-pattern-op* |
enumerated-dimension-pattern-op |
boolean-dimension-binary-pattern-op
ordered-dimension-pattern-op ::= *ordered-dimension-any-numeric-value-pattern-op* |
ordered-dimension-explicit-type-pattern-op
ordered-dimension-explicit-type-pattern-op ::= *ordered-dimension-fixnum-pattern-op* |
ordered-dimension-short-float-pattern-op |
ordered-dimension-single-float-pattern-op |
ordered-dimension-double-float-pattern-op |
ordered-dimension-long-float-pattern-op |
ordered-dimension-pseudo-probability-pattern-op
ordered-dimension-any-numeric-value-pattern-op ::= `<` | `<=` | `>=` | `>` | `=` | `/=` |
`within` | `covers` | `overlaps` |
`abuts` | `starts` | `ends`
ordered-dimension-fixnum-pattern-op ::= `<&` | `<=&` | `>=&` | `>&` | `=&` | `/=&` |
`within&` | `covers&` | `overlaps&` |
`abuts&` | `starts&` | `ends&`

ordered-dimension-short-float-pattern-op ::= `<$&` | `<=$&` | `>=$&` | `>$&` | `=$&` | `/=$&` | `within$&` | `covers$&` | `overlaps$&` | `abuts$&` | `starts$&` | `ends$&`
ordered-dimension-single-float-pattern-op ::= `<$` | `<=$` | `>=$` | `>$` | `=$` | `/=$` | `within$` | `covers$` | `overlaps$` | `abuts$` | `starts$` | `ends$`
ordered-dimension-double-float-pattern-op ::= `<$$` | `<=$$` | `>=$$` | `>$$` | `=$$` | `/=$$` | `within$$` | `covers$$` | `overlaps$$` | `abuts$$` | `starts$$` | `ends$$`
ordered-dimension-long-float-pattern-op ::= `<$$$` | `<=$$$` | `>=$$$` | `>$$$` | `=$$$` | `/=$$$` | `within$$$` | `covers$$$` | `overlaps$$$` | `abuts$$$` | `starts$$$` | `ends$$$`
ordered-dimension-pseudo-probability-pattern-op ::= `<%` | `<=%` | `>=%` | `>%` | `=%` | `/=%` | `within%` | `covers%` | `overlaps%` | `abuts%` | `starts%` | `ends%`
enumerated-dimension-pattern-op ::= `is` | *enumerated-dimension-explicit-test-pattern-op*
enumerated-dimension-explicit-test-pattern-op ::= `is-eq` | `is-eql` | `is-equal` | `is-equalp`
boolean-dimension-binary-pattern-op ::= `eqv`
boolean-dimension-unary-pattern-op ::= `true` | `false`
dimension-names ::= *dimension-name* | `(`*dimension-name*$^{+}$`)`
pattern-values ::= *pattern-value* | `(`*pattern-value*$^{+}$`)` | `(`*pattern-value*$^{+}$ `.` *pattern-value*`)` | `#(`*pattern-value*$^{+}$`)`
pattern-value ::= *point* | *interval* | *element* | *set*
interval ::= `(`*start end*`)` | `(`*start* `.` *end*`)` | `#(`*start end*`)`

Terms

point A number, `infinity`, or `-infinity`
start A number or `-infinity`
end A number or `infinity`
element An object

Description

If a unit instance appears more than once in *unit-instances*, it will be checked for selection—and potentially included in the result—multiple times.

Declared numeric (see page 143) and pseudo probability (see page 149) pattern operators are also supported, for example: `=&`, `=$&`, `=$`, `=$$`, `=$$$`, and `=%` and `within&`, `within$&`, `within$`, `within$$`, `within$$$`, and `within%`.

See also

find-verbose (page 467)
warn-about-unusual-requests (page 469)
find-all-instances-by-name (page 479)
find-instance-by-name (page 481)
find-instances (page 483)
map-instances-on-space-instances (page 493)
Declared numerics (page 143)

Examples

```
  > (filter-instances (supporting-hyps-of hyp) '(> belief .8))
  (#<hyp 183 (1835 4791) 0.82 [0..35]>
   #<hyp 233 (1835 4791) 0.89 [5..35]>
   #<hyp 419 (1835 4791) 0.85 [5..35]>)
  > (filter-instances (supporting-hyps-of hyp) `(within belief (.85
,infinity)))
  (#<hyp 233 (1835 4791) 0.89 [5..35]>
   #<hyp 419 (1835 4791) 0.85 [5..35]>)
  >
```

Note

Fixnum `overlaps` comparisons can result in bignum computations if the combined intervals of the pattern and a candidate unit instance exceeds `most-positive-fixnum`.

---

filter-instances

find-all-instances-by-name *instance-name* `&optional` *unit-class-specifier* [*Function*]
⇒ *unit-instances*

Purpose
Retrieve unit instances with a given *name*.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
instance-name	The instance name for the retrieval
unit-class-specifier	An extended unit-class specification (see below; default is `t`)
unit-instances	A proper list of unit instances

Returns
A list of the unit instances with the specified *name* of the specified *classes* if any exist; `nil` otherwise.

Detailed syntax
unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
The `:instance-name-comparison-test` function (`eq`, `eql`, `equal`, or `equalp`) specified in **define-unit-class** is used to match *instance-name* with the unit-instance's instance name. If you are using strings as the names of unit instances, you should specify `equal` (case sensitive) or `equalp` (case insensitive) as the comparison function in the unit classes of those unit instances.

See also

define-unit-class (page 330)
filter-instances (page 476)
find-all-instances-by-name (page 479)
find-instance-by-name (page 481)
find-instances (page 483)

Examples
Find all unit instances (of any unit class) that are named 419:

```
> (find-instances-by-name 419 't)
(#<hyp 419 (1835 4791) 0.85 [5..35]>)
>
```

Find all unit instances (of any unit class) that are named 419:

```
> (find-instances-by-name 419 '(hyp :plus-subclasses))
(#<hyp 419 (1835 4791) 0.85 [5..35]>)
>
```

or simply:

```
> (find-instances-by-name 419 '(hyp +))
(#<hyp 419 (1835 4791) 0.85 [5..35]>)
>
```

---

find-all-instances-by-name

find-instance-by-name *instance-name* `&optional` *unit-class-specifier errorp* [*Function*]
⇒ *unit-instance*

Purpose
Retrieve a unit instance by its instance name.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
instance-name	The instance name for the retrieval
unit-class-specifier	An extended unit-class specification (see below; default is `t`)
errorp	A generalized boolean (default is `nil`)
unit-instance	A unit instance or `nil`

Returns
The first unit instance found of the specified *class(es)* that has the specified *instance-name* if one exists; `nil` otherwise.

Errors
If *errorp* is true, an error is signaled if no unit instance is found.

Detailed syntax
unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
The `:instance-name-comparison-test` function (`eq`, `eql`, `equal`, or `equalp`) specified in **define-unit-class** is used to match *instance-name* with the unit-instance's instance name. When strings are used as the names of unit instances, `equal` (case sensitive) or `equalp` (case insensitive) should be specified as the comparison function in the unit classes of those unit instances.

See also

define-unit-class (page 330)
filter-instances (page 476)
find-instances (page 483)
find-all-instances-by-name (page 479)

Example
Find the `hyp` unit instance 419:

```
> (find-instance-by-name 419 'hyp)
#<hyp 419 (1835 4791) 0.85 [5..35]>
> (find-instance-by-name 0 'hyp)
nil
```

```
> (find-instance-by-name 0 'hyp 't)
No unit instance named 0 of class hyp was found.
>
```

REPL Note

Find-instance-by-name can be invoked using the REPL command:

`:fi` *instance-name* [*unit-classes-specifier* [*errorp*]]

which sets = to the found unit instance.

---

find-instance-by-name

find-instances *unit-classes-specifier space-instances pattern* [*Function*]
`&key` *filter-before filter-after use-marking verbose* ⇒ *unit-instances*

Purpose
Retrieve unit instances from space instances based on a retrieval pattern.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
unit-classes-specifier An extended unit-classes specification (see below)

space-instances	A space instance, a list of space instances, a space-instance path regular expression, or `t` (indicating all space instances)
pattern	A retrieval pattern (see below)
filter-before	A single-argument predicate to be applied before pattern-matching tests occur
filter-after	A single-argument predicate to be applied after pattern-matching tests occur
use-marking	A generalized boolean (default is ***use-marking***)
verbose	A generalized boolean (default is ***find-verbose***)
unit-instances	A proper list of unit instances

Returns
A newly consed list of unit instances specified by *unit-class-specifier* that reside on *space-instances* and satisfy the specified *pattern* and any predicate filters.

Detailed syntax
unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*$^{+}$`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand `+` subclasses specifier is equivalent to `:plus-subclasses` and `=` to `:no-subclasses`.

pattern ::= *subpattern* | `t` | `:all`
subpattern ::= *pattern-element* |
`(not` *subpattern*`)` |
`(and` *subpattern**`)` |
`(or` *subpattern**`)`
pattern-element ::= `(`*pattern-op dimension-names pattern-values option**`)` |
`(`*boolean-dimension-unary-pattern-op dimension-names option**`)`
pattern-op ::= *ordered-dimension-pattern-op* |
enumerated-dimension-pattern-op |
boolean-dimension-binary-pattern-op
ordered-dimension-pattern-op ::= *ordered-dimension-any-numeric-value-pattern-op* |
ordered-dimension-explicit-type-pattern-op

ordered-dimension-explicit-type-pattern-op ::= *ordered-dimension-fixnum-pattern-op* |
ordered-dimension-short-float-pattern-op |
ordered-dimension-single-float-pattern-op |
ordered-dimension-double-float-pattern-op |
ordered-dimension-long-float-pattern-op |
ordered-dimension-pseudo-probability-pattern-op
ordered-dimension-any-numeric-value-pattern-op ::= `<` | `<=` | `>=` | `>` | `=` | `/=` |
`within` | `covers` | `overlaps` |
`abuts` | `starts` | `ends`
ordered-dimension-fixnum-pattern-op ::= `<&` | `<=&` | `>=&` | `>&` | `=&` | `/=&` |
`within&` | `covers&` | `overlaps&` |
`abuts&` | `starts&` | `ends&`
ordered-dimension-short-float-pattern-op ::= `<$&` | `<=$&` | `>=$&` | `>$&` | `=$&` | `/=$&` |
`within$&` | `covers$&` | `overlaps$&` |
`abuts$&` | `starts$&` | `ends$&`
ordered-dimension-single-float-pattern-op ::= `<$` | `<=$` | `>=$` | `>$` | `=$` | `/=$` |
`within$` | `covers$` | `overlaps$` |
`abuts$` | `starts$` | `ends$`
ordered-dimension-double-float-pattern-op ::= `<$$` | `<=$$` | `>=$$` | `>$$` | `=$$` | `/=$$` |
`within$$` | `covers$$` | `overlaps$$` |
`abuts$$` | `starts$$` | `ends$$`
ordered-dimension-long-float-pattern-op ::= `<$$$` | `<=$$$` | `>=$$$` | `>$$$` | `=$$$` | `/=$$$` |
`within$$$` | `covers$$$` | `overlaps$$$` |
`abuts$$$` | `starts$$$` | `ends$$$`
ordered-dimension-pseudo-probability-pattern-op ::= `<%` | `<=%` | `>=%` | `>%` | `=%` | `/=%` |
`within%` | `covers%` | `overlaps%` |
`abuts%` | `starts%` | `ends%`
enumerated-dimension-pattern-op ::= `is` | *enumerated-dimension-explicit-test-pattern-op*
enumerated-dimension-explicit-test-pattern-op ::= `is-eq` | `is-eql` | `is-equal` | `is-equalp`
boolean-dimension-binary-pattern-op ::= `eqv`
boolean-dimension-unary-pattern-op ::= `true` | `false`
dimension-names ::= *dimension-name* | `(`*dimension-name*$^{+}$`)`
pattern-values ::= *pattern-value* |
`(`*pattern-value*$^{+}$`)` |
`(`*pattern-value*$^{+}$ `.` *pattern-value*`)` |
`#(`*pattern-value*$^{+}$`)`
pattern-value ::= *point* | *interval* | *element* | *set*
interval ::= `(`*start end*`)` | `(`*start* `.` *end*`)` | `#(`*start end*`)`

Terms

point A number, `infinity`, or `-infinity`
start A number or `-infinity`
end A number or `infinity`
element An object

Description

The *pattern* `t` matches all unit instances whose dimension values overlap the dimensional extent of at least one space instance in *space-instances*. The *pattern* `:all` matches every unit instance on a space instance in *space-instances*, regardless of dimensional overlap. Use of **find-instances** with a `t` or `:all` *pattern* in production code often indicates a missed opportunity to use a more efficient **do-instances-of-class**, **do-instances-on-space-instances**, or **do-sorted-instances-of-class**

approach (or their `map-` variants).

Declared numeric (see page 143) and pseudo probability (see page 149) pattern operators are also supported, for example: `=&`, `=$&`, `=$`, `=$$`, `=$$$`, and `=%` and `within&`, `within$&`, `within$`, `within$$`, `within$$$`, and `within%`.

See also

find-verbose (page 467)
use-marking (page 468)
warn-about-unusual-requests (page 469)
do-instances-of-class (page 470)
do-instances-on-space-instances (page 472)
do-sorted-instances-of-class (page 475)
filter-instances (page 476)
find-all-instances-by-name (page 479)
find-instance-by-name (page 481)
find-instances-of-class (page 488)
make-instances-of-class-vector (page 490)
map-instances-of-class (page 491)
map-instances-on-space-instances (page 493)
map-sorted-instances-of-class (page 496)
Declared numerics (page 143)

Examples

Here are some basic examples of finding `hyp` unit instances:

```
> (find-instances 'hyp (find-space-instance-by-path '(bb hyps))
    '(and (= x 1835) (> belief .8)))
(#<hyp 319 (1835 8419) 0.91 [4..12]>
 #<hyp 331 (1835 8419) 0.88 [15..30]>
 #<hyp 335 (1835 8419) 0.92 [15..35]>
 #<hyp 183 (1835 4791) 0.82 [0..35]>
 #<hyp 233 (1835 4791) 0.89 [5..35]>
 #<hyp 419 (1835 4791) 0.85 [5..35]>)
> (find-instances 'hyp '(bb hyps)
    `(and (= x 1835)
          (> belief ,(belief-of (find-instance-by-name 331 'hyp)))))
(#<hyp 319 (1835 8419) 0.91 [4..12]>
 #<hyp 335 (1835 8419) 0.92 [15..35]>
 #<hyp 233 (1835 4791) 0.89 [5..35]>)
> (find-instances '(hyp :no-subclasses) '(bb hyps)
    '(=& (x y) (1835 8419)))
(#<hyp 319 (1835 8419) 0.91 [4..12]>
 #<hyp 331 (1835 8419) 0.88 [15..30]>
 #<hyp 335 (1835 8419) 0.92 [15..35]>)
> (find-instances 'hyp '(bb hyps)
    '(and (= x 1835) (within belief (.85 .9))))
(#<hyp 331 (1835 8419) 0.88 [15..30]>
 #<hyp 233 (1835 4791) 0.89 [5..35]>
 #<hyp 419 (1835 4791) 0.85 [5..35]>)
```

```
> (find-instances 'hyp '(bb hyps)
    '(and (= x 1835) (overlaps velocity-range (0 10))))
(#<hyp 319 (1835 8419) 0.91 [4..12]>
 #<hyp 183 (1835 4791) 0.82 [0..35]>
 #<hyp 233 (1835 4791) 0.89 [5..35]>
 #<hyp 419 (1835 4791) 0.85 [5..35]>)
> (find-instances 'hyp '(bb hyps)
    '(and (= x 1835) (overlaps velocity-range (35 40))))
(#<hyp 335 (1835 8419) 0.92 [15..35]>
 #<hyp 183 (1835 4791) 0.82 [0..35]>
 #<hyp 233 (1835 4791) 0.89 [5..35]>
 #<hyp 419 (1835 4791) 0.85 [5..35]>)
> (find-instances 'hyp '(bb hyps)
    '(and (= x 1835) (covers velocity-range (0 10))))
(#<hyp 183 (1835 4791) 0.82 [0..35]>)
> (find-instances 'hyp '(bb hyps)
    '(and (= x 1835) (covers velocity-range (4 10))))
(#<hyp 319 (1835 8419) 0.91 [4..12]>
 #<hyp 183 (1835 4791) 0.82 [0..35]>)
> (find-instances 'hyp '(bb hyps)
    '(and (= x 1835) (within velocity-range (0 10))))
nil
> (find-instances 'hyp '(bb hyps)
    '(and (= x 1835) (within velocity-range (0 20))))
(#<hyp 319 (1835 8419) 0.91 [4..12]>)
> (find-instances 'hyp '(bb hyps)
    '(and (= x 1835) (starts velocity-range 5)))
(#<hyp 233 (1835 4791) 0.89 [5..35]>
 #<hyp 419 (1835 4791) 0.85 [5..35]>)
> (find-instances 'hyp '(bb hyps)
    '(and (= x 1835) (ends velocity-range 30)))
(#<hyp 331 (1835 8419) 0.88 [15..30]>)
> (find-instances 'hyp '(bb hyps)
    '(and (= x 1835) (abuts velocity-range (5 30))))
(#<hyp 233 (1835 4791) 0.89 [5..35]>
 #<hyp 331 (1835 8419) 0.88 [15..30]>
 #<hyp 419 (1835 4791) 0.85 [5..35]>)
> (find-instances 'hyp '(bb hyps)
    '(is color :silver))
(#<hyp 183 (1835 4791) 0.82 [0..35]>
 #<hyp 233 (1835 4791) 0.89 [5..35]>
 #<hyp 231 (1488 7405) 0.63 [0..8]>)
> (find-instances 'hyp '(bb hyps)
    '(is-eq color :yellow))
(#<hyp 331 (1835 8419) 0.88 [15..30]>
 #<hyp 311 (896 388) 0.68 [0..6]>)
>
```

Some examples finding a `word` unit instance:

```
> (find-instances 'word '(words) '(is character #\s))
(#<word 1>)
```

```
> (find-instances 'word '(words) '(is character #\x))
nil
>
```

Find `word` unit instances with special characters in them:

```
> (find-instances 'word '(words) '(overlaps char-code (0 64)))
(#<word 1>)
>
```

Note

Fixnum `overlaps` comparisons can result in bignum computations if the combined intervals of the pattern and a candidate unit instance exceeds `most-positive-fixnum`.

---

find-instances

find-instances-of-class *unit-classes-specifier* ⇒ *unit-instances* [*Function*]

Purpose
Return a list of all unit instances of the specified unit classes.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-classes-specifier An extended unit-classes specification (see below)
unit-instances A proper list of unit instances

Returns
A newly consed list of all unit instances of the unit classes specified by *unit-class-specifier*, whether or not they reside on any space instance.

Detailed syntax
unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*$^{+}$`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

See also

class-instances-count (page 329)
do-instances-of-class (page 470)
make-instances-of-class-vector (page 490)
map-instances-of-class (page 491)
map-instances-on-space-instances (page 493)
map-sorted-instances-of-class (page 496)

Example
Return the list of all `hyp` unit instances:

```
> (find-instances-of-class 'hyp)
(#<hyp 233 (1835 4791) 0.89 [5..35]>
 #<hyp 319 (1835 8419) 0.91 [4..12]>
 #<hyp 419 (1835 4791) 0.85 [5..35]>
 #<hyp 231 (1488 7405) 0.63 [0..8]>
 #<hyp 311 (896 388) 0.68 [0..6]>
   ...
 #<hyp 331 (1835 8419) 0.88 [15..30]>
 #<hyp 183 (1835 4791) 0.82 [0..35]>
 #<hyp 335 (1835 8419) 0.92 [15..35]>)
>
```

Note

In general, **do-instances-of-class** or **map-instances-of-class** is preferred over operating on the list created by **find-instances-of-class**.

---

find-instances-of-class

make-instances-of-class-vector *function* `&key` *adjustable* ⇒ *vector* [*Function*]

Purpose
Return a (newly allocated) vector containing the unit instances of the specified unit classes.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
adjustable A generalized boolean (default is `nil`)
vector A vector (optionally adjustable) with a fill pointer

Returns
The vector containing the unit instances.

Detailed syntax
unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*$^{+}$`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

See also
do-instances-of-class (page 470)
find-instances-of-class (page 488)
map-instances-of-class (page 491)
map-sorted-instances-of-class (page 496)

Example
Create a vector containing all unit instances of the unit class `hyp` and subclasses of `hyp`:

```
(make-instances-of-class-vector '(hyp :plus-subclasses))
```

or simply:

```
(make-instances-of-class-vector '(hyp +))
```

map-instances-of-class *function unit-classes-specifier* [*Function*]

Purpose
Apply a function once to each unit instance of the specified unit classes.

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
function	A function designator specifying a function object of one argument
unit-classes-specifier	An extended unit-classes specification (see below)

Detailed syntax
unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*$^{+}$`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
The specified function is applied once to each unit instance of the specified unit classes, whether or not the instances reside on any space instances.

See also

class-instances-count (page 329)
clear-space-instances (page 435)
do-instances-of-class (page 470)
find-instances-of-class (page 488)
make-instances-of-class-vector (page 490)
map-instances-on-space-instances (page 493)
map-sorted-instances-of-class (page 496)

Examples
Delete all unit instances of the unit class `hyp`:

```
(map-instances-of-class #'delete-instance 'hyp)
```

Delete all unit instances of the unit class `hyp` and instances of subclasses of `hyp`:

```
(map-instances-of-class #'delete-instance '(hyp :plus-subclasses))
```

or simply:

```
(map-instances-of-class #'delete-instance '(hyp +))
```

Note

The consequences are unspecified if an attempt is made to add or delete a unit instance while **map-instances-of-class** is in progress. There is one exception to this restriction: *function* may delete its unit instance argument, provided that deletion does not trigger the deletion of any other unit instances. For example, the following form intended to delete all space instances violates this restriction:

```
(map-instances-of-class
  #'delete-space-instance
  '(standard-space-instance :plus-subclasses))
```

because deletion of a space instance with children automatically deletes those child space instances. The function **delete-all-space-instances** provides an efficient means of deleting all space instances without violating this rule.

REPL Note

The equivalent of `(map-instances-of-class 'print` *`arg`*`)` can be invoked using the REPL command

`:pic [`*`unit-classes-specifier`*`]`

If *arg* is omitted, `t` is used as the default.

---

map-instances-of-class

map-instances-on-space-instances *function unit-classes-specifier space-instances* `&key` *pattern filter-before filter-after use-marking verbose* [*Function*]

Purpose

Apply a function once to each unit instance on space instances, optionally selected by a retrieval pattern.

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

function	A function designator specifying a function object of one argument
unit-classes-specifier	An extended unit-classes specification (see below)
space-instances	A space instance, a list of space instances, a space-instance path regular expression, or `t` (indicating all space instances)
pattern	A retrieval pattern (see below; default is `t`)
filter-before	A single-argument predicate to be applied before pattern-matching tests occur
filter-after	A single-argument predicate to be applied after pattern-matching tests occur
use-marking	A generalized boolean (default is ***use-marking***)
verbose	A generalized boolean (default is ***find-verbose***)

Detailed syntax

unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*$^{+}$`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand `+` subclasses specifier is equivalent to `:plus-subclasses` and `=` to `:no-subclasses`.

pattern ::= *subpattern* | `t` | `:all`
subpattern ::= *pattern-element* |
`(not` *subpattern*`)` |
`(and` *subpattern**`)` |
`(or` *subpattern**`)`
pattern-element ::= `(`*pattern-op dimension-names pattern-values option**`)` |
`(`*boolean-dimension-unary-pattern-op dimension-names option**`)`
pattern-op ::= *ordered-dimension-pattern-op* |
enumerated-dimension-pattern-op |
boolean-dimension-binary-pattern-op
ordered-dimension-pattern-op ::= *ordered-dimension-any-numeric-value-pattern-op* |
ordered-dimension-explicit-type-pattern-op
ordered-dimension-explicit-type-pattern-op ::= *ordered-dimension-fixnum-pattern-op* |
ordered-dimension-short-float-pattern-op |
ordered-dimension-single-float-pattern-op |
ordered-dimension-double-float-pattern-op |
ordered-dimension-long-float-pattern-op |
ordered-dimension-pseudo-probability-pattern-op

ordered-dimension-any-numeric-value-pattern-op ::= `<` | `<=` | `>=` | `>` | `=` | `/=` |
`within` | `covers` | `overlaps` |
`abuts` | `starts` | `ends`
ordered-dimension-fixnum-pattern-op ::= `<&` | `<=&` | `>=&` | `>&` | `=&` | `/=&` |
`within&` | `covers&` | `overlaps&` |
`abuts&` | `starts&` | `ends&`
ordered-dimension-short-float-pattern-op ::= `<$&` | `<=$&` | `>=$&` | `>$&` | `=$&` | `/=$&` |
`within$&` | `covers$&` | `overlaps$&` |
`abuts$&` | `starts$&` | `ends$&`
ordered-dimension-single-float-pattern-op ::= `<$` | `<=$` | `>=$` | `>$` | `=$` | `/=$` |
`within$` | `covers$` | `overlaps$` |
`abuts$` | `starts$` | `ends$`
ordered-dimension-double-float-pattern-op ::= `<$$` | `<=$$` | `>=$$` | `>$$` | `=$$` | `/=$$` |
`within$$` | `covers$$` | `overlaps$$` |
`abuts$$` | `starts$$` | `ends$$`
ordered-dimension-long-float-pattern-op ::= `<$$$` | `<=$$$` | `>=$$$` | `>$$$` | `=$$$` | `/=$$$` |
`within$$$` | `covers$$$` | `overlaps$$$` |
`abuts$$$` | `starts$$$` | `ends$$$`
ordered-dimension-pseudo-probability-pattern-op ::= `<%` | `<=%` | `>=%` | `>%` | `=%` | `/=%` |
`within%` | `covers%` | `overlaps%` |
`abuts%` | `starts%` | `ends%`
enumerated-dimension-pattern-op ::= `is` | *enumerated-dimension-explicit-test-pattern-op*
enumerated-dimension-explicit-test-pattern-op ::= `is-eq` | `is-eql` | `is-equal` | `is-equalp`
boolean-dimension-binary-pattern-op ::= `eqv`
boolean-dimension-unary-pattern-op ::= `true` | `false`
dimension-names ::= *dimension-name* | `(`*dimension-name*$^{+}$`)`
pattern-values ::= *pattern-value* |
`(`*pattern-value*$^{+}$`)` |
`(`*pattern-value*$^{+}$ `.` *pattern-value*`)` |
`#(`*pattern-value*$^{+}$`)`
pattern-value ::= *point* | *interval* | *element* | *set*
interval ::= `(`*start end*`)` | `(`*start* `.` *end*`)` | `#(`*start end*`)`

Terms

point A number, `infinity`, or `-infinity`
start A number or `-infinity`
end A number or `infinity`
element An object

Description

The *function* will be applied only once to each unit instance, even if the unit instance resides on multiple space instances.

The *pattern* `t` matches all unit instances whose dimension values overlap the dimensional extent of at least one space instance in *space-instances*. The *pattern* `:all` matches every unit instance on a space instance in *space-instances*, regardless of dimensional overlap.

Declared numeric (see page 143) and pseudo probability (see page 149) pattern operators are also supported, for example: `=&`, `=$&`, `=$`, `=$$`, `=$$$`, and `=%` and `within&`, `within$&`, `within$`, `within$$`, `within$$$`, and `within%`.

See also

find-verbose (page 467)
use-marking (page 468)
warn-about-unusual-requests (page 469)
do-instances-of-class (page 470)
do-instances-on-space-instances (page 472)
find-instances (page 483)
find-instances-of-class (page 488)
make-instances-of-class-vector (page 490)
map-instances-of-class (page 491)
map-sorted-instances-of-class (page 496)
with-find-stats (page 498)
Declared numerics (page 143)

Examples

Remove all the `hyp` unit instances that reside on the `(bb probable-hyps)` space instance, deleting those unit instances that do not reside on any other space instance:

```
(let ((space-instance
          (find-space-instance-by-path '(bb probable-hyps))))
  (map-instances-on-space-instances
     #'(lambda (instance)
          (if (>= (length (space-instances-of instance) 1))
                  (remove-instance-from-space-instance instance
space-instance)
                  (delete-instance instance)))
     'hyp
     space-instance))
```

Delete `hyp` unit instances that reside on the `(bb probable-hyps)` space instance that have a belief value less than 0.5:

```
(map-instances-on-space-instances
   #'delete-instance
   'hyp '(bb probable-hyps) :pattern '(< belief .5))
```

Note

Fixnum `overlaps` comparisons can result in bignum computations if the combined intervals of the pattern and a candidate unit instance exceeds `most-positive-fixnum`.

map-sorted-instances-of-class *function unit-classes-specifier predicate* &key *key* [*Function*]

Purpose

Apply a function once to each unit instance of the specified unit classes, in sorted order.

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

function	A function designator specifying a function object of one argument
unit-classes-specifier	An extended unit-classes specification (see below)
predicate	A function designator specifying a function object of two arguments that returns a generalized boolean
key	A function designator specifying a function object of one argument, or `nil` (default is `nil`)

Detailed syntax

unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*$^{+}$`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description

The specified function is applied once to each unit instance of the specified unit classes, whether or not the instances reside on any space instances.

See also

do-sorted-instances-of-class (page 475)
make-instances-of-class-vector (page 490)
map-instances-of-class (page 491)

Example

Print a list of all `hyp` instance names, in ascending order:

```
(map-sorted-instances-of-class
   #'(lambda (instance)
       (print (instance-name-of instance)))
   'hyp #'< :key #'instance-name-of)
```

report-find-stats `&key` *reset* [*Function*]

Purpose

Display the retrieval statistics collected for **find-instances** and **map-instances-on-space-instances**.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments

reset A generalized boolean (default is `nil`)

Description

Report-find-stats displays the retrieval statistics within the scope of an active **with-find-stats**.

If *reset* is non-`nil`, the statistics are cleared after the report is displayed.

See also

with-find-stats (page 498)

Examples

```
> (with-find-stats ()
    (scanner (find-instance-by-name 471 'hyp))
    (report-find-stats)
    (scanner (find-instance-by-name 632 'hyp)))
;; Find/Map Statistics:
;;         20 find/map operations (0 using marking, 20 using hashing)
;;        100 buckets scanned
;;       9240 instances touched
;;       9240 instances considered
;;        521 instances accepted
;;       0.16 seconds (0.80 msec/operation)
;; Find/Map Statistics:
;;         40 find/map operations (0 using marking, 40 using hashing)
;;        200 buckets scanned
;;      18480 instances touched
;;      18480 instances considered
;;       1042 instances accepted
;;       0.32 seconds (0.80 msec/operation)
(#<hyp 319 (1835 8419) 0.91 [4..12]>
 #<hyp 331 (1835 8419) 0.88 [15..30]>)
> (report-find-stats)
;; No find/map statistics are available.
>
```

with-find-stats `(&key` *initialize report*`)` *declaration** *form** $\Rightarrow$ *result** [*Macro*]

Purpose

Record and optionally display retrieval statistics for **find-instances** and **map-instances-on-space-instances**.

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

initialize A generalized boolean(default is `t`)
report A generalized boolean(default is `t`)
declaration A declare expression (not evaluated)
forms An implicit **progn** of forms to be evaluated
results The values returned by evaluating the last *form*

Returns

The values returned by evaluating the last *form*.

See also

find-instances (page 483)
map-instances-on-space-instances (page 493)
report-find-stats (page 497)
without-find-stats (page 499)

Example

Collect and display the retrieval statistics associated with running an application function `scanner`:

```
> (with-find-stats ()
    (scanner (find-instance-by-name 471 'hyp)))
;; Find/Map Statistics:
;;        20 find/map operations (0 using marking, 20 using hashing)
;;       100 buckets scanned
;;      9240 instances touched
;;      9240 instances considered
;;       521 instances accepted
;;      0.16 seconds (0.80 msec/operation)
(#<hyp 419 (1835 4791) 0.85 [5..35]>
 #<hyp 233 (1835 4791) 0.89 [5..35]>)
>
```

without-find-stats *declaration** *form** ⇒ *result** [*Macro*]

Purpose
Disable the collecting of retrieval statistics while executing *forms*.

Package :gbbopen

Module :gbbopen-core

Arguments
declaration A declare expression (not evaluated)
forms An implicit **progn** of forms to be evaluated
results The values returned by evaluating the last *form*

Returns
The values returned by evaluating the last *form*.

See also
with-find-stats (page 498)

Example
Collect and display the retrieval statistics associated with running an application function `scanner`:

```
> (with-find-stats ()
     (scanner (find-instance-by-name 471 'hyp))
     (without-find-stats
        (scanner (find-instance-by-name 632 'hyp))))
  ;; Find/Map Statistics:
  ;;        20 find/map operations (0 using marking, 20 using hashing)
  ;;       100 buckets scanned
  ;;      9240 instances touched
  ;;      9240 instances considered
  ;;       521 instances accepted
  ;;      0.16 seconds (0.80 msec/operation)
  (#<hyp 319 (1835 8419) 0.91 [4..12]>
   #<hyp 331 (1835 8419) 0.88 [15..30]>)
  >
```

5.6 Saving and Sending

This section contains `:gbbopen-core` entities that pertain to saving/sending and loading/reading objects.

block-saved/sent-time [*Variable*]

Purpose

Dynamically bound in **with-reading-saved/sent-objects-block** to the Universal Time when the block was saved/sent.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Initial value

Unbound

See also

block-saved/sent-value (page 502)
with-reading-saved/sent-objects-block (page 516)
with-saving/sending-block (page 518)

block-saved/sent-value [*Variable*]

Purpose

Dynamically bound in **with-reading-saved/sent-objects-block** to the saved/sent value associated with the block.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Initial value

Unbound

See also

block-saved/sent-time (page 501)
with-reading-saved/sent-objects-block (page 516)
with-saving/sending-block (page 518)

print-object-for-sending [*Variable*]

Purpose

Controls **print-object-for-saving/sending** printing customization for sending versus for saving.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Value type

A generalized boolean

Initial value

`nil`

See also

omitted-slots-for-saving/sending (page 510)
print-object-for-saving/sending (page 511)
print-slot-for-saving/sending (page 513)

Example

Send some `hyp` unit instances to another agent:

```
> (let ((*print-object-for-sending* 't))
    (with-saving/sending-block (stream :package ':my-app)
      (let ((*save/send-references-only* nil))
        (do-instances-of-class (instance 'hyp :plus-subclasses)
          (print-object-for-saving/sending instance stream)))))
nil
>
```

save/send-references-only [*Variable*]

Purpose

Controls whether **print-object-for-saving/sending** prints references to instances or the instance contents.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Value type

A generalized boolean

Initial value

True

See also

omitted-slots-for-saving/sending (page 510)
print-object-for-saving/sending (page 511)
print-slot-for-saving/sending (page 513)

Example

Send some `hyp` unit instances to another agent:

```
> (let ((*print-object-for-sending* 't))
    (with-saving/sending-block (stream :package ':my-app)
      (let ((*save/send-references-only* nil))
        (do-instances-of-class (instance 'hyp :plus-subclasses)
          (print-object-for-saving/sending instance stream)))))
nil
>
```

initialize-saved/sent-instance *instance slots slot-values missing-slot-names* ⇒ *instance* [*Generic Function*]

Purpose
Initializes the slots of *instance* when reading saved (or sent) values.

Method signatures
initialize-saved/sent-instance (*instance* `standard-object`) *slots slot-values missing-slot-names* ⇒ *instance*
initialize-saved/sent-instance (*instance* `standard-unit-instance`) *slots slot-values missing-slot-names* ⇒ *instance*
initialize-saved/sent-instance (*instance* `standard-space-instance`) *slots slot-values missing-slot-names* ⇒ *instance*

Package
`:gbbopen-tools`

Module
`:gbbopen-tools` (the unit-instance and space-instance methods are added by `:gbbopen-core`)

Arguments
instance	A `standard-object` instance
slots	A list of `effective-slot-definition` objects
slot-values	A list of values (corresponding to the slots in *slots*)
missing-slot-names	A list of symbols naming slots and slot values that were saved/sent, but which are no longer present in the class definition of *instance*.

Returns
The *instance*.

Description
Initialize-saved/sent-instance is called when loading a blackboard repository or journal or when reading a network-streamer connection; it should not be called directly.

See also
load-blackboard-repository (page 507)
with-reading-saved/sent-objects-block (page 516)

Examples
Restore the binary value of slot `song` in unit instances of `bird` that was saved/sent as a Base64 portable string (see the companion encoding example on page 513):

```
(defmethod initialize-saved/sent-instance :after ((bird bird)
                                                  slots slot-values
                                                  missing-slot-names)
  (declare (ignore slots slot-values missing-slot-names))
  (setf (song-of bird) (base64-decode (song-of bird))))
```

Transfer the slot value that was saved/sent as the slot named `old-slot` to the slot named `new-slot` in the current class definition of `hyp`:

```
(defmethod initialize-saved/sent-instance :after ((hyp hyp)
                                                  slots slot-values
                                                  missing-slot-names)
  (declare (ignore missing-slot-names))
  (let ((position (position 'old-slot slots
                            :key #'slot-definition-name
                            :test #'eq)))
     (setf (new-slot-of hyp) (nth position slot-values))))
```

initialize-saved/sent-instance

load-blackboard-repository *pathname* `&key` *class-name-translations coalesce-strings confirm-if-not-empty disable-events estimated-peak-forward-references external-format readtable read-eval retain-classes retain-event-functions retain-event-printing* ⇒ *pathname, saved-time, saved-value* [*Function*]

Purpose

Load the blackboard repository (all unit instances and space instances) that has been saved to a file previously by **save-blackboard-repository**.

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

pathname	A pathname designator
class-name-translations	An association list (default is `nil`)
coalesce-strings	A generalized boolean (default is `nil`)
confirm-if-not-empty	A generalized boolean (default is true)
disable-events	A generalized boolean (default is `t`)
estimated-peak-forward-references	An integer (default is `*default-estimated-peak-forward-references*`)
external-format	An external-file-format designator (default is `:default`)
readtable	A readtable (default is `*reading-saved/sent-objects-readtable*`)
read-eval	A generalized boolean (default is `nil`)
retain-classes	An extended unit-classes specification (see below)
retain-event-functions	A generalized boolean (default is `nil`)
retain-event-printing	A generalized boolean (default is `nil`)
pathname	A pathname
saved-time	A Universal Time
saved-value	An object

Returns

Three values: the pathname of the saved blackboard-repository data file, the time when the blackboard-repository data file was saved, and the save value specified when the repository was saved. If replacing a non-empty blackboard repository is not confirmed, `nil` is returned.

Events

If *disable-events* is `nil`, the following events may be signaled (in order) as unit instances and space instances are deleted prior to loading:

- `delete-instance-event`
- `unlink-event`
- `instance-removed-from-space-instance-event`
- `instance-deleted-event`

Description

If *pathname* does not specify a file type, the type `bb` is added to it. Then, `(user-homedir-pathname)` is used to supply any missing components to *pathname*.

The *class-name-translations* association list, if specified, should contain conses of the form:

```
(class-name . new-class-name)
```

for any class translations that should occur during repository loading.

If *coalesce-strings* is true, loaded strings that are `equal` become shared (`eq`). This coalescing is performed using a temporary hash table whose initial size can be specified by providing an integer value for *coalesce-strings*. If a hash table is provided as the value for *coalesce-strings*, it is used in place of the temporary hash table.

If *confirm-if-not-empty* is true, the user must confirm loading when the current blackboard repository contains unit instances.

Unit instances that are referenced before they are defined are recorded using a temporary hash table whose initial size can be specified by providing an integer value for *estimated-peak-forward-references*.

See also

confirm-if-blackboard-repository-not-empty-p (page 436)
save-blackboard-repository (page 514)
with-reading-saved/sent-objects-block (page 516)

Examples

Load the GBBopen Tutorial Example application (without running it) and then load a blackboard repository that was saved previously:

```
> :tutorial-example :noautorun
   ...
> (load-blackboard-repository "tutorial")
;; 35 temporarily forward-referenced instances (peak count)
#P"<homedir>/tutorial.bb"
3429178245
#<path 1>
>
```

Try loading the repository again:

```
> (load-blackboard-repository "tutorial")
The blackboard repository is not empty.
Continue anyway (the current contents will be deleted) [y or n]? n
nil
>
```

Define a new unit class, `new-location`, and then load the repository again, translating all `location` unit instances to `new-location` unit instances:

```
> (define-unit-class new-location (location) ())
#<standard-unit-class new-location>
> (load-blackboard-repository "tutorial"
    :class-name-translations '((location . new-location))
    :confirm-if-not-empty nil)
;; 35 temporarily forward-referenced instances (peak count)
```

```
#P"<homedir>/tutorial.bb"
3429178245
#<path 1>
> :dsbb

Space Instance                Contents
--------------                --------
known-world                   54 instances (53 new-location; 1 path)

Unit Class                    Instances
----------                    ---------
control-shell                         1 *
ks                                    4 +
ksa-queue                             2 +
new-location                         53
ordered-ksa-queue                     1 +
path                                  1
standard-space-instance               1
                              ---------
                                     63 instances
>
```

load-blackboard-repository

omitted-slots-for-saving/sending *instance* ⇒ *omitted-slot-names* [*Generic Function*]

Purpose
Add slot names of a class to the list of slots that are not saved or sent.

Method signatures
omitted-slots-for-saving/sending (*instance* `standard-space-instance`) ⇒ *omitted-slot-names*
omitted-slots-for-saving/sending (*instance* `standard-unit-instance`) ⇒ *omitted-slot-names*
omitted-slots-for-saving/sending (*instance* `t`) ⇒ `nil`

Package
`:gbbopen-tools`

Module
`:gbbopen-tools` (the unit-instance and space-instance methods are added by `:gbbopen-core`)

Arguments
instance	An instance
omitted-slot-names	A proper list

Returns
A list of omitted slot names for the class of *instance*.

See also
print-object-for-sending (page 503)
load-blackboard-repository (page 507)
print-object-for-saving/sending (page 511)
print-slot-for-saving/sending (page 513)
save-blackboard-repository (page 514)

Example
Specify that slot `complex-unsaved-slot` in `my-unit-instance` should be added to the list of slots that are not saved or sent:

```
(defmethod omitted-slots-for-saving/sending ((instance my-unit-instance))
  (cons 'complex-unsaved-slot (call-next-method)))
```

print-object-for-saving/sending *object stream* ⇒ *object* [*Generic Function*]

Purpose
Write the printed representation of *object* to *stream* when saving or sending.

Method signatures
print-object-for-saving/sending (*object* `array`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `bit-vector`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `cons`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `function`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `generic-function`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `hash-table`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `package`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `standard-class`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `standard-generic-function`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `standard-object`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `standard-unit-instance`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `string`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `structure-object`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `vector`) *stream* ⇒ *object*
print-object-for-saving/sending (*object* `t`) *stream* ⇒ *object*

Package
`:gbbopen-tools`

Module
`:gbbopen-tools` (the unit-instance method is added by `:gbbopen-core`)

Arguments
object An object
stream A stream

Returns
The *object*.

Errors
A call to **print-object-for-saving/sending** was made outside the dynamic scope of a **with-saving/sending-block**.

See also

print-object-for-sending (page 503)
omitted-slots-for-saving/sending (page 510)
print-slot-for-saving/sending (page 513)
save-blackboard-repository (page 514)
with-saving/sending-block (page 518)

Example

Define a method to save/send compiled-function references, when possible:

```
(defmethod print-object-for-saving/sending ((fn function)
                                            stream)
  (flet ((save/send-error ()
           (let ((*print-readably* nil))
             (error "Unable to save/send ~s reliably." fn))))
    (typecase fn
      (compiled-function
       (multiple-value-bind (lambda-expression closure-p name)
           (function-lambda-expression fn)
         (declare (ignore lambda-expression)
                  #+(or cmu sbcl)
                  (ignore closure-p))
         (when (or
                ;; CMUCL and SBCL always return that
                ;; closure-p is true, so we ignore their
                ;; closure-p value and risk that 'fn' might
                ;; be a non-trivial closure that can't be
                ;; represented externally:
                #-(or cmu sbcl)
                closure-p
                ;; Implementations are free to always return
                ;; nil as name or return an object that is
                ;; not valid for use as a name in function.
                ;; Fortunately, implementations usually
                ;; provide a useful name value:
                (not (symbolp name))
                (not name))
           (save/send-error))
         (format stream "#'~s" name)))
      (otherwise (save/send-error))))
  fn)
```

Note that the above method is not without risk, but it will work in many situations.

---

print-object-for-saving/sending

print-slot-for-saving/sending *instance slot-name stream* [*Generic Function*]

Purpose

Write the printed representation of slot *slot-name* in *object* to *stream* when saving or sending.

Method signatures

print-slot-for-saving/sending `(`*instance* `standard-object)` *slot-name stream*

print-slot-for-saving/sending `(`*instance* `standard-unit-instance) (`*slot-name* `(eql gbbopen::%%space-instances%%)` *stream*

Package

`:gbbopen-tools`

Module

`:gbbopen-tools` (the unit-instance method is added by `:gbbopen-core`)

Arguments

instance A `standard-object` instance

slot-name A non-`nil`, non-keyword symbol

stream A stream

Errors

A call to **print-slot-for-saving/sending** was made outside the dynamic scope of a **with-saving/sending-block**.

See also

print-object-for-sending (page 503)
omitted-slots-for-saving/sending (page 510)
print-object-for-saving/sending (page 511)
print-slot-for-saving/sending (page 513)
save-blackboard-repository (page 514)

Example

Write the binary value of slot `song` in unit instances of `bird` as a Base64 portable string (see the companion decoding example on page 505):

```
  (defmethod print-slot-for-saving/sending ((bird bird) (slot-name (eql
'song)) stream)
    (print-object-for-saving/sending (base64-encode (song-of bird)) stream))
```

save-blackboard-repository *pathname* `&key` *external-format package read-default-float-format value* [*Function*]
⇒ *saved-repository-pathname*

Purpose
Save the blackboard repository (all unit instances and space instances) to a file.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
pathname	A pathname designator
external-format	An external-file-format designator (default is `:default`)
package	A package designator indicating the package to be used when saving and loading the blackboard repository (default is `:common-lisp`)
read-default-float-format	One of the atomic type specifiers `short-float`, `single-float`, `double-float`, or `long-float` to be used when saving and loading the blackboard repository (default is `single-float`)
value	An object to be saved with the repository and returned when the repository is loaded (default is `nil`)
saved-repository-pathname	A pathname

Returns
The pathname of the saved blackboard-repository data file.

Description
If *pathname* does not specify a file type, the type `bb` is added to it. Then, `(user-homedir-pathname)` is used to supply any missing components to *pathname*.

The size of the blackboard-repository data file can be reduced by specifying a *package* containing the majority of the symbols that are written to the file.

A **with-saving/sending-block**, with the *package* and *read-default-float-format* values, is established when saving the blackboard repository.

See also

load-blackboard-repository (page 507)
omitted-slots-for-saving/sending (page 510)
print-object-for-saving/sending (page 511)
print-slot-for-saving/sending (page 513)
with-saving/sending-block (page 518)

Example
Run the GBBopen Tutorial Example application and then save the resulting blackboard repository:

```
> :tutorial-example
    ...
;; No executable KSAs remain, exiting control shell
```

```
;; Control shell 1 exited: 63 cycles completed
;; Run time: 0.04 seconds
;; Elapsed time: 0 seconds
:quiescence
> (save-blackboard-repository "tutorial" :package ':tutorial)
#P"<homedir>/tutorial.bb"
>
```

Save the repository again, this time with a value to be returned when the repository is loaded:

```
> (save-blackboard-repository "tutorial"
    :package ':tutorial
    :value (find-instance-by-name 1 'path))
#P"<homedir>/tutorial.bb"
>
```

---

save-blackboard-repository

with-reading-saved/sent-objects-block `(`*stream* `&key` *class-name-translations coalesce-strings estimated-peak-forward-references readtable read-eval*`)` *form** $\Rightarrow$ *result** [*Macro*]

Purpose

Bind reader and GBBopen reading-saved/sent-objects control variables) to values that produce standard GBBopen reading-saved/sent-objects behavior when evaluating *forms*.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

stream	A stream
class-name-translations	An association list (default is `nil`)
coalesce-strings	A generalized boolean (default is `nil`)
estimated-peak-forward-references	An integer (default is `*default-estimated-peak-forward-references*`)
readtable	A readtable (default is `*reading-saved/sent-objects-readtable*`)
read-eval	A generalized boolean (default is `nil`)
forms	An implicit **progn** of forms to be evaluated
results	The values returned by evaluating the last *form*

Returns

The values returned by evaluating the last *form*.

Description

Within the dynamic extent of the body of *forms*, all Common Lisp reader control variables (including any implementation-defined ones) and GBBopen reading-saved/sent-objects control variables are bound to values that produce standard GBBopen reading-saved/sent-objects behavior. The function **load-blackboard-repository** establishes a **with-reading-saved/sent-objects-block** when loading the blackboard repository, and this macro can be used in constructing specialized reading-saved/sent-objects operations.

The *class-name-translations* association list, if specified, should contain conses of the form:

```
(class-name . new-class-name)
```

for any class translations that should occur during reading.

If *coalesce-strings* is true, strings read within the dynamic extent of the body of *forms* that are `equal` become shared (`eq`). This coalescing is performed using a temporary hash-table whose initial size can be specified by providing an integer value for *coalesce-strings*. If a hash table is provided as the value for *coalesce-strings*, it is used in place of the temporary hash table.

Unit instances that are referenced before they are defined are recorded using a temporary hash table whose initial size can be specified by providing an integer value for *estimated-peak-forward-references*.

See also

block-saved/sent-time (page 501)
block-saved/sent-value (page 502)
load-blackboard-repository (page 507)
with-saving/sending-block (page 518)

---

with-reading-saved/sent-objects-block

with-saving/sending-block (*stream* `&key` *package read-default-float-format value*) [*Macro*]
*form** ⇒ *result**

Purpose
Bind printer and GBBopen save/send control variables) to values that produce standard GBBopen save/send-object behavior when evaluating *forms*.

Package
`:gbbopen-tools`

Module
`:gbbopen-tools`

Arguments
stream	A stream
package	A package designator indicating the package to be used when saving/sending and when loading/receiving (default is `:common-lisp`)
read-default-float-format	One of the atomic type specifiers `short-float`, `single-float`, `double-float`, or `long-float` (default is `single-float`)
value	An object to be saved/sent and bound to ***block-saved/sent-value*** when loading/receiving with **with-reading-saved/sent-objects-block** (default is `nil`)
forms	An implicit **progn** of forms to be evaluated
results	The values returned by evaluating the last *form*

Returns
The values returned by evaluating the last *form*.

Description
Within the dynamic extent of the body of *forms*, all Common Lisp printer control variables (including any implementation-defined ones) and GBBopen save/send control variables are bound to values that produce standard GBBopen save/send-object behavior. The function **save-blackboard-repository** establishes a **with-saving/sending-block** when saving the blackboard repository, and this macro can be used in constructing specialized saving/sending operations.

See also

save-blackboard-repository (page 514)
print-object-for-saving/sending (page 511)
with-reading-saved/sent-objects-block (page 516)

5.7 Queue Management

The :queue module provides queue-management objects and operators.

clear-queue *queue* [*Generic Function*]

Purpose
Quickly remove all elements from queue.

Method signatures
clear-queue (*queue* queue)

Package :gbbopen

Module :queue

Arguments
queue A GBBopen queue

See also
insert-on-queue (page 523)
make-queue (page 525)
remove-from-queue (page 535)

Example
Remove all pending KSAs that have been triggered but not yet executed:

```
(clear-queue pending-ksas)
```

do-queue (*var queue*) *declaration** {*tag* | *form*}* [*Macro*]

Purpose
Iterate over each queue element on the specified *queue*.

Package :gbbopen

Module :queue

Arguments
var A variable symbol
queue A GBBopen queue
declaration A declare expression (not evaluated)
tag A `go` tag (not evaluated)
form A form

Description
The iteration over queue elements is performed in queue order (first to last).

See also
map-queue (page 526)
queue (page 532)
ordered-queue (page 530)

Example
Count the number of pending KSAs that were triggered by `hyp`:

```
> (let ((count 0))
    (do-queue (ksa pending-ksas)
      (when (memq hyp (collect-trigger-instances ksa))
        (incf& count)))
    count)
31
>
```

first-queue-element *queue* ⇒ *queue-element* [*Generic Function*]

Purpose
Return the first queue element on *queue*.

Method signatures
first-queue-element (*queue* queue) ⇒ *queue-element*

Package :gbbopen

Module :queue

Arguments
queue A GBBopen queue
queue-element A GBBopen queue element object

Returns
The first queue element on *queue*.

See also
last-queue-element (page 524)
nth-queue-element (page 528)

Example
```
> (first-queue-element pending-ksas)
#<ksa 2217>
>
```

insert-on-queue *queue-element queue* ⇒ *queue-element* [*Generic Function*]

Purpose
Insert a queue element on *queue*.

Method signatures
insert-on-queue (*queue-element* queue-element) (*queue* `queue`) ⇒ *queue-element*

insert-on-queue (*queue-element* `queue-element`) (*queue* `ordered-queue`) ⇒ *queue-element*

Package `:gbbopen`

Module `:queue`

Arguments
queue-element A GBBopen queue element object

queue A GBBopen queue

Returns
The supplied *queue-element*.

Description
If *queue* is an ordered queue, the position of *queue-element* in *queue* is based on the key and test functions provided when the queue was created. If *queue* is a standard queue, *queue-element* is inserted at the end of the queue.

See also
clear-queue (page 520)

make-queue (page 525)

remove-from-queue (page 535)

Example
```
> (insert-on-queue ksa pending-ksas)
#<ksa 2372>
>
```

last-queue-element *queue* ⇒ *queue-element* [*Generic Function*]

Purpose
Return the last queue element on *queue*.

Method signatures
last-queue-element `(`*queue* `queue)` ⇒ *queue-element*

Package `:gbbopen`

Module `:queue`

Arguments
queue A GBBopen queue
queue-element A GBBopen queue element object

Returns
The last queue element on *queue*.

See also
first-queue-element (page 522)
nth-queue-element (page 528)

Example
```
> (last-queue-element pending-ksas)
#<ksa 2372>
>
```

make-queue `&rest` *initargs* ⇒ *queue* [*Function*]

Purpose
Make a GBBopen queue.

Package `:gbbopen`

Module `:queue`

Arguments
initargs An initialization argument list
queue A GBBopen queue

Returns
The newly created queue.

Errors
Use of an initialization argument that has not been declared as valid.

The supplied or generated instance name is identical to the instance name of an existing unit instance of *class*.

See also
queue (page 532)
ksa-queue (page 610)
ordered-queue (page 530)
ordered-ksa-queue (page 613)

Example
```
> (setf pending-ksas (make-queue :class 'ordered-ksa-queue
                                 :key #'rating-of))
#<ordered-ksa-queue>
>
```

map-queue *function queue* [*Generic Function*]

Purpose
Apply a function to each queue element on the specified *queue*.

Method signatures
map-queue `(`***function*** `t) (`***queue*** `queue)`

Package `:gbbopen`

Module `:queue`

Arguments
function A function designator specifying a function object of one argument
queue A GBBopen queue

Description
The *function* is applied to the queue elements in queue order (first to last).

See also
do-queue (page 521)
queue (page 532)
ordered-queue (page 530)

Example
Count the number of pending KSAs that were triggered by `hyp`:

```
> (let ((count 0))
    (map-queue #'(lambda (ksa)
                   (when (memq hyp (collect-trigger-instances ksa))
                     (incf& count)))
               pending-ksas)
    count)
31
>
```

next-queue-element *queue-element* ⇒ *next-queue-element* [*Generic Function*]

Purpose
Return the queue element that follows *queue-element* on a GBBopen queue.

Method signatures
next-queue-element (*queue-element* `queue-element`) ⇒ *next-queue-element*

Package `:gbbopen`

Module `:queue`

Arguments
queue-element A GBBopen queue element object
next-queue-element A GBBopen queue element object

Returns
The queue element that follows *queue-element*.

See also
previous-queue-element (page 531)

Example
```
> (next-queue-element ksa)
#<ksa 2166>
>
```

nth-queue-element *n queue* ⇒ *queue-element* [*Generic Function*]

Purpose
Return the nth queue element on *queue*.

Method signatures
nth-queue-element `(`*n* `fixnum) (`*queue* `queue)` ⇒ *queue-element* or `nil`

Package `:gbbopen`

Module `:queue`

Arguments
n	A `fixnum`
queue	A GBBopen queue
queue-element	A GBBopen queue element objec

Returns
The specified queue element or `nil` if none exists.

Description
Returns the nth element in *queue* (zero origin) or `nil` if the queue is shorter than *n*. If *n* is negative, return the nth element counting backward from the end of the queue (one origin).

See also
first-queue-element (page 522)
last-queue-element (page 524)

Examples
Return the first element on `pending-ksas` (equivalent to **first-queue-element**):

```
> (nth-queue-element 0 pending-ksas)
#<ksa 2217>
>
```

Return the last element on `pending-ksas` (equivalent to **last-queue-element**):

```
> (nth-queue-element -1 pending-ksas)
#<ksa 2372>
>
```

on-queue-p *queue-element* ⇒ *queue* or `nil` [*Generic Function*]

Purpose
Determine if *queue-element* resides on a queue by returning the queue or `nil`.

Method signatures
on-queue-p (*queue-element* `queue-element`) ⇒ *queue*

Package `:gbbopen`

Module `:queue`

Arguments
queue-element A GBBopen queue element object
queue A GBBopen queue

Returns
The queue queue on which *queue-element* resides or `nil` if *queue-element* is not on a queue.

See also
queue-element (page 533)
show-queue (page 536)

Example
Return the queue on which `ksa` resides:

```
> (on-queue-p ksa)
#<ordered-queue>
>
```

ordered-queue [*Unit Class*]

Package

`:gbbopen`

Module

`:queue`

Description

The unit class whose instances are used as the header of ordered (sorted) GBBopen queues.

See also

clear-queue (page 520)
executed-ksas-of (page 602)
make-queue (page 525)
obviated-ksas-of (page 611)
on-queue-p (page 529)
ordered-ksa-queue (page 613)
queue (page 532)
queue-element (page 533)
show-queue (page 536)

previous-queue-element *queue-element* ⇒ *previous-queue-element* [*Generic Function*]

Purpose
Return the queue element that precedes *queue-element* on a GBBopen queue.

Method signatures
previous-queue-element (*queue-element* `queue-element`) ⇒ *previous-queue-element*

Package `:gbbopen`

Module `:queue`

Arguments
queue-element	A GBBopen queue element object
previuos-queue-element	A GBBopen queue element object

Returns
The queue element that precedes *queue-element*.

See also
next-queue-element (page 527)

Example
```
> (previous-queue-element ksa)
#<ksa 2166>
>
```

queue [*Unit Class*]

Package

`:gbbopen`

Module

`:queue`

Description

The unit class whose instances are used as the header of GBBopen queues.

See also

clear-queue (page 520)
ksa-queue (page 610)
make-queue (page 525)
ordered-queue (page 530)
queue-element (page 533)
show-queue (page 536)

queue-element [*Unit Class*]

Package

`:gbbopen`

Module

`:queue`

Description

Objects that inherit from the unit class **queue-element** can be elements of GBBopen queues.

See also

on-queue-p (page 529)
ordered-queue (page 530)
queue (page 532)

Example

Define a KS activation class whose instances can be kept in a queue of pending KSAs:

```
(define-unit-class ksa (standard-unit-instance queue-element)
  ((rating
    :initform -1
    :type rating)
   ... ))
```

queue-length *queue* `&optional` *recount-p* ⇒ *integer* [*Generic Function*]

Purpose
Return the length of *queue*.

Method signatures
queue-length `(`*queue* `queue) &optional` *recount-p* ⇒ *integer*

Package `:gbbopen`

Module `:queue`

Arguments
queue A GBBopen queue
recount-p If true, actually counts the individual queue elements (default is `nil`)
integer An integer

Returns
The *queue* length.

Description
Normally **queue-length** simply returns a count that is maintained with the queue. Although highly unlikely, this count could become inaccurate if queue-element insertion or deletion operations are aborted in process. If *recount-p* is true, the elements are actually counted and then the count maintained with the queue is updated and returned.

Examples
Return the number of KSAs in the queue `pending-ksas`:

```
> (queue-length pending-ksas)
896
>
```

Count and then return the actual number of KSAs in the queue `pending-ksas`:

```
> (queue-length pending-ksas 't)
896
>
```

remove-from-queue *queue-element* ⇒ *queue-element* [*Generic Function*]

Purpose
Remove a queue element from its queue.

Method signatures
remove-from-queue (*queue-element* `queue-element`) ⇒ *queue-element*

Package `:gbbopen`

Module `:queue`

Arguments
queue-element A GBBopen queue element object

Returns
The supplied *queue-element*.

See also
clear-queue (page 520)
insert-on-queue (page 523)

Example
```
> (remove-from-queue ksa)
#<ksa 2372>
>
```

show-queue *queue* `&key` *start end show-element-function* [*Generic Function*]

Purpose
Print the elements on *queue*.

Method signatures
show-queue `(`*queue* `queue) &key` *start end show-element-function*

Package `:gbbopen`

Module `:queue`

Arguments
queue	A GBBopen queue
start	An integer specifying the first queue element to be shown (default is `0`)
end	An integer specifying the last queue element to be shown or `nil` indicating that the last queue element is to be shown (default is `nil`)
show-element-function	A function designator specifying a function object of two arguments used to print each queue element line (default is `#'standard-show-queue-element`)

See also
clear-queue (page 520)
executed-ksas-of (page 602)
make-queue (page 525)
obviated-ksas-of (page 611)
ordered-ksa-queue (page 613)
ordered-queue (page 530)
queue (page 532)
queue-element (page 533)
on-queue-p (page 529)
pending-ksas-of (page 614)

Example
Show the control shell's pending KSAs queue (early in a `:tutorial-example` run):

```
>  (show-queue (pending-ksas-of (current-control-shell)))
    0. #<ksa 7 random-walk-ks 100>
    1. #<ksa 1 count-center-locations-ks 90>
    2. #<ksa 4 print-walk-ks 80>(show-queue pending-ksas :end 5)
>
```

6 GBBopen Extensions

GBBopen extension modules provide support for journaling and for network streaming between GBBopen streamer nodes. Documentation for these entities is arranged into the following sections:

- streaming entities (Section 6.1)
- journaling entities (Section 6.2)
- networking-streaming entities (Section 6.3)

6.1 Streaming

The `:streaming` module provides common entities used with journaling (Section 6.2) and network streaming (Section 6.3).

The entities in this module are experimental and are subject to change.

add-mirroring *streamer* [*unit-class-or-instance-specifier*]] `&key` *slot-names paths* [*Function*]

Purpose
Add mirroring of one or more unit classes.

Package `:gbbopen`

Module `:streaming`

Arguments
streamer	A streamer
unit-class-or-instance-specifier	An extended unit-class or instance specification (see below; default is `t`)
slot-names or slot-name	A slot-name or list of slot-names (default is `t`)
paths or path	A space-instance path regular expression (default is `(*)`)

Detailed syntax
unit-class-or-instance-specifier ::= *unit-instance* | `(`*unit-instance*`*)` | *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
The *paths* argument is either the symbol `t` (indicating all space instances) or a list representing a regular expression where the following reserved symbols are interpreted as follows:
- `=` matches one occurrence in a space-instance path
- `?` matches zero or one occurrence in a space-instance path
- `+` matches one or more occurrences in a space-instance path
- `*` matches zero or more occurrences in a space-instance path
- `^` move to parent

See also
remove-mirroring (page 560)

Example
Add full mirroring of `hyp` unit instances to *streamer*:

```
(add-mirroring streamer 'hyp)
```

Note
Unit-instance-specific mirroring is not yet implemented in GBBopen.

add-to-broadcast-streamer *streamer broadcast-streamer* [*Function*]

Purpose

Add a journal or network streamer to a broadcast streamer.

Package `:gbbopen`

Module `:streaming`

Arguments

streamer	A journal or network streamer.
broadcast-streamer	A broadcast streamer

Errors

See also

make-broadcast-streamer (page 543)
remove-from-broadcast-streamer (page 546)

Example

Add a network streamer to streamer node `"you-too"` to broadcast streamer `broadcast-streamer`:

```
> (add-to-broadcast-streamer
    (open-network-streamer "me" "you-too")
    broadcast-streamer)
#<broadcast-streamer 2 constituents>
>
```

clear-streamer-queue *streamer* [*Function*]

Purpose

Clear the contents of a streamer queue without writing its contents, emptying the queue for further queueing.

Package

`:gbbopen`

Module

`:streaming`

Arguments

streamer A streamer

See also

with-queued-streaming (page 563)
write-streamer-queue (page 565)

Example

Clear streamer `*streamer*`:

```
(clear-streamer-queue *streamer* :tag (get-universal-time))
```

close-streamer *streamer* [*Generic Function*]

Purpose

Close a journal or network streamer.

Method signatures

close-streamer (*streamer* `journal-streamer`)

close-streamer (*streamer* `network-streamer`)

Package `:gbbopen`

Module `:streaming`

Arguments

streamer A journal or network streamer

Errors

Threads (multiprocessing) is not supported on the Common Lisp implementation (only network streamers require threads).

See also

define-streamer-node (page 573)
find-streamer-node (page 575)
make-journal-streamer (page 569)
open-network-streamer (page 578)
open-streamer-p (page 544)
stream-of (page 552)

Example

Close the network streamer `*streamer*`:

```
(close-streamer *streamer*)
```

make-broadcast-streamer *streamers* ⇒ *broadcast-streamer* [*Function*]

Purpose

Create a broadcast streamer that can be used to write to multiple journal and network streamers.

Alternate syntax

make-broadcast-streamer `&key` *package read-default-float-format external-format* ⇒ *broadcast-streamer*

Package `:gbbopen`

Module `:streaming`

Arguments

streamers	A list of journal or network streamers.
package	A package designator indicating the package to be used when writing to and reading from the streamers associated with the *broadcast-streamer* (default is `:common-lisp`)
read-default-float-format	One of the atomic type specifiers `short-float`, `single-float`, `double-float`, or `long-float` to be used when to and reading from the streamers associated with the *broadcast-streamer* (default is `single-float`)
external-format	An external-file-format designator (default is `:default`)
broadcast-streamer	A broadcast streamer

Errors

See also

add-to-broadcast-streamer (page 540)
remove-from-broadcast-streamer (page 546)
stream-of (page 552)

Examples

Create a broadcast streamer containing network streamers to streamer nodes `"you"` and `"you-too"`:

```
> (make-broadcast-streamer
    (open-network-streamer "me" "you")
    (open-network-streamer "me" "you-too"))
#<broadcast-streamer 2 constituents>
>
```

Create a broadcast streamer with no constituents to be used with UTF-8 streamers:

```
> (make-broadcast-streamer :external-format ':utf-8)
#<broadcast-streamer 0 constituents>
>
```

open-streamer-p *streamer* ⇒ *boolean* [*Function*]

Purpose
Determine if a journal or network streamer is open.

Package :gbbopen

Module :streaming

Arguments
streamer A journal or network streamer

Returns
True if the streamer is open; otherwise `nil`.

See also

close-streamer (page 542)
find-streamer-node (page 575)
make-journal-streamer (page 569)
open-network-streamer (page 578)
stream-of (page 552)

Example
Check if the network streamer `*streamer*` is open:

```
> (open-streamer-p *streamer*)
t
```

read-queued-streaming-block *tag string-stream* [*Generic Function*]

Purpose
Reads a queued-streaming block.

Method signatures
read-queued-streaming-block `(`*tag* `t)` *string-stream*

Package `:gbbopen`

Module `:streaming`

Arguments
tag An object
string-stream An input string stream

Description
read-queued-streaming-block is called when a queued-streaming block is read; it should not be called directly.

See also
with-queued-streaming (page 563)
write-streamer-queue (page 565)

Examples
Write a method that prints the time a queued-streaming block was started (recorded as the block's *tag*) when the block is read from a journal file or network-streamer connection:

```
(defmethod read-queued-streaming-block :around ((tag integer)
                                                string-stream)
  (declare (ignorable string-stream))
  (format t "~&;; Reading ~a queued-streaming block...~%"
            (full-date-and-time tag))
  (call-next-method)
  (format t "~&;; Reading queued-streaming block completed.~%"))
```

Write a method that prints a notice when an empty queued-streaming block (tagged with the *tag* `:empty`) has been read:

```
(defmethod read-queued-streaming-block :after ((tag (eql ':empty))
                                               string-stream)
  (declare (ignorable string-stream))
  (format t "~&;; An empty queued-streaming block was read.~%"))
```

remove-from-broadcast-streamer *streamer broadcast-streamer* [*Function*]

Purpose

Remove a journal or network streamer from a broadcast streamer.

Package :gbbopen

Module :streaming

Arguments

streamer	A journal or network streamer.
broadcast-streamer	A broadcast streamer

Errors

See also

make-broadcast-streamer (page 543)
add-to-broadcast-streamer (page 540)

Example

Remove the network streamer to streamer node `"you-too"` from broadcast streamer `broadcast-streamer`:

```
> (remove-from-broadcast-streamer
    (open-network-streamer "me" "you-too")
    broadcast-streamer)
#<broadcast-streamer 1 constituent>
>
```

stream-add-instance-to-space-instance *unit-instance space-instance-or-path streamer* ⇒ *instance* [*Function*]

Purpose
Write adding a unit instance to a space instance to a streamer.

Package
`:gbbopen`

Module
`:streaming`

Arguments
unit-instance The added unit instance
space-instance-or-path The space instance or space-instance path to which the unit instance is added
streamer A streamer

Returns
The *unit-instance*.

See also

stream-delete-instance (page 548)
stream-instance (page 549)
stream-instances (page 550)
stream-instances-of-class (page 551)
stream-instances-on-space-instances (page 553)
stream-remove-instance-from-space-instance (page 558)

Examples
Stream to streamer `*streamer*` the addition of a highly plausible hypothesis unit instance, `good-hyp`, to the `hyps` space instance:

```
> (stream-add-instance-to-space-instance
    good-hyp (find-space-instance-by-path '(bb hyps)))
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

or

```
> (stream-add-instance-to-space-instance good-hyp '(bb hyps))
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

stream-delete-instance *unit-instance streamer* ⇒ *instance* [*Function*]

Purpose
Write deleting a unit instance (or space instance) to a streamer.

Package :gbbopen

Module :streaming

Arguments
unit-instance A unit instance (or space instance)
streamer A streamer

Returns
The *unit-instance*.

See also

stream-add-instance-to-space-instance (page 547)
stream-instance (page 549)
stream-instances (page 550)
stream-instances-of-class (page 551)
stream-instances-on-space-instances (page 553)
stream-remove-instance-from-space-instance (page 558)

Example
Stream deleting a `hyp` unit instance to streamer `*streamer*`:

```
> (stream-delete-instance (find-instance-by-name 419 'hyp) *streamer*)
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

stream-instance *unit-instance streamer* ⇒ *instance* [*Function*]

Purpose
Write a unit instance (or space instance) to a streamer.

Package :gbbopen

Module :streaming

Arguments
unit-instance A unit instance (or space instance)
streamer A streamer

Returns
The *unit-instance*.

See also

stream-add-instance-to-space-instance (page 547)
stream-delete-instance (page 548)
stream-instances (page 550)
stream-instances-of-class (page 551)
stream-instances-on-space-instances (page 553)
stream-remove-instance-from-space-instance (page 558)

Example
Stream a `hyp` unit instance to streamer `*streamer*`:

```
> (stream-instance (find-instance-by-name 419 'hyp) *streamer*)
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

stream-instances *instances streamer* ⇒ *instances* [*Function*]

Purpose
Write a list of unit instances (and space instances) to a streamer.

Package :gbbopen

Module :streaming

Arguments
instances A list of unit instances (and space instances)
streamer A streamer

Returns
The *instances*.

See also

stream-add-instance-to-space-instance (page 547)
stream-delete-instance (page 548)
stream-instance (page 549)
stream-instances-of-class (page 551)
stream-instances-on-space-instances (page 553)
stream-remove-instance-from-space-instance (page 558)

Example
Stream the results of a blackboard retrieval to streamer *streamer*:

```
  > (stream-instances (find-instances 'hyp (find-space-instance-by-path '(bb
hyps))
                      '(and (= x 1835) (> belief .8)))
      *streamer*)
  (#<hyp 319 (1835 8419) 0.91 [4..12]>
   #<hyp 331 (1835 8419) 0.88 [15..30]>
   #<hyp 335 (1835 8419) 0.92 [15..35]>
   #<hyp 183 (1835 4791) 0.82 [0..35]>
   #<hyp 233 (1835 4791) 0.89 [5..35]>
   #<hyp 419 (1835 4791) 0.85 [5..35]>)
  >
```

stream-instances-of-class *unit-classes-specifier streamer* [*Function*]

Purpose
Write all unit instances of the specified unit classes to a streamer.

Package :gbbopen

Module :streaming

Arguments
unit-classes-specifier An extended unit-classes specification (see below)
streamer A streamer

Detailed syntax
unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*$^{+}$`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

See also

stream-add-instance-to-space-instance (page 547)
stream-delete-instance (page 548)
stream-instance (page 549)
stream-instances (page 550)
stream-instances-on-space-instances (page 553)
stream-remove-instance-from-space-instance (page 558)

Example
Stream all unit instances of the unit class `hyp` to streamer `*streamer*`:

```
(stream-instances-of-class 'hyp *streamer*)
```

stream-of *streamer* ⇒ *stream* [*Generic Reader*]

Purpose
Return the stream associated with a journal or network streamer.

Package :gbbopen

Module :streaming

Arguments
streamer A journal or network streamer

Returns
The associated stream if the streamer is open; otherwise `nil`.

See also
close-streamer (page 542)
find-streamer-node (page 575)
make-journal-streamer (page 569)
open-network-streamer (page 578)
open-streamer-p (page 544)

Example
Get the output file stream of the journal streamer `*streamer*`:

```
> (stream-of *streamer*)
#<output file stream "<homedir>/tutorial.jnl">
```

stream-instances-on-space-instances *unit-classes-specifier space-instances streamer* [*Function*]
`&key` *pattern filter-before filter-after*
use-marking verbose

Purpose

Write unit instances on space instances, optionally selected by a retrieval pattern, to a streamer.

Package `:gbbopen`

Module `:streaming`

Arguments

unit-classes-specifier An extended unit-classes specification (see below)

space-instances A space instance, a list of space instances, a space-instance path regular expression, or `t` (indicating all space instances)

streamer A streamer

pattern A retrieval pattern (see below; default is `t`)

filter-before A single-argument predicate to be applied before pattern-matching tests occur

filter-after A single-argument predicate to be applied after pattern-matching tests occur

use-marking A generalized boolean (default is ***use-marking***)

verbose A generalized boolean (default is ***find-verbose***)

Detailed syntax

unit-classes-specifier ::= `t` | *single-unit-class-specifier* | `(`*single-unit-class-specifier*$^{+}$`)`
single-unit-class-specifier ::= *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand `+` subclasses specifier is equivalent to `:plus-subclasses` and `=` to `:no-subclasses`.

pattern ::= *subpattern* | `t` | `:all`
subpattern ::= *pattern-element* |
`(not` *subpattern*`)` |
`(and` *subpattern**`)` |
`(or` *subpattern**`)`
pattern-element ::= `(`*pattern-op dimension-names pattern-values option**`)` |
`(`*boolean-dimension-unary-pattern-op dimension-names option**`)`
pattern-op ::= *ordered-dimension-pattern-op* |
enumerated-dimension-pattern-op |
boolean-dimension-binary-pattern-op
ordered-dimension-pattern-op ::= *ordered-dimension-any-numeric-value-pattern-op* |
ordered-dimension-explicit-type-pattern-op
ordered-dimension-explicit-type-pattern-op ::= *ordered-dimension-fixnum-pattern-op* |
ordered-dimension-short-float-pattern-op |
ordered-dimension-single-float-pattern-op |
ordered-dimension-double-float-pattern-op |
ordered-dimension-long-float-pattern-op |
ordered-dimension-pseudo-probability-pattern-op

ordered-dimension-any-numeric-value-pattern-op ::= `<` | `<=` | `>=` | `>` | `=` | `/=` |
`within` | `covers` | `overlaps` |
`abuts` | `starts` | `ends`
ordered-dimension-fixnum-pattern-op ::= `<&` | `<=&` | `>=&` | `>&` | `=&` | `/=&` |
`within&` | `covers&` | `overlaps&` |
`abuts&` | `starts&` | `ends&`
ordered-dimension-short-float-pattern-op ::= `<$&` | `<=$&` | `>=$&` | `>$&` | `=$&` | `/=$&` |
`within$&` | `covers$&` | `overlaps$&` |
`abuts$&` | `starts$&` | `ends$&`
ordered-dimension-single-float-pattern-op ::= `<$` | `<=$` | `>=$` | `>$` | `=$` | `/=$` |
`within$` | `covers$` | `overlaps$` |
`abuts$` | `starts$` | `ends$`
ordered-dimension-double-float-pattern-op ::= `<$$` | `<=$$` | `>=$$` | `>$$` | `=$$` | `/=$$` |
`within$$` | `covers$$` | `overlaps$$` |
`abuts$$` | `starts$$` | `ends$$`
ordered-dimension-long-float-pattern-op ::= `<$$$` | `<=$$$` | `>=$$$` | `>$$$` | `=$$$` | `/=$$$` |
`within$$$` | `covers$$$` | `overlaps$$$` |
`abuts$$$` | `starts$$$` | `ends$$$`
ordered-dimension-pseudo-probability-pattern-op ::= `<%` | `<=%` | `>=%` | `>%` | `=%` | `/=%` |
`within%` | `covers%` | `overlaps%` |
`abuts%` | `starts%` | `ends%`
enumerated-dimension-pattern-op ::= `is` | *enumerated-dimension-explicit-test-pattern-op*
enumerated-dimension-explicit-test-pattern-op ::= `is-eq` | `is-eql` | `is-equal` | `is-equalp`
boolean-dimension-binary-pattern-op ::= `eqv`
boolean-dimension-unary-pattern-op ::= `true` | `false`
dimension-names ::= *dimension-name* | `(`*dimension-name*$^{+}$`)`
pattern-values ::= *pattern-value* |
`(`*pattern-value*$^{+}$`)` |
`(`*pattern-value*$^{+}$ `.` *pattern-value*`)` |
`#(`*pattern-value*$^{+}$`)`
pattern-value ::= *point* | *interval* | *element* | *set*
interval ::= `(`*start end*`)` | `(`*start* `.` *end*`)` | `#(`*start end*`)`

Terms

point A number, `infinity`, or `-infinity`
start A number or `-infinity`
end A number or `infinity`
element An object

Description

A unit instance will be streamed only once, even if the unit instance resides on multiple space instances.

The *pattern* `t` matches all unit instances whose dimension values overlap the dimensional extent of at least one space instance in *space-instances*. The *pattern* `:all` matches every unit instance on a space instance in *space-instances*, regardless of dimensional overlap.

Declared numeric (see page 143) and pseudo probability (see page 149) pattern operators are also supported, for example: `=&`, `=$&`, `=$`, `=$$`, `=$$$`, and `=%` and `within&`, `within$&`, `within$`, `within$$`, `within$$$`, and `within%`.

See also

stream-add-instance-to-space-instance (page 547)
stream-delete-instance (page 548)
stream-instance (page 549)
stream-instances (page 550)
stream-instances-of-class (page 551)
stream-remove-instance-from-space-instance (page 558)

Example

Stream all unit instances of the unit class `hyp` to streamer `*streamer*` that reside on the `(bb probable-hyps)` space instance that have a belief value greater than 0.5:

```
(stream-instances-on-space-instances 'hyp '(bb probable-hyps) *streamer*
   :pattern '(> belief .5))
```

Note

Fixnum `overlaps` comparisons can result in bignum computations if the combined intervals of the pattern and a candidate unit instance exceeds `most-positive-fixnum`.

stream-link *unit-instance slot other-unit-instances streamer* ⇒ *other-unit-instances* [*Function*]

Purpose

Write links added between a unit instance and one or more unit instances to a streamer.

Package

`:gbbopen`

Module

`:streaming`

Arguments

unit-instance A unit instance (or space instance)

slot A non-`nil`, non-keyword symbol naming the link slot or a slot meta object

other-unit-instances A unit instance, a link-pointer object, or a list of unit instances and link-pointer objects

streamer A streamer

Returns

The supplied *other-unit-instances*.

See also

stream-nonlink-slot-update (page 557)
stream-unlink (page 559)

Example

Stream adding a link of `support-hyp` to the `supporting-hyps` link slot of the `hyp` unit instance `unit-instance` to streamer `*streamer*`:

```
> (stream-link unit-instance 'supporting-hyps support-hyp *streamer*)
#<hyp 231 (1488 7405) 0.63 [0..8]>
>
```

stream-nonlink-slot-update *unit-instance slot new-value streamer* ⇒ *new-value* [*Function*]

Purpose
Write an update to the value of a non-link slot of unit instance (or space instance) to a streamer.

Package :gbbopen

Module :streaming

Arguments
unit-instance	A unit instance (or space instance)
slot	A non-`nil`, non-keyword symbol naming the slot or a slot meta object
new-value	An object
streamer	A streamer

Returns
The supplied *new-value*.

See also
stream-link (page 556)
stream-unlink (page 559)

Example
Stream a new belief value for a `hyp` unit instance to streamer `*streamer*`:

```
> (stream-nonlink-slot-update (find-instance-by-name 419 'hyp)
    'belief 0.88 *streamer*)
0.88
>
```

stream-remove-instance-from-space-instance *unit-instance space-instance-or-path streamer* ⇒ *instance* [*Function*]

Purpose
Write removing a unit instance from a space instance to a streamer.

Package
`:gbbopen`

Module
`:streaming`

Arguments
unit-instance The removed unit instance

space-instance-or-path The space instance or space-instance path from which the unit instance is removed

streamer A streamer

Returns
The *unit-instance*.

See also

stream-add-instance-to-space-instance (page 547)
stream-delete-instance (page 548)
stream-instance (page 549)
stream-instances (page 550)
stream-instances-of-class (page 551)
stream-instances-on-space-instances (page 553)

Examples

Examples
Stream to streamer `*streamer*` the removal of an incorrect hypothesis unit instance, `incorrect-hyp`, from the `hyps` space instance:

```
> (stream-remove-instance-from-space-instance
    incorrect-hyp (find-space-instance-by-path '(bb hyps)))
#<hyp 311 (896 388) 0.68 [0..6]>
>
```

or

```
> (stream-remove-instance-from-space-instance incorrect-hyp '(bb hyps))
#<hyp 311 (896 388) 0.68 [0..6]>
>
```

stream-unlink *unit-instance slot other-unit-instances streamer* ⇒ *other-unit-instances* [*Function*]

Purpose

Write links removed between a unit instance and one or more unit instances to a streamer.

Package

`:gbbopen`

Module

`:streaming`

Arguments

unit-instance A unit instance (or space instance)

slot A non-`nil`, non-keyword symbol naming the link slot or a slot meta object

other-unit-instances A unit instance, a link-pointer object, or a list of unit instances and link-pointer objects

streamer A streamer

Returns

The supplied *other-unit-instances*.

See also

stream-link (page 556)

stream-nonlink-slot-update (page 557)

Example

Stream removing a link of `support-hyp` from the `supporting-hyps` link slot of the `hyp` unit instance `unit-instance` to streamer `*streamer*`:

```
> (stream-unlink unit-instance 'supporting-hyps support-hyp *streamer*)
#<hyp 231 (1488 7405) 0.63 [0..8]>
>
```

remove-mirroring *streamer* [*unit-class-or-instance-specifier*]] `&key` *slot-names paths* [*Function*]

Purpose
Remove mirroring of one or more unit classes.

Package `:gbbopen`

Module `:streaming`

Arguments
streamer	A streamer
unit-class-or-instance-specifier	An extended unit-class or instance specification (see below; default is `t`)
slot-names or slot-name	A slot-name or list of slot-names (default is `t`)
paths or path	A space-instance path regular expression (default is `(*)`)

Detailed syntax
unit-class-or-instance-specifier ::= *unit-instance* | `(`*unit-instance*`*``)` |
atomic-unit-class |
`(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description
The *paths* argument is either the symbol `t` (indicating all space instances) or a list representing a regular expression where the following reserved symbols are interpreted as follows:
- `=` matches one occurrence in a space-instance path
- `?` matches zero or one occurrence in a space-instance path
- `+` matches one or more occurrences in a space-instance path
- `*` matches zero or more occurrences in a space-instance path
- `^` move to parent

See also

stream-add-instance-to-space-instance (page 547)
add-mirroring (page 539)
stream-remove-instance-from-space-instance (page 558)

Example
Remove full mirroring of `hyp` unit instances to *streamer*:

```
(remove-mirroring streamer 'hyp)
```

Note
Unit-instance-specific mirroring is not yet implemented in GBBopen.

with-mirroring-disabled (*option**) *declaration** *form** ⇒ *result** [*Macro*]

Purpose
Disable mirroring during evaluation of *forms*.

Package :gbbopen

Module :streaming

Arguments
option No options are currently supported
declaration A declare expression (not evaluated)
forms An implicit **progn** of forms to be evaluated
results The values returned by evaluating the last *form*

Returns
The values returned by evaluating the last *form*.

See also
add-mirroring (page 539)
remove-mirroring (page 560)
with-mirroring-enabled (page 562)

Example
Create a hyp without mirroring:

```
> (with-mirroring-disabled ()
    (make-instance 'hyp
      :location (list x y)
      :classification '(:car :truck)
      :color ':red
      :belief .85
      :velocity-range '(5 35)
      :supporting-hyps supporting-hyps))
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

with-mirroring-enabled (*option**) *declaration** *form** ⇒ *result** [*Macro*]

Purpose
Restore mirroring during evaluation of *forms*.

Package :gbbopen

Module :streaming

Arguments
option No options are currently supported
declaration A declare expression (not evaluated)
forms An implicit **progn** of forms to be evaluated
results The values returned by evaluating the last *form*

Returns
The values returned by evaluating the last *form*.

See also
add-mirroring (page 539)
remove-mirroring (page 560)
with-mirroring-disabled (page 561)

Example
Create a hyp without mirroring, then add supporting-hypothesis links with mirroring:

```
> (with-mirroring-disabled
    (let ((hyp (make-instance 'hyp
                 :location (list x y)
                 :classification '(:car :truck)
                 :color ':red
                 :belief .85
                 :velocity-range '(5 35))))
      (with-mirroring-enabled ()
        (linkf (supporting-hyps-of hyp) supporting-hyps))
      hyp))
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

with-queued-streaming (*streamer* [*tag* [*write-empty-queue-p*]]) *form** ⇒ *result** [*Macro*]

Purpose
Queue streaming during evaluation of *forms*.

Package :gbbopen

Module :streaming

Arguments
streamer	A streamer
tag	An object (default is `nil`)
write-empty-queue-p	A generalized boolean (default is `nil`)
forms	An implicit **progn** of forms to be evaluated
results	The values returned by evaluating the last *form*

Returns
The values returned by evaluating the last *form*.

Description
Instead of writing changes directly to a streamer's journal file or network connection, the changes are queued until the last *form* has been evaluated. Then the queued changes are written as a block that is marked with the *tag* value. Queued streaming is thread-local (so different threads can have their own open queues at the same time) and **with-queued-streaming** forms can be nested within a single thread.

If the value of *write-empty-queue-p* is true, the streamer queue is written even if it is empty (and **read-queued-streaming-block** will be called with the *tag* value when the empty queue is received at a streamer node or read during journal loading).

See also

clear-streamer-queue (page 541)
read-queued-streaming-block (page 545)
write-streamer-queue (page 565)

Examples
Assuming that `hyp` unit instances are being mirrored to streamer `*streamer*`, queue the mirroring of a created `hyp` tagged with the space instance `(bb hyps)`:

```
  > (with-queued-streaming (*streamer* (find-space-instance-by-path '(bb
hyps)))
       (make-instance 'hyp
          :location (list x y)
          :classification '(:car :truck)
          :color ':red
          :belief .85
          :velocity-range '(5 35)
          :supporting-hyps supporting-hyps))
  #<hyp 419 (1835 4791) 0.85 [5..35]>
```

```
>
```

Write an empty queue (with tag `:empty`) to streamer `*streamer*`:

```
> (with-queued-streaming (*streamer* ':empty 't))
nil
>
```

---

with-queued-streaming

write-streamer-queue *streamer* `&key` *tag write-empty-queue-p* [*Function*]

Purpose

Write the contents of a streamer queue, emptying it for further queueing.

Package

`:gbbopen`

Module

`:streaming`

Arguments

streamer	A streamer
tag	An object (default is `nil`)
write-empty-queue-p	A generalized boolean (default is `nil`)

See also

clear-streamer-queue (page 541)
read-queued-streaming-block (page 545)
with-queued-streaming (page 563)

Examples

Write the contents of streamer `*streamer*`, using and retaining the existing *tag* and *write-empty-queue-p* values associated with the queue:

```
(write-streamer-queue *streamer*)
```

Write the contents of streamer `*streamer*`, using the existing *tag* and *write-empty-queue-p* values associated with the queue (and retaining the *write-empty-queue-p* value), but establishing a new *tab* value for the emptied queue:

```
(write-streamer-queue *streamer* :tag (get-universal-time))
```

6.2 Journaling

Journal writing and reading is provided by the `:streaming` module.

The entities in this module are experimental and are subject to change.

load-journal *pathname* `&key` *class-name-translations coalesce-strings disable-events estimated-peak-forward-references external-format readtable read-eval retain-classes retain-event-functions retain-event-printing* [*Function*]
⇒ *pathname, saved-time, saved-value*

Purpose
Load a journal file.

Package
`:gbbopen`

Module
`:streaming`

Arguments
pathname	A pathname designator
class-name-translations	An association list (default is `nil`)
coalesce-strings	A generalized boolean (default is `nil`)
disable-events	A generalized boolean (default is `t`)
estimated-peak-forward-references	An integer (default is `*default-estimated-peak-forward-references*`)
external-format	An external-file-format designator (default is `:default`)
readtable	A readtable (default is `*reading-saved/sent-objects-readtable*`)
read-eval	A generalized boolean (default is `nil`)
retain-classes	An extended unit-classes specification (see below)
retain-event-functions	A generalized boolean (default is `nil`)
retain-event-printing	A generalized boolean (default is `nil`)
pathname	A pathname
saved-time	A Universal Time
saved-value	An object

Returns
Three values: the pathname of the journal file, the time when the journal was started, and the save value specified when the journal was created.

Description
If *pathname* does not specify a file type, the type `jnl` is added to it. Then, `(user-homedir-pathname)` is used to supply any missing components to *pathname*.

The *class-name-translations* association list, if specified, should contain conses of the form:

```
(class-name . new-class-name)
```

for any class translations that should occur during repository loading.

If *coalesce-strings* is true, loaded strings that are `equal` become shared (`eq`). This coalescing is performed using a temporary hash table whose initial size can be specified by providing an integer value for *coalesce-strings*. If a hash table is provided as the value for *coalesce-strings*, it is used in place of the temporary hash table.

Unit instances that are referenced before they are defined are recorded using a temporary hash table whose initial size can be specified by providing an integer value for *estimated-peak-forward-references*.

See also

make-journal-streamer (page 569)
save-blackboard-repository (page 514)
with-reading-saved/sent-objects-block (page 516)

Example

Load the GBBopen Tutorial Example application (without running it) and then load a journal that was written previously:

```
> :tutorial-example :noautorun
   ...
> (load-journal "tutorial")
;; 35 temporarily forward-referenced instances (peak count)
#P"<homedir>/tutorial.jnl"
3429178245
#<path 1>
>
```

---

load-journal

make-journal-streamer *pathname* `&key` *external-format package read-default-float-format value* ⇒ *journal-streamer* [*Function*]

Purpose

Create a journal streamer to record to a file changes made to unit instances (and space instances).

Package `:gbbopen`

Module `:streaming`

Arguments

pathname	A pathname designator
external-format	An external-file-format designator (default is `:default`)
package	A package designator indicating the package to be used when writing the journal and when loading the journal file (default is `:common-lisp`)
read-default-float-format	One of the atomic type specifiers `short-float`, `single-float`, `double-float`, or `long-float` to be used when writing the journal and loading the journal file (default is `single-float`)
value	An object to be saved with the journal and returned when the journal file is loaded (default is `nil`)
journal-streamer	A journal streamer

Returns

The created journal streamer.

Description

If *pathname* does not specify a file type, the type `jnl` is added to it. Then, `(user-homedir-pathname)` is used to supply any missing components to *pathname*.

The size of the journal file can be reduced by specifying a *package* containing the majority of the symbols that are written to the file.

A **with-saving/sending-block**, with the *package* and *read-default-float-format* values, is established when writing to the journal.

See also

load-journal (page 567)
omitted-slots-for-saving/sending (page 510)
stream-instance (page 549)
save-blackboard-repository (page 514)
with-saving/sending-block (page 518)

Example

Load the `:streaming` module followed by the GBBopen Tutorial Example application (without running it). Then write a journal file recording what occurred when taking a walk:

```
> :streaming
   ...
> :tutorial-example :noautorun
```

```
   ...
> (defparameter *streamer*
    (make-journal-streamer "tutorial" :package ':tutorial))
*streamer*
> *streamer*
#<journal-streamer "<homedir>/tutorial.jnl">
> (add-mirroring *streamer* 'standard-space-instance)
nil
> (add-mirroring *streamer* 'path)
nil
> (add-mirroring *streamer* 'location)
nil
> (take-a-walk)
;; Control shell 1 started
   ...
;; Explicit :stop issued by KS print-walk-ks
;; Control shell 1 exited: 41 cycles completed
;; Run time: 0.01 seconds
;; Elapsed time: 0 seconds
:stop
>
```

---

make-journal-streamer

6.3 Network Streaming

The `:network-streaming` module provides network streaming entities.

The entities in this module are experimental and are subject to change.

default-network-stream-server-port [*Variable*]

Purpose

Specifies the default service port for the network stream server at a streamer node.

Package

`:gbbopen`

Module

`:network-streaming`

Value type

An integer or a string specifying the service port

Initial value

`1968`

See also

define-streamer-node (page 573)

define-streamer-node *name* `&key` *host port documentation external-format package read-default-float-format* ⇒ *streamer-node* [*Macro*]

Purpose

Define or redefine a streamer node.

Package `:gbbopen`

Module `:network-streaming`

Arguments

name	An object naming the streamer node (not evaluated)
host	A 32-bit internet address or a string specifying the remote host (default `"localhost"`)
port	An integer or a string specifying the service port (default is ***default-network-stream-server-port***)
documentation	A documentation string or `nil` (default is `nil`)
passphrase	A string or `nil` (default is `nil`)
external-format	An external-file-format designator (default is `:default`)
package	A package designator indicating the package to be used when establishing a streaming connection to *node* (default is `:common-lisp`)
read-default-float-format	One of the atomic type specifiers `short-float`, `single-float`, `double-float`, or `long-float` to be used when establishing a streaming connection to *streamer-node* (default is `single-float`)
authorized-nodes	Needed (default is `:all`)
accepted-streamer-node-class	Needed (default is `accepted-streamer-node`)
accepted-streamer-node-initargs	Needed (default is `nil`)

Returns

The newly defined or modified streamer node object.

See also

default-network-stream-server-port (page 572)
close-streamer (page 542)
find-streamer-node (page 575)
open-network-streamer (page 578)
open-streamer-p (page 544)
stream-of (page 552)

Example

Define a streamer node:

```
> (define-streamer-node "me"
    :host "127.0.0.1"
    :package ':gbbopen-user
    :passphrase "Who goes there?"
    :authorized-nodes '("you"))
```

```
#<streamer-node "me">
>
```

Note

Modifications to a streamer node object apply the next time that a network streamer is created for the streamer node—the changes to not affect an established network streamer.

---

define-streamer-node

find-streamer-node *name* ⇒ *streamer-node* or `nil` [*Function*]

Purpose
Return a streamer node given its name.

Package `:gbbopen`

Module `:network-streaming`

Arguments
name	An object naming the streamer node.
errorp	A generalized boolean (default is `nil`)
streamer-node	A streamer node

Returns
The streamer node named *name* or `nil`, if none has been defined.

See also
define-streamer-node (page 573)
open-network-streamer (page 578)

Examples
Return the streamer node named `"master"`:

```
> (find-streamer-node "master")
#<streamer-node "master">
>
```

Find a non-existent streamer node:

```
> (find-streamer-node "missing")
nil
> (find-streamer-node "missing" 't)
Error: No streamer node named "missing"
>>
```

kill-network-stream-server *streamer-node* ⇒ *thread* [*Function*]

Purpose
Create a connection-server thread that accepts network-stream connections.

Package
`:gbbopen`

Module
`:network-streaming`

Arguments
streamer-node A local streamer node or its name

Returns
True if the network-stream connection server was running when killed; `nil` otherwise.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also

define-streamer-node (page 573)
find-streamer-node (page 575)
network-stream-server-running-p (page 577)
start-network-stream-server (page 579)

Example
Kill the network-stream connection server running at local streamer node `"me"`:

```
> (kill-network-stream-server "me")
t
>
```

network-stream-server-running-p *streamer-node* ⇒ *boolean* [*Function*]

Purpose
Determine if a network-stream connection server for a local streamer node is running.

Package
`:gbbopen`

Module
`:network-streaming`

Arguments
streamer-node A local streamer node or its name
boolean A generalized boolean

Returns
True if the network-stream connection server is alive; `nil` otherwise.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
define-streamer-node (page 573)
find-streamer-node (page 575)
kill-network-stream-server (page 576)
start-network-stream-server (page 579)

Example
See if the a network-stream connection server streamer node `"me"` is running:

```
> (network-stream-server-running-p "me")
t
>
```

open-network-streamer *streamer-node local-streamer-node* `&rest` *initargs* [*Function*]
⇒ *network-streamer*

Purpose

Return a connection to an external streamer node (from the specified local streamer node), establishing a new connection if one does not already exist.

Package

`:gbbopen`

Module

`:network-streaming`

Arguments

streamer-node	A non-local streamer node or its name
local-streamer-node	A local streamer node or its name
initargs	An initialization argument list
network-streamer	A network streamer

Returns

An open network streamer (either existing or newly created—see below).

Description

If an open network streamer exists between *streamer-node* and *local-streamer-node*, that network streamer is returned. Otherwise, a new connection to *streamer-node* is established and a new network streamer is returned.

The class of the created network streamer is specified by the `streamer-class` slot of *streamer-node*.

See also

close-streamer (page 542)
define-streamer-node (page 573)
find-streamer-node (page 575)
network-stream-server-running-p (page 577)
open-streamer-p (page 544)
stream-of (page 552)

Example

Return a network streamer connecting streamer node `"me"` and streamer node `"you"`:

```
> (open-network-streamer "me" "you")
#<network-streamer 127.0.0.1:1969>
>
```

start-network-stream-server *streamer-node* ⇒ *thread* [*Function*]

Purpose
Create a connection-server thread that accepts network-stream connections to a local streamer node.

Package
`:gbbopen`

Module
`:network-streaming`

Arguments
streamer-node A local streamer node or its name
thread A thread

Returns
The new connection-server thread thread.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
define-streamer-node (page 573)
find-streamer-node (page 575)
kill-network-stream-server (page 576)

Example
Start a network-stream connection server at local streamer node `"me"`:

```
> (start-network-stream-server "me")
#<thread Network Connection Server>
>
```

7 Agenda Control Shell

The Agenda Shell module, `:agenda-shell`, provides a responsive, agenda-based control shell.

Note that the Agenda Shell requires that the idle-loop process has been started on CMUCL and that multiprocessing has been started on LispWorks. (An error message when the Agenda Shell is started will instruct you on what to do if this is not the case.)

abort-ks-execution <no arguments> [*Function*]

Purpose

Abort the currently executing KSA.

Package

`:agenda-shell`

Module

`:agenda-shell`

See also

exit-control-shell (page 604)

Example

Abort the currently executing KSA::

```
(abort-ks-execution)
```

activation-cycle-of *ksa* ⇒ *cycle-number* [*Generic Reader*]

Purpose
Returns the cycle number when a KSA was activated.

Method signatures
activation-cycle-of (*ksa* `ksa`) ⇒ *cycle-number*

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
ksa	A KSA
cycle-number	An integer

Returns
The activation cycle number of *ksa*

Description
This generic function accesses the value stored in the `activation-cycle` nonlink slot of *ksa*. This value is maintained by the Agenda Shell and should not be changed.

See also
ksa (page 609)

Example
Return the activation cycle of `ksa`:

```
> (activation-cycle-of ksa)
1192
>
```

collect-trigger-instances *source* ⇒ *trigger-instances* [*Generic Function*]

Purpose
Return the trigger unit instances of a KSA, an event, or a list of KSAs or events.

Method signatures
collect-trigger-instances (*cons* `cons`) ⇒ *trigger-instances*
collect-trigger-instances (*event* `single-instance-event`) ⇒ *trigger-instances*
collect-trigger-instances (*ksa* `ksa`) ⇒ *trigger-instances*
collect-trigger-instances (*event* `multiple-instance-event`) ⇒ *trigger-instances*
collect-trigger-instances (*event* `non-instance-event`) ⇒ `nil`

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
source	A KSA, event, or a list of KSAs or events
trigger-instances	A proper list

Returns
The list of trigger unit instances

See also
sole-trigger-instance-of (page 620)

Example
Return the trigger unit instances of a KSA:

```
> (collect-trigger-instances ksa)
(#<hyp 419 (1835 4791) 0.85 [5..35]>
 #<hyp 233 (1835 4791) 0.89 [5..35]>)
>
```

control-shell-running-p <no arguments> ⇒ *boolean* [*Function*]

Purpose
Determine if a control shell is running.

Package
`:agenda-shell`

Module
`:agenda-shell`

Arguments
boolean A generalized boolean

Returns
True if the control shell is running; `nil` otherwise.

See also
start-control-shell (page 622)
restart-control-shell (page 617)

Example
See if the control shell is running:

```
> (control-shell-running-p)
nil
>
```

current-control-shell <no arguments> ⇒ *control-shell-or-nil* [*Function*]

Purpose
Return the object representing the current control shell.

Package
`:agenda-shell`

Module
`:agenda-shell`

Arguments
control-shell-or-nil A `control-shell` object or `nil`

Returns
The object representing the current control shell (in the current thread), if one can be determined; otherwise `nil`.

See also
control-shell-running-p (page 585)
start-control-shell (page 622)

Example
```
> (current-control-shell)
#<control-shell 1>
>
```

define-ks *ks-name* `&key` *activation-predicate enabled execution-function ks-class ksa-class obviation-events obviation-predicate precondition-function rating retrigger-events retrigger-function revalidation-predicate trigger-events* ⇒ *ks* [*Macro*]

Purpose

Define or redefine a knowledge source (KS).

Package

`:agenda-shell`

Module

`:agenda-shell`

Arguments

ks-name	A symbol naming the KS (not evaluated)
activation-predicate	A function designator specifying a function object of two arguments (the KS unit instance and the event object) that returns a generalized boolean or `nil` (default is `nil`)
enabled	A generalized boolean (default is `t`)
execution-function	A function designator specifying a function object of one argument (the KSA unit instance) or `nil` (default is `nil`)
ks-class	A class or a symbol specifying a class (not evaluated)
ksa-class	A class or a symbol specifying a class (not evaluated)
obviation-events	An *event-specification* (see below, not evaluated)
obviation-predicate	A function designator specifying a function object of two arguments (the KSA unit instance and the event object) that returns a generalized boolean or `nil` (default is `nil`)
precondition-function	A function designator specifying a function object of two arguments (the KS unit instance and the event object) or `nil` (default is `nil`)
rating	A rating (default is `1`)
retrigger-events	An *event-specification* (see below, not evaluated)
retrigger-function	A function designator specifying a function object of two arguments (the KSA unit instance and the event object) or `nil` (default is `nil`)
revalidation-predicate	A function designator specifying a function object of one argument (the KSA unit instance) that returns a generalized boolean or `nil` (default is `nil`)
trigger-events	An *event-specification* (see below, not evaluated)
ks	A KS

Returns

The unit instance representing the KS

Detailed syntax

event-specification ::= `(`*event-signature*`*`)

event-signature ::= `(`*event-class-specifier*
[*unit-class-or-instance-specifier*
[{`:slot-name` *slot-name*} | {`:slot-names` *slot-names*} |
{`:path` *path*} | {`:paths` *paths*}]])

event-class-specifier ::= *atomic-event-class* | `(`*atomic-event-class subeventing-specifier*`)` | `t`

atomic-event-class ::= *event-class* | *event-class-name*
subeventing-specifier ::= `:plus-subevents` | `:no-subevents` | `+` | `=`

The shorthand + subevents specifier is equivalent to `:plus-subevents` and = to `:no-subevents`.

unit-class-or-instance-specifier ::= *unit-instance* | `(`*unit-instance*`*)` | *atomic-unit-class* | `(`*atomic-unit-class subclassing-specifier*`)` | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description

A KS definition creates a unit instance of class *ks-class* which specifies how activations of the KS are created and executed. The lifetime of each KS activation involves the following sequence:

- When an event matching one of the event specifications in *trigger-events* occurs and the KS is enabled:
  - the *activation-predicate*, if specified, is called and must return true for potential activation to continue
  - the *precondition-function*, if specified, is called and must return an integer rating for potential activation to continue
- The KS is activated (a unit instance of class *ksa-class* is created) and given the rating returned by the *precondition-function* or the constant *rating* value defined for the KS if no *precondition-function* was specified. The current control-shell cycle number is stored in the `activation-cycle` slot of the KSA unit instance.
- The KSA is placed on the queue of pending KSAs.
- If an event matching one of the event specifications in *obviation-events* occurs, the *obviation-predicate*, if specified, is called. If it returns true, the pending KSA is removed from the pending KSAs queue, the current control-shell cycle number is stored in the `obviation-cycle` slot of the KSA, and the KSA is placed on the queue of obviated KSAs.
- If an event matching one of the event specifications in *retrigger-events* occurs, the *retrigger-function*, if specified, is called. A *retrigger-function* is often used to change the triggering context of the KSA or its rating.
- When the pending KSA is selected for execution (typically because has the highest rating above the *minimum-ksa-execution-rating* currently in effect for the control shell), the *revalidation-predicate*, if specified, is called. If the *revalidation-predicate* returns `nil`, the pending KSA is removed from the pending KSAs queue, the current control-shell cycle number is stored in the `obviation-cycle` slot of the KSA, and the KSAis placed on the queue of obviated KSAs.
- The pending KSA is removed from the pending KSAs queue, the current control-shell cycle number is stored in the `execution-cycle` slot of the KSA unit instance, and the *execution-function* is called.
- The executed KSA is placed on the queue of executed KSAs.

KS functions and predicates

The Agenda Shell provides a rich set of KS functions and predicates to manage the progression of KSAs from initial triggering and activation through obviation or execution. A typical KS will only require a subset of these functions and predicates.

An *activation-predicate* is a function that is called with two arguments, the unit instance representing the KS and the object representing the triggering event. The *activation-predicate* should

return a generalized boolean that indicates whether the KS should continue to be considered for activation in response to the event. Typically, an *activation-predicate* is specified for a KS that does not require a *precondition-function* rating computation, but that does require an activate/don't-activate decision.

A *precondition-function* is a function that is called with two arguments, the unit instance representing the KS and the object representing the triggering event. The *precondition-function* should return one of the following sets of values:

- `nil` indicating the KS is not to be activated in response to the event
- `:stop` (and, optionally, additional values to be returned by the control shell) indicating that the control shell is to exit immediately
- An integer execution rating for the KSA (and, optionally, initialization arguments to be used when creating the KSA unit instance)

An *execution-function* is a function that implements the KS. When an activation of the KS is executed, this function is called with one argument, the unit instance representing the KSA. If the execution function returns the value `:stop` (and, optionally, a additional values to be returned by the control shell), the control shell will exit immediately.

An *obviation-predicate* is a function that is called with two arguments, the unit instance representing the KSA and the object representing the obviation event. The *obviation-predicate* should return a generalized boolean that indicates whether the KSA should be obviated.

A *retrigger-function* is a function that is called with two arguments, the unit instance representing the KSA and the object representing the retrigger event. The *retrigger-function* can perform whatever activities are needed in response to the event. Typically this involves augmenting the triggering context of the KSA or changing its execution rating.

A *revalidation-predicate* is a function that is called with one argument, the unit instance representing the KSA. The *revalidation-predicate* is called immediately before a KSA is executed and should return a generalized boolean that indicates whether the KSA should be executed (if true) or obviated (if false).

See also

define-ks-class (page 591)
define-ksa-class (page 595)
describe-ks (page 599)
ensure-ks (page 600)
ks (page 606)
ks-enabled-p (page 607)
standard-event-instance (page 411)
undefine-ks (page 627)

Examples

Define an initial KS that is triggered when the control shell is started:

```
(define-ks initial
   :trigger-events ((control-shell-started-event))
   :execution-function #'initial-ks-function)
```

Define a KS named `aggregate-hyps` that is triggered whenever a `hyp` unit instance is created:

```
(define-ks aggregate-hyps
   :trigger-events ((instance-created-event hyp))
   :precondition-function #'aggregate-hyps-precondition-function
   :execution-function #'aggregate-hyps-ks-function)
```

Note

Unit-instance-specific KS triggers are not yet implemented in GBBopen.

---

define-ks

define-ks-class *ks-class-name* ({*superclass-name*}*) ({*slot-specifier*}*) {*class-option*}* [*Macro*]
⇒ *new-ks-class*

Purpose
Define or redefine a ks class.

Package :agenda-shell

Module :agenda-shell

Arguments
ks-class-name A non-nil, non-keyword symbol that names the ks class

superclass-name A non-nil, non-keyword symbol that specifies a direct superclass of the ks class *ks-class-name*

slot-specifiers See below

class-options See below

new-ks-class A new or modified ks class object

Returns
The newly defined or modified ks class object.

Errors
The specified *superclass-names* do not include at least one ks class name. This error is signaled on class finalization.

Detailed syntax
(Syntax shown in gray is not supported in GBBopen Version 1.5, but will become available in a future release.)

slot-specifier ::= *slot-name* |
(*nonlink-slot-name* [[*nonlink-slot-option*]]) |
(*link-slot-name* [[*link-slot-option*]])
nonlink-slot-name ::= *slot-name*
link-slot-name ::= *slot-name*
link-slot-option ::= *slot-option* |
{:link *inverse-link-slot-specifier*} |
{:singular *boolean*} |
{:sort-function *function*} |
{:sort-key *function*}
inverse-link-slot-specifier ::= (*unit-class-name link-slot-name* [:singular *boolean*]) |
:reflexive
nonlink-slot-option ::= *slot-option* |
{:reader *reader-function-name*}* |
{:writer *writer-function-name*}*
slot-option ::= {:accessor *reader-function-name*}* |
{:allocation *allocation-type*} |
{:documentation *string*} |
{:initarg *initarg-name*}* |
{:initform *form*} |
{:type *type-specifier*}

class-option ::= (:abstract *boolean*) |
(:default-initargs . *initarg-list*) |
(:dimensional-values *dimension-value-specifier**) |
(:documentation *string*) |
(:estimated-instances *integer*) |
(:export-accessors *boolean*) |
(:export-class-name *boolean*) |
(:export-slot-names *direct-slots-specifier*) |
(:generate-accessors *direct-slots-specifier*) |
(:generate-accessors-format {:prefix | :suffix} |
(:generate-accessors-prefix {*string* | *symbol*}) |
(:generate-accessors-suffix {*string* | *symbol*}) |
(:generate-initargs *direct-slots-specifier*) |
(:initial-space-instances *initial-space-instance-specifier*) |
(:instance-name-comparison-test *instance-name-comparison-test*) |
(:metaclass *class-name*) |
(:retain {*boolean* | :propagate}) |
(:use-global-instance-name-counter *boolean*)
initial-space-instance-specifier ::= {*space-instance-path*+ | *function*}
dimension-value-specifier ::= *incomposite-dv-specifier* | *composite-dv-specifier*
incomposite-dv-specifier ::= (*dimension-name dimension-value-spec dimension-value-place*)
composite-dv-specifier ::= (*dimension-name dimension-value-specifier*
composite-type dimension-value-place)
composite-type ::= :set | :sequence |
{:ascending-series *ordering-dimension-name*} |
{:descending-series *ordering-dimension-name*}
dimension-value-specifier ::= *dimension-value-type* |
(*ordered-dimension-value-type* [*ordered-comparison-type*]) |
(*enumerated-dimension-value-type* [*enumerated-comparison-type*]) |
(*boolean-dimension-value-type* [*boolean-comparison-type*])
dimension-value-type ::= *ordered-dimension-value-type* |
enumerated-dimension-value-type |
boolean-dimension-value-type
ordered-dimension-value-type ::= :point | :interval | :mixed
enumerated-dimension-value-type ::= :element
boolean-dimension-value-type ::= :boolean
ordered-comparison-type ::= number | fixnum | short-float | single-float |
double-float | long-float |
pseudo-probability
enumerated-comparison-type ::= eq | eql | equal | equalp
boolean-comparison-type ::= t
dimension-value-place ::= {*slot-name* [*slot-name*]} | {*function* [*slot-name*]}
direct-slots-specifier ::= nil | t | *included-slot-name** |
{t :exclude *excluded-slot-name**}

The default *ordered-comparison-type*, if unspecified, is number. The default *enumerated-comparison-type*, if unspecified, is eql. The default *boolean-comparison-type* is t.

A *dimension-value-place* with two *slot-names* is allowed only for an :interval dimension-value specification.

Terms

class-name	A non-`nil`, non-keyword symbol that names a class
documentation	A documentation string
initarg-list	An initialization argument list
slot-name	A non-`nil`, non-keyword symbol
instance-name-comparison-test	One of the four standardized hash table test function names: `eq`, `eql`, `equal`, or `equalp` (default for classes of metaclass **standard-unit-class** is `eql`)

Description

A *dimension-value-place* with two *slot-names* can be specified only for `:interval` dimension-value types.

Each *superclass-name* argument specifies a direct superclass of the new class. If the superclass list is empty, then the direct superclass defaults to the single class **ks**.

The `:metaclass` *class-name* class option, if specified, must be a subclass of **standard-unit-class**. The default metaclass value is also **standard-unit-class**.

Inheritance of class options

The set of *dimensional-values* for a unit class is the union of the sets specified in the *dimensional-values* options of the class and its superclasses. When more than one dimension-value specification is supplied for a given dimension, the one supplied by the most specific class is used.

The effective *initial-space-instances* value for a unit class is the value specified in the definition of the most specific unit class. (No additive inheritance of *initial-space-instances* is performed.) If no definitions specify an *initial-space-instances* value, `nil` is used.

The *instance-name-comparison-test* value is not inherited. If no value is specified in the unit-class definition, the default initialization value associated with the metaclass is used.

If a *retain* value is not specified, a value of `:propagate` is used as the default if any parent unit classes have a `:propagate` retention value; otherwise `nil` is used as the default value.

The *use-global-instance-name-counter* value is not inherited. If no value is specified in the unit-class definition, the default initialization value associated with the metaclass is used.

See also

define-ks	(page 587)
delete-blackboard-repository	(page 442)
standard-unit-class	(page 369)
with-generate-accessors-format	(page 136)

Examples

Define a ks class, `ks-with-lock`, that has an additional slot containing a lock that can be used to synchronize operations on each defined KS of that class.

```
> (define-ks-class ks-with-lock ()
    ((lock :initform (make-lock :name "KS Lock"))))
#<standard-unit-class ks-with-lock>
>
```

Do the same, but with a mixin class:

```
> (define-unit-class ks-lock-mixin ()
    ((lock :initform (make-lock :name "KS Lock"))))
#<standard-unit-class ks-lock-mixin>
> (define-ks-class ks-with-lock (ks ks-lock-mixin)
    ())
#<standard-unit-class ks-with-lock>
>
```

define-ks-class

define-ksa-class *ksa-class-name* (*{superclass-name}**) (*{slot-specifier}**) *{class-option}** [*Macro*]
⇒ *new-ksa-class*

Purpose
Define or redefine a ksa class.

Package :agenda-shell

Module :agenda-shell

Arguments
ksa-class-name A non-nil, non-keyword symbol that names the ksa class

superclass-name A non-nil, non-keyword symbol that specifies a direct superclass of the ksa class *ksa-class-name*

slot-specifiers See below

class-options See below

new-ksa-class A new or modified ksa class object

Returns
The newly defined or modified ksa class object.

Errors
The specified *superclass-names* do not include at least one ksa class name. This error is signaled on class finalization.

Detailed syntax
slot-specifier ::= *slot-name* |
(*nonlink-slot-name* [[*nonlink-slot-option*]]) |
(*link-slot-name* [[*link-slot-option*]])

nonlink-slot-name ::= *slot-name*

link-slot-name ::= *slot-name*

link-slot-option ::= *slot-option* |
{:link *inverse-link-slot-specifier*} |
{:singular *boolean*} |
{:sort-function *function*} |
{:sort-key *function*}

inverse-link-slot-specifier ::= (*unit-class-name link-slot-name* [:singular *boolean*]) |
:reflexive

nonlink-slot-option ::= *slot-option* |
{:reader *reader-function-name*}* |
{:writer *writer-function-name*}*

slot-option ::= {:accessor *reader-function-name*}* |
{:allocation *allocation-type*} |
{:documentation *string*} |
{:initarg *initarg-name*}* |
{:initform *form*} |
{:type *type-specifier*}

class-option ::= (:abstract *boolean*) |
(:default-initargs . *initarg-list*) |
(:dimensional-values *dimension-value-specifier**) |
(:documentation *string*) |
(:estimated-instances *integer*) |
(:export-accessors *boolean*) |
(:export-class-name *boolean*) |
(:export-slot-names *direct-slots-specifier*) |
(:generate-accessors *direct-slots-specifier*) |
(:generate-accessors-format {:prefix | :suffix} |
(:generate-accessors-prefix {*string* | *symbol*}) |
(:generate-accessors-suffix {*string* | *symbol*}) |
(:generate-initargs *direct-slots-specifier*) |
(:initial-space-instances *initial-space-instance-specifier*) |
(:instance-name-comparison-test *instance-name-comparison-test*) |
(:metaclass *class-name*) |
(:retain {*boolean* | :propagate}) |
(:use-global-instance-name-counter *boolean*)
initial-space-instance-specifier ::= {*space-instance-path*+ | *function*}
dimension-value-specifier ::= *incomposite-dv-specifier* | *composite-dv-specifier*
incomposite-dv-specifier ::= (*dimension-name dimension-value-spec dimension-value-place*)
composite-dv-specifier ::= (*dimension-name dimension-value-specifier*
composite-type dimension-value-place)
composite-type ::= :set | :sequence |
{:ascending-series *ordering-dimension-name*} |
{:descending-series *ordering-dimension-name*}
dimension-value-specifier ::= *dimension-value-type* |
(*ordered-dimension-value-type* [*ordered-comparison-type*]) |
(*enumerated-dimension-value-type* [*enumerated-comparison-type*]) |
(*boolean-dimension-value-type* [*boolean-comparison-type*])
dimension-value-type ::= *ordered-dimension-value-type* |
enumerated-dimension-value-type |
boolean-dimension-value-type
ordered-dimension-value-type ::= :point | :interval | :mixed
enumerated-dimension-value-type ::= :element
boolean-dimension-value-type ::= :boolean
ordered-comparison-type ::= number | fixnum | short-float | single-float |
double-float | long-float |
pseudo-probability
enumerated-comparison-type ::= eq | eql | equal | equalp
boolean-comparison-type ::= t
dimension-value-place ::= {*slot-name* [*slot-name*]} | {*function* [*slot-name*]}
direct-slots-specifier ::= nil | t | *included-slot-name** |
{t :exclude *excluded-slot-name**}

The default *ordered-comparison-type*, if unspecified, is number. The default *enumerated-comparison-type*, if unspecified, is eql. The default *boolean-comparison-type* is t.

A *dimension-value-place* with two *slot-names* is allowed only for an :interval dimension-value specification.

Terms

class-name	A non-`nil`, non-keyword symbol that names a class
documentation	A documentation string
initarg-list	An initialization argument list
slot-name	A non-`nil`, non-keyword symbol
instance-name-comparison-test	One of the four standardized hash table test function names: `eq`, `eql`, `equal`, or `equalp` (default for classes of metaclass **standard-unit-class** is `eql`)

Description

A *dimension-value-place* with two *slot-names* can be specified only for `:interval` dimension-value types.

Each *superclass-name* argument specifies a direct superclass of the new class. If the superclass list is empty, then the direct superclass defaults to the single class **ksa**.

The `:metaclass` *class-name* class option, if specified, must be a subclass of **standard-ksa-class**. The default metaclass value is also **standard-ksa-class**.

Inheritance of class options

The set of *dimensional-values* for a unit class is the union of the sets specified in the *dimensional-values* options of the class and its superclasses. When more than one dimension-value specification is supplied for a given dimension, the one supplied by the most specific class is used.

The effective *initial-space-instances* value for a unit class is the value specified in the definition of the most specific unit class. (No additive inheritance of *initial-space-instances* is performed.) If no definitions specify an *initial-space-instances* value, `nil` is used.

The *instance-name-comparison-test* value is not inherited. If no value is specified in the unit-class definition, the default initialization value associated with the metaclass is used.

If a *retain* value is not specified, a value of `:propagate` is used as the default if any parent unit classes have a `:propagate` retention value; otherwise `nil` is used as the default value.

The *use-global-instance-name-counter* value is not inherited. If no value is specified in the unit-class definition, the default initialization value associated with the metaclass is used.

See also

define-ks (page 587)
delete-blackboard-repository (page 442)
standard-ksa-class (page 621)
with-generate-accessors-format (page 136)

Examples

Define a ksa class, `ksa-with-lock`, that has an additional slot containing a lock that can be used to synchronize operations on each KSA of that class.

```
> (define-ksa-class ksa-with-lock ()
    ((lock :initform (make-lock :name "KSA Lock"))))
#<standard-ksa-class ksa-with-lock>
>
```

Do the same, but with a mixin class:

```
> (define-unit-class ksa-lock-mixin ()
    ((lock :initform (make-lock :name "KSA Lock"))))
#<standard-unit-class ksa-lock-mixin>
> (define-ksa-class ksa-with-lock (ksa ksa-lock-mixin)
    ())
#<standard-ksa-class ksa-with-lock>
>
```

---

define-ksa-class

describe-ks *ks-name* [*Generic Function*]

Purpose
Print information about a knowledge source (KS).

Method signatures
describe-ks (*ks-name* `symbol`)

describe-ks (*ks* `ks`)

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
unit-class-name A unit-class or a symbol specifying a unit class.

Description
The description is printed to the ***standard-output*** stream.

See also
define-ks (page 587)
ks (page 606)

Example
```
> (describe-ks 'start-control-shell-ks)

KS: start-control-shell-ks
  Trigger events:        ((control-shell-started-event))
  Precondition function: #<Function scse-precondition>
  Execution function:    #<Function scse-fn>
>
```

ensure-ks *ks-name* `&key` *activation-predicate enabled execution-function ks-class ksa-class obviation-events obviation-predicate precondition-function rating retrigger-events retrigger-function revalidation-predicate trigger-events* ⇒ *ks* [*Function*]

Purpose

Programmatically define or redefine a knowledge source (KS).

Package

`:agenda-shell`

Module

`:agenda-shell`

Arguments

ks-name	A symbol naming the KS
activation-predicate	A function designator specifying a function object of two arguments (the KS unit instance and the event object) that returns a generalized boolean or `nil` (default is `nil`)
enabled	A generalized boolean (default is `t`)
execution-function	A function designator specifying a function object of one argument (the KSA unit instance) or `nil` (default is `nil`)
ks-class	A class or a symbol specifying a class
ksa-class	A class or a symbol specifying a class
obviation-events	An *event-specification* (see below)
obviation-predicate	A function designator specifying a function object of two arguments (the KSA unit instance and the event object) that returns a generalized boolean or `nil` (default is `nil`)
precondition-function	A function designator specifying a function object of two arguments (the KS unit instance and the event object) or `nil` (default is `nil`)
rating	A rating (default is `1`)
retrigger-events	An *event-specification* (see below)
retrigger-function	A function designator specifying a function object of two arguments (the KSA unit instance and the event object) or `nil` (default is `nil`)
revalidation-predicate	A function designator specifying a function object of one argument (the KSA unit instance) that returns a generalized boolean or `nil` (default is `nil`)
trigger-events	An *event-specification* (see below)
ks	A KS

Returns

The unit instance representing the KS

Detailed syntax

event-specification ::= `(`*event-signature**`)`

event-signature ::= `(`*event-class-specifier*
[*unit-class-or-instance-specifier*
[{`:slot-name` *slot-name*} | {`:slot-names` *slot-names*} |
{`:path` *path*} | {`:paths` *paths*}]])

event-class-specifier ::= *atomic-event-class* | `(`*atomic-event-class subeventing-specifier*`)` | `t`

atomic-event-class ::= *event-class* | *event-class-name*
subeventing-specifier ::= `:plus-subevents` | `:no-subevents` | `+` | `=`

The shorthand + subevents specifier is equivalent to `:plus-subevents` and = to `:no-subevents`.

unit-class-or-instance-specifier ::= *unit-instance* | (*unit-instance**) |
atomic-unit-class |
(*atomic-unit-class subclassing-specifier*) | `t`
atomic-unit-class ::= *unit-class* | *unit-class-name*
subclassing-specifier ::= `:plus-subclasses` | `:no-subclasses` | `+` | `=`

The shorthand + subclasses specifier is equivalent to `:plus-subclasses` and = to `:no-subclasses`.

Description

This function is called to define or redefine a KS. It is the functional equivalent of **define-ks** and is called by the expansion of the **define-ks** macro. (See the description of **define-ks** for details of KS definition and redefinition.)

See also

define-ks (page 587)
ks (page 606)
ks-enabled-p (page 607)
undefine-ks (page 627)

Example

Define an initial KS that is triggered when the control shell is started:

```
(ensure-ks 'initial
   :trigger-events '((control-shell-started-event))
   :execution-function #'initial-ks-function)
```

executed-ksas-of *control-shell* ⇒ *ksa-queue* [*Generic Reader*]

Purpose
Returns the executed KSAs queue of *control-shell*.

Method signatures
executed-ksas-of (*control-shell* `control-shell`) ⇒ *ksa-queue*

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
control-shell A `control-shell` object
ksa-queue A **ksa-queue** object

Returns
The executed KSAs queue object.

Description
This generic function accesses the **ksa-queue** stored in the `executed-ksas` link slot of *ksa*. This value is maintained by the Agenda Shell and should not be changed.

See also

clear-queue (page 520)
ksa (page 609)
ksa-queue (page 610)
make-queue (page 525)
obviated-ksas-of (page 611)
on-queue-p (page 529)
pending-ksas-of (page 614)
queue (page 532)
queue-element (page 533)
show-queue (page 536)

Example
Show the control shell's pending KSAs queue (early in a `:tutorial-example` run):

```
>  (show-queue (executed-ksas-of (current-control-shell)))
    0. #<ksa 2 startup-ks 100>
    1. #<ksa 3 random-walk-ks 100>
    2. #<ksa 5 random-walk-ks 100>
    3. #<ksa 6 random-walk-ks 100>
>
```

execution-cycle-of *ksa* ⇒ *cycle-number* or `nil` [*Generic Reader*]

Purpose
Returns the cycle number when a KSA was executed.

Method signatures
execution-cycle-of (*ksa* `ksa`) ⇒ *cycle-number*

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
ksa A KSA
cycle-number An integer or `nil`

Returns
The execution cycle number of *ksa* or `nil`, if *ksa* has not been executed

Description
This generic function accesses the value stored in the `execution-cycle` nonlink slot of *ksa*. This value is maintained by the Agenda Shell and should not be changed.

See also
ksa (page 609)

Example
Return the execution cycle of `ksa`:

```
> (execution-cycle-of ksa)
1237
>
```

exit-control-shell `&rest` *result-form** [*Function*]

Purpose
Exit the Agenda Shell.

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
result-form A form

Errors
Exit-control-shell called outside the context of an executing control shell.

See also

abort-ks-execution (page 582)
control-shell-running-p (page 585)
current-control-shell (page 586)
restart-control-shell (page 617)
start-control-shell (page 622)

Example
Exit the Agenda Shell, indicating that a solution, `solution`, was found:

```
(exit-control-shell ':solution-found solution)
```

find-ks-by-name *ks-name* ⇒ *ks* or `nil` [*Function*]

Purpose
Return a KS unit instance given its name.

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
ks-name A symbol naming the KS.
ks A KS

Returns
The KS unit instance named *ks-name* or `nil`, if none has been defined.

See also
define-ks (page 587)
ks (page 606)

Example
Return the KS named `start-control-shell-ks`:

```
> (find-ks-by-name 'start-control-shell-ks)
#<ks start-control-shell-ks>
>
```

ks [*Unit Class*]

Package

`:agenda-shell`

Module

`:agenda-shell`

Description

The class **ks** is the default class of instances created by **define-ks**.

See also

define-ks (page 587)
ksa (page 609)

ks-enabled-p *ks* ⇒ *boolean* [*Generic Function*]

Purpose
Determine if the specified KS is enabled for execution.

Setf syntax
(setf (ks-enabled-p *ks*) *boolean*) ⇒ *boolean*

Method signatures
ks-enabled-p (*ks* `ks`) ⇒ *boolean*
(setf ks-enabled-p) *boolean* (*ks* `ks`) ⇒ *boolean*

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
ks A KS
boolean A generalized boolean

Returns
True if the KS is enabled for execution; `nil` otherwise.

Description
This generic function accesses the value stored in the `enabled` nonlink slot of *ks*.

See also
define-ks (page 587)
ks (page 606)
undefine-ks (page 627)

Examples
See if KS *ks* is enabled for execution:

```
> (ks-enabled-p ks)
t
>
```

Now disable KS `ks`:

```
(setf (ks-enabled-p ks) nil)
```

Check once again:

```
> (ks-enabled-p ks)
nil
>
```

ks-of *ksa* ⇒ *ks* [*Generic Reader*]

Purpose
Returns the knowledge source (KS) unit instance of a KSA.

Method signatures
ks-of (*ksa* `ksa`) ⇒ *ks*

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
ksa A KSA
ks A KS

Returns
The KS unit instance of *ksa*

Description
This generic function accesses the value stored in the `ks` link slot of *ksa*. This value is maintained by the Agenda Shell and should not be changed.

See also
ks (page 606)
ksa (page 609)

Example
Return the KS of a KSA:

```
> (ks-of ksa)
#<ks start-control-shell-ks>
>
```

ksa [*Unit Class*]

Package

`:agenda-shell`

Module

`:agenda-shell`

Description

The class **ksa** is the default class of unit instances representing KS activations.

See also

ks (page 606)

ksa-queue [*Unit Class*]

Package :agenda-shell

Module :agenda-shell

Description

The unit class whose instances are used as the header of KSA queues.

See also

clear-queue (page 520)
executed-ksas-of (page 602)
ksa (page 609)
make-queue (page 525)
obviated-ksas-of (page 611)
on-queue-p (page 529)
ordered-ksa-queue (page 613)
ordered-queue (page 530)
queue (page 532)
queue-element (page 533)
show-queue (page 536)

obviated-ksas-of *control-shell* ⇒ *ksa-queue* [*Generic Reader*]

Purpose
Returns the obviated KSAs queue of *control-shell*.

Method signatures
obviated-ksas-of (*control-shell* `control-shell`) ⇒ *ksa-queue*

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
control-shell A `control-shell` object
ksa-queue A **ksa-queue** object

Returns
The obviated KSAs queue object.

Description
This generic function accesses the **ksa-queue** stored in the `obviated-ksas` link slot of *ksa*. This value is maintained by the Agenda Shell and should not be changed.

See also

clear-queue (page 520)
executed-ksas-of (page 602)
ksa (page 609)
ksa-queue (page 610)
make-queue (page 525)
on-queue-p (page 529)
pending-ksas-of (page 614)
queue (page 532)
queue-element (page 533)
show-queue (page 536)

Example
Show the control shell's obviated KSAs queue (early in a `:tutorial-example` run):

```
> (show-queue (obviated-ksas-of (current-control-shell)))
The queue is empty.
>
```

obviation-cycle-of *ksa* ⇒ *cycle-number* or `nil` [*Generic Reader*]

Purpose
Returns the cycle number when a KSA was obviated.

Method signatures
obviation-cycle-of (*ksa* `ksa`) ⇒ *cycle-number*

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
ksa	A KSA
cycle-number	An integer or `nil`

Returns
The obviation cycle number of *ksa* or `nil`, if *ksa* has not been obviated

Description
This generic function accesses the value stored in the `obviation-cycle` nonlink slot of *ksa*. This value is maintained by the Agenda Shell and should not be changed.

See also
ksa (page 609)

Example
Return the obviation cycle of `ksa`:

```
> (obviation-cycle-of ksa)
1211
>
```

ordered-ksa-queue [*Unit Class*]

Package

`:agenda-shell`

Module

`:agenda-shell`

Description

The unit class whose instances are used as the header of ordered (sorted) KSA queues.

See also

clear-queue (page 520)
ksa (page 609)
ksa-queue (page 610)
make-queue (page 525)
on-queue-p (page 529)
ordered-queue (page 530)
pending-ksas-of (page 614)
queue (page 532)
queue-element (page 533)
show-queue (page 536)

pending-ksas-of *control-shell* ⇒ *ordered-ksa-queue* [*Generic Reader*]

Purpose
Returns the pending KSAs queue of *control-shell*.

Method signatures
pending-ksas-of (*control-shell* `control-shell`) ⇒ *ordered-ksa-queue*

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
control-shell A `control-shell` object
ordered-ksa-queue An **ordered-ksa-queue** object

Returns
The pending KSAs queue object.

Description
This generic function accesses the **ordered-ksa-queue** stored in the `pending-ksas` link slot of *ksa*. This value is maintained by the Agenda Shell and should not be changed.

See also

clear-queue (page 520)
executed-ksas-of (page 602)
ksa (page 609)
make-queue (page 525)
obviated-ksas-of (page 611)
on-queue-p (page 529)
ordered-ksa-queue (page 613)
ordered-queue (page 530)
queue-element (page 533)
show-queue (page 536)

Example
Show the control shell's pending KSAs queue (early in a `:tutorial-example` run):

```
>  (show-queue (pending-ksas-of (current-control-shell)))
    0. #<ksa 7 random-walk-ks 100>
    1. #<ksa 1 count-center-locations-ks 90>
    2. #<ksa 4 print-walk-ks 80>(show-queue pending-ksas :end 5)
>
```

rating [*Type*]

Package

`:agenda-shell`

Module

`:agenda-shell`

Description

An integer between `most-negative-rating` (-32768) and `most-positive-rating` (32767), inclusive. Ratings are used by the Agenda Shell to order pending KSAs.

rating-of *ksa* ⇒ *rating* [*Generic Accessor*]

Purpose
Accesses the rating of a KSA.

Setf syntax
(setf (rating-of *ksa*) *rating*) ⇒ *rating*

Method signatures
rating-of (*ksa* `ksa`) ⇒ *rating*
(setf rating-of) *rating* (*ksa* `ksa`) ⇒ *rating*

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
ksa A KSA
rating A rating

Returns
The rating of *ksa*

Description
This generic function accesses the value stored in the `rating` nonlink slot of *ksa*. This value is used by the Agenda Shell to determine when to execute the KSA.

See also
define-ks (page 587)
ks (page 606)
ksa (page 609)

Examples
Return the rating of a KSA:

```
> (rating-of ksa)
58
> (setf (rating-of ksa) 80)
80
> (rating-of ksa)
80
>
```

Note
The rating of a pending KSA can be changed by using **setf** or related macros with this accessor.

restart-control-shell `&key` *instance-name* ⇒ *result** [*Function*]

Purpose
Restart the agenda shell named *instance-name*.

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
instance-name An object naming a control-shell instance (default is `1`)
results (see below)

Returns
One of the following sets of multiple values:

- `:quiescence`—If the control-shell scheduling loop is terminated due to quiescence (that is, no more executable KSAs remain in the queue of pending KSAs)
- `:stop` and (optionally) associated reasons, as multiple values—If one of the following conditions occurs:
  - The **exit-control-shell** function is called.
  - A precondition function or KS-execution function returns `:stop` and, optionally, associated reasons.
- *Result-values*—If the control-shell is terminated by calling **exit-control-shell**.

Events
A `control-shell-restarted-event` is signaled.

Errors
The idle-loop process has not been started on CMUCL. Multiprocessing has not been started on LispWorks.

See also

abort-ks-execution (page 582)
control-shell-running-p (page 585)
current-control-shell (page 586)
exit-control-shell (page 604)
run-polling-functions (page 303)
start-control-shell (page 622)

Example
Restart the Agenda Shell (in this case, without any KSs defined):

```
> (restart-control-shell)
;; Control shell 1 restarting after cycle 2
;; No executable KSAs remain, exiting control shell
;; Control shell 1 exited: 4 cycles completed
;; Run time: 0 seconds
```

```
;; Elapsed time: 0 seconds
:quiescence
>
```

Note

When a non-`nil` `:run-polling-functions` value is supplied to **start-control-shell** (the default on Common Lisp implementations without threads), **run-polling-functions** is called at the beginning of every control-shell-cycle and at one-half-second intervals when the Agenda Shell is hibernating due to quiescence.

---

restart-control-shell

sole-trigger-event-of *ksa* ⇒ *event* or `nil` [*Generic Function*]

Purpose
Return the sole trigger event of a KSA.

Method signatures
sole-trigger-event-of `(`*ksa* `ksa)` ⇒ *event* or `nil`

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
ksa A KSA
event An event object or `nil`

Returns
The trigger event or `nil`, if one was not found for *ksa*

Description
If more than one trigger event is found for *ksa*, an error is signaled.

See also
sole-trigger-instance-of (page 620)

Example
Return the (sole) trigger event of a KSA:

```
> (sole-trigger-event-of ksa)
#<instance-created-event hyp>
>
```

sole-trigger-instance-of *source* ⇒ *trigger-instance* or `nil` [*Generic Function*]

Purpose
Return the trigger unit instance of a KSA, event, or a list of KSAs or events.

Method signatures
sole-trigger-instance-of (*cons* `cons`) ⇒ *trigger-instance* or `nil`
sole-trigger-instance-of (*event* `single-instance-event`) ⇒ *trigger-instance* or `nil`
sole-trigger-instance-of (*ksa* `ksa`) ⇒ *trigger-instance* or `nil`
sole-trigger-instance-of (*event* `multiple-instance-event`) ⇒ *trigger-instance* or `nil`
sole-trigger-instance-of (*event* `non-instance-event`) ⇒ `nil`

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
source	A KSA, event, or a list of KSAs or events
trigger-instance	A unit instance or `nil`

Returns
The trigger unit instance or `nil`, if one was not found in *source*

Description
Typically, **sole-trigger-instance-of** is called with a single KSA or single-instance event. If more than one trigger unit instance is found in *source*, an error is signaled.

See also
collect-trigger-instances (page 584)
sole-trigger-event-of (page 619)

Example
Return the (sole) trigger unit instance of a KSA:

```
> (sole-trigger-instance-of ksa)
#<hyp 419 (1835 4791) 0.85 [5..35]>
>
```

standard-ksa-class [*Unit Class*]

Package

`:agenda-shell`

Module

`:agenda-shell`

Description

The class **standard-ksa-class** is the default class of ksa classes defined by **define-ksa-class**. It is a subclass of **standard-unit-class**.

See also

define-ksa-class (page 595)
print-instance-slots (page 107)
standard-unit-class (page 369)

start-control-shell `&rest` *initargs* ⇒ *result** [*Function*]

Purpose
Start the Agenda Shell.

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
initargs An initialization argument list (see below)
results (See below)

Returns
One of the following sets of multiple values:

- `:quiescence`—If the control-shell scheduling loop is terminated due to quiescence (that is, no more executable KSAs remain in the queue of pending KSAs)
- `:stop` and (optionally) associated reasons, as multiple values—If one of the following conditions occurs:
  - The **exit-control-shell** function is called.
  - A precondition function or KS-execution function returns `:stop` and, optionally, associated reasons.
- *Result-values*—If the control-shell is terminated by calling **exit-control-shell**.

Events
A `control-shell-started-event` is signaled.

Errors
The idle-loop process has not been started on CMUCL. Multiprocessing has not been started on LispWorks.

Detailed syntax
Available *initargs* are:

awaken-on-event	A generalized boolean (default is `t`)
continue-past-quiescence	A generalized boolean (default is `nil`)
fifo-queue-ordering	A generalized boolean (default is `t`)
hibernate-on-quiescence	A generalized boolean (default is `nil`)
minimum-ksa-execution-rating	A rating (default is `1`)
instance-name	An object naming this control-shell instance (default is `1`)
output-stream	Control-shell output stream (default is `*trace-output*`)
pause	A generalized boolean (default is `nil`)
print	A generalized boolean (default is `t`)
run-polling-functions	A generalized boolean (default is `t` on non-threaded Common Lisp implementations; `nil` otherwise)
save-executed-ksas	A generalized boolean (default is `nil`)
save-obviated-ksas	A generalized boolean (default is `nil`)

stepping	Control-shell stepping options (default is `nil`)
stepping-stream	Control-shell stepping stream (default is `*query-io*`)

Description

Many Agenda Shell behaviors can be customized by providing non-default values for the following *initargs*:

`:awaken-on-event`	A generalized boolean value indicating whether the control shell is to be awakened from hibernation when any event is signaled
`:continue-past-quiescence`	A generalized boolean value indicating whether the control shell loop should continue even when quiescence-event processing has failed to produce any executable KSAs; use with caution, as the control shell will only exit by an explicit call to **exit-control-shell**
`:fifo-queue-ordering`	A generalized boolean value that indicates a newly rated pending KSA is to be placed ahead of equally rated KSAs (first-in, first out) or after them (last-in, first out)
`:hibernate-on-quiescence`	A generalized boolean value that determines whether the control-shell will hibernate rather than exit when no executable KSA exists; this decision point is never reached when `:continue-past-quiescence` is true
`:minimum-ksa-execution-rating`	The minimum rating value that a pending KSA must have to be executed
`:output-stream`	The stream to be used for control shell output
`:pause`	A generalized boolean that determines whether the control shell should hibernate until awakened at the start of each cycle
`:print`	A generalized boolean that determines whether start, restart, and termination messages are printed by the control shell
`:run-polling-functions`	A generalized boolean that determines whether polling functions (provided by the `:polling-functions` module (see page 298)) are to be run at the start of each control-shell cycle
`:save-executed-ksas`	A generalized boolean that determines whether executed KSA instances are to be saved on the executed-ksas queue
`:save-obviated-ksas`	A generalized boolean that determines whether obviated KSA instances are to be saved on the obviated-ksas queue
`:stepping`	A list of stepping options (see below) indicating the kinds of control-shell stepping that are to be enabled initially, or the symbol `t`, indicating all stepping options are enabled
`:stepping-stream`	The stream to be used for control shell stepping

Stepping options

The supported stepping options and their interpretations are as follows:

`:activation-predicate`	about to execute the activation predicate of a KS
`:ks-activation`	about to create a KS activation
`:ksa-execution`	about to execute a KS activation
`:obviation-predicate`	about to execute the obviation predicate of a KS
`:precondition-function`	about to execute the precondition function of a KS
`:process-event`	about to perform control-shell processing associated with an event

:quiescence	about to perform activities triggered by control-shell quiescence
:retrigger-function	about to execute the retrigger function of a KS
:retrigger-function	about to execute the revalidation predicate of a KS

See also

control-shell-running-p (page 585)
current-control-shell (page 586)
exit-control-shell (page 604)
run-polling-functions (page 303)
restart-control-shell (page 617)

Examples

Start the Agenda Shell (in this case, without any KSs defined):

```
> (start-control-shell)
;; Control shell 1 started
;; No executable KSAs remain, exiting control shell
;; Control shell 1 exited: 2 cycles completed
;; Run time: 0 seconds
;; Elapsed time: 0 seconds
:quiescence
>
```

Start the Agenda Shell (again without any KSs defined, but with stepping enabled):

```
> (start-control-shell :stepping 't)
;; Control shell 1 started
>> CS Step (cycle 1):
   About to signal quiescence... [?  entered]
Stepping commands (follow with <Return>):
   d       Disable this kind of stepping (:quiescence)
   e       Enable another kind of stepping
   f       Evaluate a form
   h or ?  Help (this text)
   q       Quit (disable all stepping and continue)
   s       Show enabled stepping kinds
   x       Exit control shell
   =       Describe the object of interest (bound to ==)
   +       Enable all stepping
   -       Disable all stepping
   <Space> Continue (resume processing)
>> CS Step (cycle 1):
   About to signal quiescence... [d entered]
:quiescence stepping disabled
>> CS Step (cycle 1):
   About to signal quiescence... [q entered]
All stepping disabled
;; No executable KSAs remain, exiting control shell
;; Control shell 1 exited: 2 cycles completed
;; Run time: 0 seconds
;; Elapsed time: 54 seconds
:quiescence
>
```

Note

When a non-`nil` `:run-polling-functions` value is supplied to **start-control-shell** (the default on Common Lisp implementations without threads), **run-polling-functions** is called at the beginning of every control-shell-cycle and at one-half-second intervals when the Agenda Shell is hibernating due to quiescence.

---

start-control-shell

trigger-events-of *ksa* ⇒ *events* [*Generic Reader*]

Purpose
Returns the list of events that triggered a KSA

Method signatures
trigger-events-of `(`*ksa* `ksa)` ⇒ *events*

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
ksa A KSA
events A proper list

Returns
The list of events that triggered *ksa*

Description
This generic function accesses the value stored in the `trigger-events` link slot of *ksa*.

See also
define-ks (page 587)
ks (page 606)
ksa (page 609)

Example
Return the events that triggered a KSA:

```
> (trigger-events-of ksa)
(#<instance-created-event #<hyp 233 (1835 4791) 0.89 [5..35]>)
>
```

undefine-ks *ks-name* `&rest` *ignored-initargs* ⇒ *deleted-ks-unit-instance* [*Macro*]

Purpose
Undefine (delete) a knowledge source (KS).

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
ks-name	A symbol naming the KS (not evaluated, but the remaining arguments are evaluated)
ignored-initargs	The remaining initialization arguments are ignored
deleted-ks-unit-instance	A KS unit instance or `nil`

Returns
The deleted KS unit instance, if KS *ks-name* was undefined (deleted).

Description
A KS is undefined by deleting the unit instance corresponding to the KS. The **undefine-ks** macro provides a convenient shortcut for undefining a KS by minimally editing the defining form (such as from within an editor buffer).

See also
define-ks (page 587)
ks-enabled-p (page 607)

Examples
Undefine the KS named `initial`. The following forms are all equivalent:

```
> (undefine-ks initial
     :trigger-events '((control-shell-started-event))
     :execution-function #'initial-ks-function)
#<deleted-unit-instance ks initial>
> (undefine-ks initial)
#<deleted-unit-instance ks initial>
> (delete-instance
     (find-instance-by-name 'initial '(ks :plus-subclasses)))
#<deleted-unit-instance ks initial>
> (delete-instance
     (find-instance-by-name 'initial '(ks +)))
#<deleted-unit-instance ks initial>
>
```

Glossary

alist
An association list.

association list
A list of conses representing an association of keys with values. The car of each cons is the key and the cdr is the value associated with that key.

atomic operation
A computation that, once started, is completed without being interrupted by another thread.

autorun forms
Forms in a module file that are to be evaluated conditionally when the file is loaded based on the value of ***autorun-modules***. Part of the Module Manager Facility (see page 21).

blackboard repository
The internal storage containing all unit instance and space instance objects and associated retrieval data structures.

boolean dimension
A dimension of `:boolean` dimension type where `:boolean` dimension values are either true (non-`nil`) or false (`nil`).

broadcast streamer
An object that is used as a saving or sending destination for multiple journal and network streamers.

circular list
A list that has no termination because it includes an earlier portion of itself in its successive sublists.

class
An object that uniquely (directly or indirectly) determines the structure and behavior of a set of other objects. Members of this set are called instances of the class.

class designator
A class or a symbol that names a class.

class option
An option that refers to a class as a whole or to all the slots of the class.

comparison type
A symbol that indicates the way that dimension values are compared. For an `:ordered` dimension value, one of: `number`, `fixnum`, `short-float`, `single-float`, `double-float`, or `long-float`. For an `:enumerated` dimension value, one of: `eq`, `eql`, `equal`, or `equalp`. For a `:boolean` dimension value: `t`.

composite dimension value
A dimension value that is a set, sequence, or series of dimension values.

condition variable
A condition variable provides an atomic means for a thread to release a lock (or recursive lock) that it holds and go to sleep until it is awakened by another thread. Once awakened, the lock that it was holding is reacquired atomically before the awakened thread is allowed to do anything else. A Portable Threads condition-variable object includes the lock that is associated with the condition variable, and the condition-variable object can be used directly as a lock.

cons
An object with two components called the **car** and the **cdr**. Conses are used to construct lists.

dimension

A conceptual extent within which values that share some relationship can be placed. GBBopen uses dimensionality to relate the extent representations of unit instances, space instances, and retrieval patterns.

GBBopen supports three dimension types:

1. ordered dimensions (`:ordered`)
2. enumerated dimensions (`:enumerated`)
3. boolean dimensions (`:boolean`)

Real-world dimensions (such as time and location) can be represented as ordered dimensions.

dimension name
A symbol used to identify a dimension. In general, two dimensions with different dimension types should not be given the same dimension name.

dimension type
The interpretation associated with a dimension of a unit class, a space instance, or a retrieval pattern; one of `:ordered`, `:enumerated`, or `:boolean`.

dimension value
The value that can be used to position a unit instance on a dimension in one or more space instances.

dimension-value type
The interpretation associated with a dimension value of a unit instance; one of `:point`, `:interval`, `:mixed` (both points and intervals), `:element`, or `:boolean`. The dimension-value types `:point`, `:interval`, and `:mixed` indicate values in an ordered dimension, `:element` indicates a value in an enumerated dimension, and `:boolean` indicates a value in a boolean dimension.

dimensional extent
The dimensions of a space instance.

dotted list
A list that is terminated by a non-`nil` atom rather than the empty list, `nil`.

enumerated dimension
A dimension of `:enumerated` dimension type where `:element` dimension values are individual elements from an extensible set of discrete elements.

ESET
A keys-only table that automatically transitions between list and hash-table implementations. The keys are compared using `eq`.

ET
A key-and-value table that automatically transitions between list and hash-table implementations. The keys are compared using `eq`.

event
An activity that is noticed, and signaled, by GBBopen.

event class
An object that is a subclass of **standard-event-class**.

event function
A function that is associated with one or more event specifications and is called whenever such an event occurs. (See **signal-event** for the required event-function arguments for each event metaclass.

event instance
An object whose class is a subclass of **standard-event-instance**.

event metaclass
One of five "types" of event. Every event class has one of the following event metaclasses: `non-instance-event-class`, `instance-event-class`, `space-instance-event-class`, `nonlink-slot-event-class`, or `link-slot-event-class`.

executable KS activation
A pending KS activation that meets the criteria for execution, such as having a rating above the minimum KSA execution rating in effect for the control shell.

executed KS activation
A KS activation that has completed execution and will, therefore, not be operated on again by the control shell.

extended event-class specification
A specification of one or more event classes as indicated by one of the following:

- a event class
- a symbol naming a event class
- a list containing one of the above followed by the keyword `:plus-subevents` or the keyword `:no-subevents`
- the symbol `t`, which is equivalent to `(standard-event-instance :plus-subevents)`

extended unit-class specification
A specification of one or more unit classes as indicated by one of the following:

- a unit class
- a symbol naming a unit class
- a list containing one of the above followed by the keyword `:plus-subclasses` or the keyword `:no-subclasses`
- the symbol `t`, which is equivalent to `(standard-unit-instance :plus-subclasses)`

extended unit-class or instance specification
A specification of one or more unit instances or one or more unit classes as indicated by one of the following:

- a unit class
- a symbol naming a unit class
- a list containing one of the above followed by the keyword `:plus-subclasses` or the keyword `:no-subclasses`
- the symbol `t`, which is equivalent to `(standard-unit-instance :plus-subclasses)`
- a unit instance
- a list of unit instances

extended unit-classes specification
A specification of one or more unit classes as indicated by one of the following:

- a unit class
- a symbol naming a unit class
- a list containing one of the above followed by the keyword `:plus-subclasses` or the keyword `:no-subclasses`
- a list of one or more of the above
- the symbol `t`, which is equivalent to `(standard-unit-instance :plus-subclasses)`

feature
A symbol in the list value of the variable ***features***. The features in this features list are used to control conditional compilation and implementation-specific behaviors.

form
An object (including an expression) to be evaluated.

function designator
: An object that specifies a function. Either: a symbol (denoting the function named by that symbol in the global environment), or a function object (denoting itself). The term “function” is often used to denote a function designator, with the term “function object” used when referring specifically to a function object.

function object
: An object of type `function`. The term “function” is often used to denote a function designator, with the term “function object” used when referring specifically to a function object.

fixnum
: An integer between `most-negative-fixnum` and `most-positive-fixnum` inclusive.

generalized boolean
: An object used as a truth value, where `nil` represents false and all other objects represent true.

generalized reference
: A reference to a location storing a value as if to a variable.

generic function
: A function whose behavior depends on the classes or identities of the arguments supplied to it.

incomplete unit instance
: A forward-referenced unit instance created during the loading of a blackboard repository or journal file or the reading of a network stream. An incomplete unit instance does not reside on any space instance or have all of its saved/sent slot values present until it becomes "complete" by loading or receiving a form containing the instance’s full details.

incomposite dimension value
: A dimension value that is a single point, interval, element, or boolean (i.e., not a composite dimension value).

initialization argument list
: A list of alternating names and values used to initialize or reinitialize instances of classes. If more than one name and value pair has the same name, only the first such pair is used to provide the value.

instance
: An object whose structure and behavior is uniquely (directly or indirectly) determined by a class object.

instance name
: An object that uniquely identifies an instance of a unit class. The same object can be used to identify instances of different unit classes, but the same instance name cannot be used with two instances of the same class.

interval
: A cons, two-element list, or two-element array containing the start and end value representing the set of real numbers between them, inclusive.

journal
: A file containing changes made to the blackboard repository that can be loaded in order to perform the recorded changes.

journal streamer
: An object that is used as a saving destination for journaling functions.

journaling
: Writing changes made to the blackboard repository to a file that can be loaded in order to perform the recorded changes. Journaling can be used with or without repository saving to recreate a blackboard repository.

keyword symbol
A symbol whose home package is the `keyword` package.

knowledge source
The expertise associated with a collaborating computational entity in a blackboard application (often abbreviated as "KS"). More specifically, a KS is an object containing the expertise and other information associated with a computational entity. A `ks` object is also a unit instance, but KSs are normally described by their more specific categorization.

KS activation
The application of a KS to a specific computational context (often abbreviated as "KSA"). More specifically, a KSA is a ksa-class object representing the KS activation. A `ksa` is also a unit instance, but they are normally described by their more specific categorization.

KS execution
The execution of a KS activation.

ks class
An object that is a subclass of **standard-unit-class** that is used to represent a KS.

ksa class
An object that is a subclass of **standard-ksa-class** that is used to represent a KSA.

left-leaning red-black tree
An LLRB tree.

LLRB tree
A binary search tree that is roughly balanced, keeping the worst-case time for operations such as inserting, deleting, and finding values proportional to the height of the tree. Left-leaning red-black (LLRB) trees are a simpler version of red-black trees introduced by Sedgewick in 2008 where all red links must lean left except during inserts and deletes.

link
A bi-directional relationship between two unit instances represented by a pair of pointers, one at each unit instance pointing to the other unit instance. GBBopen's link operators maintain the bi-directional consistency of link pointers.

link slot
A slot designated for the outgoing pointers of links associated with that slot.

link-pointer object
An object that can be used as a pointer in a link slot. A link-pointer object must have a **link-instance-of** method defined for it whose result returns the unit instance to be used as the link pointer.

link-slot place
A form which is suitable for use as a generalized reference to a link slot. Typical examples of link-slot-place forms include:

```
(slot-accessor unit-instance)
(slot-value unit-instance slot-name)
```

where:

slot-accessor is a symbol specifying an accessor function for a link slot
unit-instance is a unit instance
slot-name is a symbol naming a link slot in *unit-instance*

lock
A mutual-exclusion object that allows multiple threads to synchronize activities or access to shared resources. A lock has two states, unlocked or locked by a specific thread. Once a lock is held by a thread, any other threads attempting to lock it will block. When the lock-holding

thread unlocks (releases) the lock, one of the blocked threads will acquire (lock) it and proceed. If the thread that is holding the lock attempts to re-acquire it, an error is signaled (see recursive lock).

metaobject
An instance of a metaobject class.

metaobject class
A class object that is a subclass of exactly one of the following classes: `class`, `slot-definition`, `generic-function`, `method`, and `method-combination`.

module
A set of related files that form a component, library, or application. Part of the Module Manager Facility (see page 21).

namestring
A string that represents a filename.

network streamer
An object that is used as a sending destination for network streaming functions.

network streaming
Sending unit instances (and space instances, changes made to the blackboard repository, and commands to a streamer node (typically in another Common Lisp image).

non-keyword symbol
A symbol whose home package is not the `keyword` package.

obviated KS activation
An unexecuted KS activation that has been deemed unnecessary and will therefore never be executed.

Offset Universal Time
A non-negative integer number of seconds measured from the beginning of a time base later than the year 1900 (ignoring leap seconds). Offset Universal Time is Universal Time that is offset by an integer time-base value so that the most often used Offset Universal Time values in an application are fixnums.

ordered dimension
A dimension of `:ordered` dimension type where `:point`, `:interval`, and `:mixed` dimension values are points or intervals on a continuous, real-number extent.

ordering dimension
The dimension whose dimension values are used to order a series-composite dimension value.

ordered queue
A doubly linked, ordered queue. A GBBopen queue is headed by an object that is a subclass of **ordered-queue**.

package designator
A string designator (denoting a string that designates the name or nickname of a package) or a package (denoting itself).

passive socket
A socket that is used to accept a connection initiation to a specific service port.

patch
A modification to the existing code of an application that is loaded after the regular application code, either at startup or into an executing application.

path expression
A regular expression representing one or more space-instance paths.

pathname
A structured representation of the name of a file. A pathname has six components: host, device, directory, name, type, and version.

pathname designator
A namestring (denoting the corresponding pathname), a stream associated with a file (denoting the pathname used to open the file), or a pathname (denoting itself).

pending KS activation
A KS activation that has not been executed or obviated.

periodic function
A function of no arguments that is run repeatedly at a specified interval, at a resolution as brief as supported by `sleep`. A separate thread is spawned to manage the periodic invocations of the specified function.

A count can also be provided for the periodic function. When specified, this value is decremented prior to each invocation of the function and, when it is no longer positive, the periodic-function thread is terminated.

predicate function
A function that returns a generalized-boolean value.

proper list
A list terminated by the empty list. (The empty list is a proper list.)

property
(of a property list) 1. A pair of elements in a property list representing the name of a property and its associated value. 2. The value of a property.

property list
A list containing an even number of elements that represent alternating names (sometimes called indicators or keys) and their associated values.

pseudo-probability
A discretized `fixnum` representation for probability values that maps probability values in the range [0.0..1.0] to integers in the range [0..1000].

pseudo-probability-ln
A discretized `fixnum` representation for the natural logarithm of a pseudo-probability value. Pseudo-probability-ln values range from [-6907756..0].

queue
A doubly linked queue. A GBBopen queue is headed by an object that is a subclass of **queue**. GBBopen queues that maintain a sorted ordering of queue elements are provided by ordered queues.

queue element
An object that is a subclass of **queue-element**.

quiescence
A control-shell state when no more executable KSAs are in the queue of pending KSAs.

rating
An integer between -32768 and 32767 inclusive, used by the Agenda Shell to order pending KSAs (see **rating**).

recursive lock
A mutual-exclusion object that allows multiple threads to synchronize activities or access to shared resources. A recursive lock has two states, unlocked or locked by a specific thread. Once a recursive lock is held by a thread, any other threads attempting to lock it will block. When the lock-holding thread unlocks (releases) the recursive lock, one of the blocked threads will acquire (lock) it and proceed. If the thread that is holding the recursive lock attempts to re-acquire it,

that thread is allowed to proceed as if it had acquired the lock (without error or blocking, see lock).

relative directory
A directory defined in relation to another directory definition. Part of the Module Manager Facility (see page 21).

REPL command
A keyword command that can be entered at the top level read-eval-print loop (REPL) in your Common Lisp environment. REPL commands (some with arguments) provide convenient shortcuts for often-used operations.

retrieval pattern
A list argument to **filter-instances**, **find-instances**, and **map-instances-on-space-instances** specifying the dimension value requirements for selecting unit instances to be returned.

required module
A sequence of modules that must be compiled (if needed) and loaded, in order, before the requiring module is compiled (if needed) and loaded. Part of the Module Manager Facility (see page 21).

root directory
A fixed anchor directory for a tree of relative directory definitions. Part of the Module Manager Facility (see page 21).

scheduled function
An object that contains a function that may be scheduled to run at an absolute or relative time. When that specified time arrives, the function is invoked with a single argument: the scheduled-function object.

A repeat interval can also be specified for the scheduled function. When specified, this value is used whenever the function is invoked to schedule the function again at a new time relative to the current invocation.

Scheduled functions can be scheduled to a resolution of one second. Periodic function invocations at brief time intervals are provided by periodic functions.

series-composite dimension value
A dimension value that is a series of dimension values that are ordered by the dimension values of another dimension (the series-composite ordering dimension).

series-composite ordering dimension
The dimension whose values are used to order a series-composite dimension value.

sequence-composite dimension value
A dimension value that is a sequence of dimension values.

set-composite dimension value
A dimension value that is a set of dimension values.

slot
A component of an object that can store a value.

space
Pertaining to a space class or space instance.

space class
An object that is a subclass of **standard-space-class**.

space instance
An object whose class is a subclass of **standard-space-instance**. A space instance is also a unit instance, but space instances are normally described by their more specific categorization.

space-instance path
A complete list of space-instance names, starting with the most distant indirect parent space-instance name, that uniquely identifies a space instance in the blackboard repository.

standard-gbbopen-instance
An object whose class is a subclass of **standard-gbbopen-instance**. It is a superclass of **standard-event-instance** and **standard-unit-instance**.

storage specification
A specification of how unit instances are to be stored on a space instance.

streamer
An object that is used as a saving or sending destination for journaling or network-streaming functions.

streamer node
A logical "host" for sending and receiving streamed updates to unit instances and to the blackboard repository. Each streaming node is resolved uniquely by *host* and *port* and should be defined with a unique *name*. Although multiple streaming nodes can be defined for the same Common Lisp image, typically only one streaming node is defined per image.

streamer queue
A thread local queue of changes associated with a streamer that are being held until in memory until the queue is written or cleared.

string designator
An object that denotes a string. One of: a character (denoting a singleton string that has the character as its only element), a symbol (denoting the string that is its name), or a string (denoting itself).

subclasses
The classes that inherit from a class.

subevents
The classes that inherit from an event class.

system name
A keyword associating a set of REPL commands, directory definitions, and module definitions with a specific library or application.

thread
A thread in a multi-threaded Common Lisp implementation or a Lisp process in a Common Lisp that provides multiprocessing.

thread-local binding
A dynamic binding that is visible only to a single thread.

three-way comparison test
A function of two arguments that returns: a negative fixnum if the first argument is less than the second argument, zero if the two arguments are equal, and a positive fixnum if the first argument is greater than the second argument.

time zone
A rational number between -24 (inclusive) and 24 (inclusive) that represents a time zone as a number of hours offset from Greenwich Mean Time. A non-integral time zone must be a multiple of $\frac{1}{3600}$.

unit
Pertaining to a unit class or unit instance.

unit class
An object that is a subclass of **standard-unit-class**.

unit instance
: An object whose class is a subclass of **standard-unit-instance**. A space instance is also a unit instance, but space instances are normally described by their more specific categorization.

Universal Time
: A non-negative integer number of seconds measured from the beginning of the year 1900 (ignoring leap seconds).

variable symbol
: A symbol that can accept a binding.

Index

Page references are shown in **bold** when they refer to the definition or main source of information on the entry. A page reference that is given in *green italics* indicates an instructive example of the use of that entity.