GBBopen

GBBopen Reference Manual

Version 0.9.6

Dan Corkill

The GBBopen Project
http://GBBopen.org

August 27, 2007
21:51 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.

Zack Rubinstein and Susan Lander continue to demand additional capabilities and contribute suggestions. Other questioners and bug reporters include: Clayton Morrison, 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. BBTech Corporation provided hardware and Internet resources.

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 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, 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 11) that supports compiling and loading GBBopen components. This mini-module facility is designed for ease of use and for simplicity in porting to Common Lisp implementations. For example, to compile all core GBBopen modules and execute a basic trip-tests file, you only need to evaluate the following forms within your Common Lisp environment:

```
> (load "<install-directory>startup.lisp")
; Loading <install-directory>/startup.lisp
;  Loading <install-directory>/source/mini-module/mini-module-loader.lisp
;   Loading <install-directory>/source/mini-module/mini-module.lisp
;  Loading <install-directory>/source/modules.lisp
T
> (mini-module:compile-module :gbbopen-test :propagate :create-dirs)
; Compiling <install-directory>/source/mini-module/mini-module.lisp
     ...
;; Running basic GBBopen tests...
     ...
```

GBBopen should compile, load, and run the basic tests file without error.

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-specification string must be entered as two backslash characters. For example:

```
> (load "c:\\GBBopen\\startup.lisp")
```

As can be seen from the file-loading messages, the `startup.lisp` file loads a bootstrap loader file for the module facility (the file `source/mini-module/mini-module-loader.lisp`). This bootstrap file then loads the module system file (`source/mini-module/mini-module.lisp`) followed by the module definitions for all GBBopen modules (contained in the file `source/modules.lisp`).

The `:propagate` option to **compile-module** causes any file in the required modules to be compiled if its binary file does not exist or is not up to date. The `:create-dirs` option automatically creates any directories that are missing in the compiled-file directory tree.

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 facility and definitions have been loaded. A personal `gbbopen-init.lisp` file is a very useful mechanism for defining user-specific modules, application modules, GBBopen parameters, and other personalizations.

Here is a simple example of a personal `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)

(mini-module:define-root-directory :my-app-root
```

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

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

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

Next, we show how to add convenient GBBopen keyword commands to the read-eval-print loop (REPL) in your Common Lisp environment and how to define a personal top-level command that makes it easy to compile and load the `:my-app` module—even if GBBopen has not yet been loaded into your Common Lisp image.

GBBopen top-level commands

The file <*install-directory*>`gbbopen-init.lisp` is an alternative for <*install-directory*>`startup.lisp` that adds some handy GBBopen keyword commands to the top-level REPL listener for Allegro CL, CLISP, CMUCL, ECL, LispWorks, OpenMCL, 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 top-level command processing. GBBopen does provide a swank extension in the SLIME REPL that supports GBBopen keyword commands.

To make GBBopen keyword commands available, simply load <*install-directory*>`gbbopen-init.lisp` from your personal Common Lisp initialization file. (You should not explicitly load the <*install-directory*>`startup.lisp` file when using <*install-directory*>`gbbopen-init.lisp`—it will handle the <*install-directory*>`startup.lisp` file loading for you.)

The standard GBBopen top-level commands are defined in the file <*install-directory*>`commands.lisp`. In many Common Lisp implementations, top-level keyword commands that have arguments can be specified using either a list or a spread representation. For example:

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

or

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

However, OpenMCL and the SLIME REPL interface do not support the spread representation, and Allegro CL and LispWorks do not support the list representation. CLISP also does not support the list representation and currently does not support command arguments in its native REPL.

Equivalent functions in the `:common-lisp-user` package are always defined for each GBBopen top-level keyword command, and these functions can be used when top-level keyword processing is not fully supported.

Personal gbbopen-commands.lisp file

You can add your own top-level 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 home directory (as defined by **user-homedir-pathname**), it is loaded automatically by GBBopen's `gbbopen-init.lisp` file. For example, the following personal `gbbopen-commands.lisp` file defines a top-level keyword 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-tll-command :my-app (&rest options)
  "Compile and load my GBBopen application module"
  (startup-module :my-app options :gbbopen-user))

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

The macro `define-tll-command` and the function `startup-module` that are used in this example are both defined in the file <*install-directory*>`gbbopen-init.lisp`. Most GBBopen users prefer to have their Common Lisp initialization file load <*install-directory*>`gbbopen-init.lisp` rather than <*install-directory*>`startup.lisp` in order to use the standard keyword commands for GBBopen—and their own, personalized extensions—in the Lisp listener.

Personal gbbopen-modules directory

Earlier, we showed a simple example of using a personal `gbbopen-init.lisp` file to define the module `:my-app`. If you develop or use a number of modules or applications, GBBopen provides an alternative mechanism that is even more convenient.

If a `gbbopen-modules` directory is present in the user's home directory (as defined by **user-homedir-pathname**), it is assumed to contain symbolic links (or "pseudo symbolic-link" files on Windows) to individual GBBopen module directory trees. Each module directory tree can contain:

- a `commands.lisp` file that specifies top-level commands for the module (loaded after the personal `gbbopen-commands.lisp` file if there is one in the user's home directory)
- a `modules.lisp` file that contains module definitions (loaded after the personal `gbbopen-init.lisp` file if there is one in the user's home directory)
- a directory named `source` containing all the additional source files for the module or application

We highly recommend following this packaging convention, which mirrors that of GBBopen itself. It is very easy to 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 of type `.sym` (containing target directory path as its sole line) as a stand-in for the symbolic link. The *GBBopen Tutorial* will provide a more detailed example of this.

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 154) module. The browser used by **browse-hyperdoc** is specified by the value of `*preferred-browser*`. A different

value can be provided in either your Common Lisp initialization file or in your personal `gbbopen-init.lisp` file. Changing the default setting in `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 `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!

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 155)

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.

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

Purpose

Define a top-level-loop command.

Package

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

Module

Defined in `gbbopen-init.lisp`

Arguments

command-name A keyword symbol naming the command
lambda-list A lambda-list
declaration A declare expression
documentation A string
form A form

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).

Documentation is a documentation string that may be associated the the top-level-loop command *command-name* in certain Common Lisp implementations.

See also

startup-module (page 9)

Examples

Define a top-level-loop command named `:ds` to be a handy shortcut to the Common Lisp `describe` function:

```
(define-tll-command :ds (obj)
  "Describe"
  (describe (eval obj)))
```

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

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

startup-module *module-name options package-name* [*Function*]

Purpose

Compile and load a GBBopen module, even if GBBopen is not yet loaded, and set the current package.

Package

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

Module

Defined in `gbbopen-init.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
`: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

package-name A package name

Description

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

See also

compile-module (page 14)
define-module (page 15)
define-tll-command (page 8)

Example

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

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

2 Mini-Module System

The mini-module 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 mini-module facility was designed for simplicity and portability, yet it is powerful enough to manage substantial software projects, each operating on multiple hardware platforms and Common Lisp implementations.

The mini-module facility is sufficient for many situations, and if not, there are more complex open-source **defsystem** packages, such as ASDF (http://www.weitz.de/asdf-install/), that are available.

The `:mini-module` module is automatically loaded by the GBBopen <*install-directory*>`startup.lisp` file (via the `mini-module-loader.lisp` file located in the `mini-module` 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 mini-module facility by the `startup.lisp` file. A personal `gbbopen-init.lisp` file is very useful for defining user-specific modules, application modules, GBBopen parameters, and other personalizations (see Section 1).

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 `:mini-module`)

Module

`:mini-module`

Value type

A generalized boolean

Initial value

`nil`

Description

By default, ***automatically-create-missing-directories*** is set to `nil`, but it can be set or bound to true to instruct **compile-module** to create any needed directories in the compiled-file tree rather than generating a continuable error. The compile-module option `:create-dirs` can also be specified to bind ***automatically-create-missing-directories*** to true during module compilation.

See also

compile-module (page 14)

Example

Automatically create, as needed, any missing directory in the compiled-file tree, rather than generating a continuable error:

```
(setf common-lisp-user::*automatically-create-missing-directories* 't)
```

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 `:mini-module`)

Module
`:mini-module`

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 14)
load-module (page 21)
load-module-file (page 22)

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))
```

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

Purpose
Compiles and loads the files in a module.

Package `:mini-module`

Module `:mini-module`

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
`: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

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

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**):

`: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
`:source`	Do not compile the file (load the source instead)

See also

automatically-create-missing-directories	(page 12)
autorun-modules	(page 13)
define-module	(page 15)
load-module	(page 21)
load-module-file	(page 22)
startup-module	(page 9)

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

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

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

Purpose
Defines a module to the module facility.

Package `:mini-module`

Module `:mini-module`

Arguments
module-name A keyword symbol naming a module

module-options See below

Errors
The value `: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** `)`

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 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`, and `:files` 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.

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.

File-options have the following effects:

`: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
`:source`	Do not compile the file (load the source instead)

See also

define-relative-directory (page 17)
define-root-directory (page 18)
compile-module (page 14)
load-module (page 21)
load-module-file (page 22)

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

---

define-module

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

Purpose

Defines a directory relative to a root or relative directory.

Package `:mini-module`

Module `:mini-module`

Arguments

module-name	A keyword symbol naming the relative directory
directory	The keyword symbol name of a root or relative directory ancestor of the relative directory
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 place of 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. If this directory location changes, the relative directory is adjusted automatically.

See also

define-root-directory (page 18)
show-defined-directories (page 25)

Examples

Define a relative directory named `:my-tests`:

```
(define-relative-directory :my-tests :my-app-root "tests")
```

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

```
(define-relative-directory :my-performance-tests :my-tests "performance")
```

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

Purpose

Defines a root directory.

Package `:mini-module`

Module `:mini-module`

Arguments

name	A keyword symbol naming the root directory
directory-specification	One of the following: • A string specifying the root directory • A pathname specifying the root directory • A keyword naming a previously defined root directory • 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 place of 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 handled 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.

See also

define-relative-directory (page 17)
show-defined-directories (page 25)

Examples

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

```
(define-root-directory :my-app-root
  (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
  (make-pathname :directory *load-truename*))
```

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

Purpose
Print information about a module.

Package :mini-module

Module :mini-module

Arguments
module-name A keyword symbol naming a module

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

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

See also
define-module (page 15)

Example
Describe the :gbbopen-tools module:

```
> (describe-module :gbbopen-tools)
Module :gbbopen-tools (loaded)
  Requires: (:mini-module)
  Fully expanded requires: (:mini-module)
  Source directory: /home/gbbopen/current/source/tools/
  Compiled directory: /home/gbbopen/current/linux86-allegro-8.0/tools/
  Forces recompile date: Jan 15 17:11
  Files: Jan 15 17:11 preamble
         Jan 15 17:11 declarations (:forces-recompile)
         Jan 15 17:11 declared-numerics (:forces-recompile)
         Jan 15 17:11 defflags (:forces-recompile)
         Jan 15 17:11 tools (:forces-recompile)
         Jan 15 17:11 define-class (:forces-recompile)
         Jan 15 17:11 epilogue
```

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

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

Package :mini-module

Module :mini-module

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

Returns
The 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. The pathname returned for relative directories is the source pathname.

See also
define-relative-directory (page 17)
define-root-directory (page 18)
show-defined-directories (page 25)

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 module:

```
> (get-directory :gbbopen-tools)
#P"/home/GBBopen/current/source/tools"
```

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

Purpose
Loads the files in a module.

Package `:mini-module`

Module `:mini-module`

Arguments
module-name A keyword symbol naming a module

option Any of the following keywords:

`:noautorun`	Binds ***autorun-modules*** to nil during compilation and loading
`:print`	Incrementally prints information during compilation and 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 fi

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

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**):

`: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 13)
compile-module (page 14)
define-module (page 15)
load-module-file (page 22)

Example
Load the GBBopen User module and all its required modules:

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

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

Purpose
Loads a single file from a module.

Package `:mini-module`

Module `:mini-module`

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 compilation and loading
`:print`	Incrementally prints information during compilation and loading
`:source`	Loads from the source file even if the existing compiled file is newer than the source fi

Returns
The pathname of the loaded file.

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

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 13)
define-module (page 15)
load-module (page 21)

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

```
> (load-module-file :gbbopen-tools "tools" :source)
#P"/home/GBBopen/current/source/tools/tools.lisp"
```

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

Purpose
Return the source and compiled directories of a module.

Package `:mini-module`

Module `:mini-module`

Arguments
module-name A keyword symbol naming a module

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

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

See also
define-module (page 15)
load-module-file (page 22)

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

```
> (module-directories :agenda-shell)
#P"/home/GBBopen/current/source/gbbopen/control-shells"
#P"/home/GBBopen/current/compiled-dir-name/gbbopen/control-shells"
```

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

Purpose
Returns a value indicating whether a module has been fully loaded into Common Lisp.

Package `:mini-module`

Module `:mini-module`

Arguments
module-name A keyword symbol naming a module

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

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

See also
define-module (page 15)
load-module (page 21)

Example
Check if the Agenda Shell module has been loaded:

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

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

Purpose

Show all root and relative directory definitions.

Package :mini-module

Module :mini-module

See also

define-relative-directory (page 17)
define-root-directory (page 18)
get-directory (page 20)

Example

List the currently defined root and relative directories:

```
> (show-defined-directories)
:gbbopen
    Relative to :gbbopen-root
    Subdirectories: ("gbbopen")
:gbbopen-root
    Root: /home/gbbopen/current/
:gbbopen-tools
    Relative to :gbbopen-root
    Subdirectories: ("tools")
:mini-module-root
    Root: /home/gbbopen/current/
```

3 Tools

The GBBopen Tools module, `:gbbopen-tools`, contains useful Common Lisp additions and utilities. Convenient (shorthand macros (see page 72) for declared fixnum, short-float, single-float, double-float, and long-float numeric operators are also provided by the `:gbbopen-tools` module.

Uniform interfaces to implementation-specific thread (multiprocessing) capabilities are provided by the `:portable-threads` module (see page 83), polling functions for Common Lisp implementations that do not provide thread capabilities are provided by the `:polling-functions` module (see page 137), uniform socket interfaces are provided by the `:portable-sockets` module (see page 143), uniform interfaces to the operating system are provided by the `:os-interface` module (see page 154), and queue-management objects and operations are provided by the `:queue` module (see page 158).

:full-safety [*Feature*]

Purpose

Disable GBBopen optimizations.

Example

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

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

month-preceeds-date [*Variable*]

Purpose
Control month and date ordering for GBBopen date formatting.

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

Module
`:mini-module`

Value type
A generalized boolean

Initial value
True

See also
brief-date-and-time (page 31)

Examples
Toggle between date formatting options:

```
> (let ((*month-preceeds-date* t))
    (brief-date-and-time))
"Feb 16 12:11"
> (let ((*month-preceeds-date* nil))
    (brief-date-and-time))
"16 Feb 12:11"
```

Note
This variable is defined in the `:mini-module` module to make it available as early as possible.

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

Purpose

Bound a numeric value 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)

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
```

brief-date-and-time `&optional` *universal-time time-zone include-seconds* ⇒ *string* [*Function*]

Purpose

Return a brief date-and-time string.

Package

`:gbbopen-tools (home package is :mini-module)`

Module

`:mini-module`

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)

include-seconds A generalized boolean (default is `nil`)

Returns

A 12-character string (15-characters if *include-seconds* is non-`nil`)

Description

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.

See also

month-preceeds-date (page 29)

Examples

Display the current date and time:

```
> (brief-date-and-time)
"Feb 16 12:11"
```

Display the current date and time (with seconds):

```
> (brief-date-and-time nil nil 't)
"Feb 16 12:11:38"
```

Display the current date and time as GMT:

```
> (brief-date-and-time nil 0)
"Feb 16 17:11"
```

The date and time 10 days ago:

```
> (brief-date-and-time (- (get-universal-time) (* 60 60 24 10)))
"Feb 6 12:11"
```

The date and time 100 days ago:

```
> (brief-date-and-time (- (get-universal-time) (* 60 60 24 100)))
"Nov  8, 2004"
```

The date and time 100 days ago (with seconds):

```
> (brief-date-and-time (- (get-universal-time) (* 60 60 24 100)))
"Nov  8, 2004   "
```

Note

This function is loaded with the `:mini-module` module to make it available as early as possible.

---

brief-date-and-time

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

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 of two arguments that returns a generalized boolean (default is `#'eql`)
test-not	A function 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 of one argument, or `nil` (default is `nil`)

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 91)

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

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

Purpose
Destructively delete all *item*s 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*s 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*.

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

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

define-class *class-name* (*{superclass-name}**) [*documentation*] [*Macro*]
(*{slot-specifier}**) *{class-option}** ⇒ *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*
documentation	A documentation string
slot-specifiers	See below
class-options	See below

Returns

The newly defined 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-class-name` *boolean*) |
(`:export-accessors` *boolean*) |
(`: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

initarg-list An initialization argument list

slot-name A non-`nil`, non-keyword symbol

Description

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

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`.

See also

define-unit-class (page 190)
make-instance (page 236)
with-generate-accessors-format (page 67)

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>
```

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

until (page 63)
while (page 64)

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
```

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

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*.

Examples

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

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

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.

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

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

Package
`:gbbopen-tools`

Module
`:gbbopen-tools`

Arguments
class A class

Returns
The supplied *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

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.

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

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

See also
list-length-2-p (page 44)

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

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

See also
list-length-1-p (page 43)

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
```

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

Purpose
Return a keyword symbol with the same print name as *symbol*.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
symbol A symbol

Returns
The keyword symbol.

Example
```
> (make-keyword 'gbbopen)
:gbbopen
```

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

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
delq (page 35)

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

> (memq 'x '(a b c b))
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 of two arguments that returns a generalized boolean (default is `#'<`)
key A function of one argument, or `nil` (default is `nil`)

Returns
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)
```

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

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

instance A standard-gbbopen-instance object
stream A stream

See also

standard-gbbopen-instance (page 60)

Example

```
(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))))
```

printv *form** ⇒ *result** [*Macro*]

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

Package `:gbbopen-tools`

Module `:gbbopen-tools`

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.

Example
```
> (printv (list 1 2) (list 3 4) '"A string"
          "Return multiple values:" (values 5 6))
;;  (list 1 2) => (1 2)
;;  (list 3 4) => (3 4)
;;  '"A string" => "A string"
;; Return multiple values:
;;  (values 5 6) => 5; 6
5
6
```

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

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

See also
pushnew-acons (page 51)
pushnew/incf-acons (page 53)

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 of one argument, or `nil` (default is `nil`)
test	A function of two arguments that returns a generalized boolean (default is `#'eql`)
test-not	A function of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)

Returns

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

See also

push-acons (page 50)
pushnew/incf-acons (page 53)

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 of one argument, or `nil` (default is `nil`)
test	A function of two arguments that returns a generalized boolean (default is `#'eql`)
test-not	A function of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)

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 of one argument, or `nil` (default is `nil`)
test	A function of two arguments that returns a generalized boolean (default is `#'eql`)
test-not	A function of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)

Returns

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

See also

push-acons (page 50)
pushnew-acons (page 51)

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))
```

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

Purpose
Nondestructively remove a property from a property list.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
plist A property list
indicator An object

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.

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 of two arguments that returns a generalized boolean (default is `#'eql`)

test-not A function of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)

key A function of one argument, or `nil` (default is `nil`)

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 *list1* appears in *list2*.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

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

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

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)
```

sole-element *list* ⇒ *object* [*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

Returns
The sole element of *list*.

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.

spliting-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`)

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)
```

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 **standard-event-instance** and **standard-unit-instance**.

See also

print-instance-slots (page 48)
standard-event-instance (page 262)
standard-unit-instance (page 267)

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 string (not evaluated)
forms	Zero or more forms

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

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

See also

define-unit-class (page 190)

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 *forms* repeatedly, as long as *test-form* evaluates to `nil`.

Package

`:gbbopen-tools`

Module

`:gbbopen-tools`

Arguments

test-form A form
declaration A declare expression
forms An implicit **progn** of forms to be evaluated

See also

do-until (page 38)
while (page 64)

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 *forms* repeatedly, until *test-form* evaluates to `nil`.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments

test-form A form
declaration A declare expression
forms An implicit **progn** of forms to be evaluated

See also

do-until (page 38)
until (page 63)

Examples

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

with-error-handling *form error-form** $\Rightarrow$ *result** [*Macro*]

Purpose
Evaluate *error-forms* if an error occurs while evaluating *form*.

Package :gbbopen-tools

Module :gbbopen-tools

Arguments
form A form
error-forms Zero or more forms

Returns
The values returned by evaluating *form* unless an error occurs during that evaluation in which case values of evaluating *error-forms* are returned.

Description
If an error occurs while evaluating *form*, *error-forms* are evaluated and the values returned by the last *error-form* is returned. A lexical function, **error-message**, is available for use within *error-forms*. This lexical function accepts no arguments and returns a string describing the error that occurred during the evaluation of *form*.

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)))
;;  (error-message) => "Attempt to divide 10 by zero."
```

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

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

Returns

The values returned by evaluating *form*.

Description

This macro provides a convenient means of declaring small code fragements 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 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

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 36)
define-event-class (page 185)
define-space-class (page 187)
define-unit-class (page 190)

Examples

Define three classes, `point`, `circle` and `rectangle`, generating GBB-sytle "*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)
      (format nil "~a-~a-~a"
        (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
forms	An implicit **progn** of forms to be evaluated

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, the forms associated with each *symbol* and make every *symbol* references inside *form*s 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
forms	An implicit **progn** of forms to be evaluated

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)

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 Declared Numerics

The `:gbbopen-tools` module contains a set of numeric-operation macros that provide convenient shorthands for declaring fixnum, short-float, single-float, double-float, and long-float numeric operators. If the feature **:full-safety** is present at compile time, these macros do not make their fixnum, single-float, double-float, and long-float declarations.

The names of the declared-numeric macros include these "type-indicator" characters:

Characters	Declared Type
&	`fixnum`
$&	`short-float`
$	`single-float`
$$	`double-float`
$$$	`long-float`

Thus, the declared-numeric + macros are:

+&	`fixnum` addition
+$&	`short-float` addition
+$	`single-float` addition
+$$	`double-float` addition
+$$$	`long-float` addition

Some examples of declared-numeric macros 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 macros for each declared numeric type are listed in the following sections.

Notes

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. Digitool's Macintosh Common Lisp maps both double-float and single-float types to the 64-bit IEEE 754 double format (*only* the short-float type maps to the IEEE 754 single format).

Fixnum numeric-operation macros

These fixnum numeric-operation macros are defined in the `:gbbopen-tools` module:

Macro	Operator	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)`
`evenp&`	`evenp`	`(evenp& x)`
`floor&`	`floor`	`(floor& x divisor)`
`incf&`	`incf`	`(incf& 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))
```

Short-float macros

These short-float numeric operation macros are defined in the `:gbbopen-tools` module:

Macro	Operator	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)
evenp$&	evenp	(evenp$& x)
floor$&	floor	(floor$& x divisor)
incf$&	incf	(incf$& 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 macros

These single-float numeric operation macros are defined in the `:gbbopen-tools` module:

Macro	Operator	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)
evenp$	evenp	(evenp$ x)
floor$	floor	(floor$ x divisor)
incf$	incf	(incf$ 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 macros

These double-float numeric operation macros are defined in the `:gbbopen-tools` module:

Macro	Operator	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)
evenp$$	evenp	(evenp$$ x)
floor$$	floor	(floor$$ x divisor)
incf$$	incf	(incf$$ 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 macros

These long-float numeric operation macros are defined in the `:gbbopen-tools` module:

Macro	Operator	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)
evenp$$$	evenp	(evenp$$$ x)
floor$$$	floor	(floor$$$ x divisor)
incf$$$	incf	(incf$$$ 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))
```

3.2 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.

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`)

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
ot2ut (page 80)
set-ot-base (page 81)
ut2ot (page 82)

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-univeral-time An Offset Universal Time value

Returns
The equivalent Universal Time value

Errors
The Offset Universal Time time-base value has not been set.

See also
check-ot-base (page 79)
ut2ot (page 82)
set-ot-base (page 81)

Example
```
> (set-ot-base 1 7 2007)
3409014016
> (ot2ut -15071348)
3393942668
```

set-ot-base [*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
month	An integer between 1 and 12, inclusive
year	An integer indicating the year A.D. However, if this 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. This can be useful for initializing applications that do not need to represent historical dates or save or communicate Offset Universal Time values.

See also
check-ot-base (page 79)
ot2ut (page 80)
ut2ot (page 82)

Examples
Set the time base for Offset Universal Time to today:

```
> (set-ot-base)
3410655616
```

Set the time base for Offset Universal Time to July 1, 2007:

```
> (set-ot-base 1 7 2007)
3409014016
```

ut2ot `&optional` *universal-time* ⇒ *offset-universal-time* [*Function*]

Purpose
Convert a Offset Universal Time value to an Offset Universal Time value.

Package `:gbbopen-tools`

Module `:gbbopen-tools`

Arguments
univeral-time A Universal Time value (default is the current Universal Time)

Returns
The equivalent Offset Universal Time value

Errors
The Offset Universal Time time-base value has not been set.

See also
check-ot-base (page 79)
ot2ut (page 80)
set-ot-base (page 81)

Examples
```
> (set-ot-base 1 7 2007)
3409014016
> (ut2ot)
-15071348
> (ut2ot 3393942668)
-15071348
```

3.3 Portable Threads

The `:portable-threads` module 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). GBBopen adds the feature `:threads-not-available` on Common Lisp implementations without thread support.

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**. Note that 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 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 some complex and expensive interaction between the operating-system scheduling and the Common Lisp implementation. Given this cost and complexity, some Common Lisp implementations using 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`.

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 managed and invoked by a separate `"Scheduled-Function Scheduler"` thread. Unless the run time of the invoked function is brief, the 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 underlying Common Lisp's `sleep` function. A separate thread is spawned to manage each periodic function.

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 115)
schedule-function-relative (page 117)
unschedule-function (page 132)

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 17 30 0)
    :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...
```

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 107)
spawn-periodic-function (page 121)

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
```

all-scheduled-functions <no arguments> ⇒ *list-of-scheduled-functions* [*Function*]

Purpose

Return a list of all scheduled functions that are currently scheduled.

Package

`:portable-threads`

Module

`:portable-threads`

Returns

A list of `scheduled-function` objects.

See also

make-scheduled-function (page 112)
schedule-function (page 115)
schedule-function-relative (page 117)
unschedule-function (page 132)

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.

all-threads <no arguments> ⇒ *list-of-threads* [*Function*]

Purpose
Return a list of all threads.

Package
`:portable-threads`

Module
`:portable-threads`

Returns
A list of objects representing the threads.

See also
current-thread (page 104)
thread-alive-p (page 123)
threadp (page 127)
spawn-thread (page 129)

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** ⇒ *value* [*Macro*]

Purpose
Execute *forms* as an atomic operation.

Package `:portable-threads`

Module `:portable-threads`

Arguments
form A form

Returns
The single value returned by evaluating the last *form*.

Description
This macro provides atomicity in the following entities: **atomic-decf**, **atomic-delete**, **atomic-flush**, **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 90)
atomic-delete (page 91)
atomic-flush (page 93)
atomic-incf (page 94)
atomic-push (page 96)
atomic-pushnew (page 97)
atomic-pop (page 95)

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).

Returns
The new value of *place*.

See also
as-atomic-operation (page 89)
atomic-delete (page 91)
atomic-flush (page 93)
atomic-incf (page 94)
atomic-pop (page 95)
atomic-pushnew (page 97)

Examples
```
> x
5
> (atomic-decf x)
4
> (atomic-decf x 1.5)
2.5
```

atomic-delete *item place* `&key` *from-end test test-not start end count key* ⇒ *result-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 of two arguments that returns a generalized boolean (default is `#'eql`)
test-not	A function 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 of one argument, or `nil` (default is `nil`)

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 89)
atomic-flush (page 93)
atomic-pop (page 95)
atomic-push (page 96)
atomic-pushnew (page 97)
counted-delete (page 33)

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

Returns

The *place* value prior to being set to `nil`.

See also

as-atomic-operation (page 89)
atomic-delete (page 91)
atomic-pop (page 95)
atomic-push (page 96)
atomic-pushnew (page 97)

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).

Returns
The new value of *place*.

See also
as-atomic-operation (page 89)
atomic-decf (page 90)

Examples
```
> x
2
> (atomic-incf x)
3
> (atomic-incf x 1.5)
4.5
```

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

Returns

The first element (the car) of the list stored in *place*.

See also

as-atomic-operation (page 89)
atomic-delete (page 91)
atomic-flush (page 93)
atomic-push (page 96)
atomic-pushnew (page 97)

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

Returns
The new value of *place*.

See also
as-atomic-operation (page 89)
atomic-delete (page 91)
atomic-flush (page 93)
atomic-pop (page 95)
atomic-pushnew (page 97)

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 of one argument, or `nil` (default is `nil`)

test A function of two arguments that returns a generalized boolean (default is `#'eql`)

test-not A function of two arguments that returns a generalized boolean (use of `:test-not` is deprecated)

Returns

The new value of *place*.

See also

as-atomic-operation (page 89)
atomic-delete (page 91)
atomic-flush (page 93)
atomic-pop (page 95)
atomic-push (page 96)

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.

See also
hibernate-thread (page 106)

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 109)
define-class (page 36)

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 101)
condition-variable-wait (page 102)
condition-variable-wait-with-timeout (page 103)
make-condition-variable (page 109)

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 100)
condition-variable-wait (page 102)
condition-variable-wait-with-timeout (page 103)
make-condition-variable (page 109)

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 100)
condition-variable-signal (page 101)
condition-variable-wait-with-timeout (page 103)
make-condition-variable (page 109)

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* [*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

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 100)
condition-variable-signal (page 101)
condition-variable-wait (page 102)
make-condition-variable (page 109)

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`

Returns
The object representing the current thread.

See also
all-threads (page 88)
spawn-thread (page 129)

Example
```
> (current-thread)
#<thread Listener 1>
```

Note
On Common Lisp implementations without threads, `nil` is returned.

encode-time-of-day *hour minute second* `&optional` *universal-time* ⇒ *universal-time* [*Function*]

Purpose
Return a universal time representing a specified time-of-day.

Package `:portable-threads`

Module `:portable-threads`

Arguments
hour	An integer between 0 and 23, inclusive
minute	An integer between 0 and 59, inclusive
second	An integer between 0 and 59, 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
schedule-function (page 115)

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 17 0 0) :repeat-interval #.(* 24 60 60))
```

Verbosely change `quitting-time` to 5:30pm every day:

```
> (schedule-function 'quitting-time (encode-time-of-day 17 30 0)
    :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...
```

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 98)

Example
Hibernate the current thread:

```
(hibernate-thread)
```

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 86)
all-threads (page 88)
kill-thread (page 108)
spawn-periodic-function (page 121)

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>
```

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-thread (page 129)
thread-alive-p (page 123)

Example
```
(kill-thread thread)
```

make-condition-variable `&rest` *initargs* [*Function*]
`&key` *class*
⇒ *condition-variable*

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`)

Returns
The created **condition-variable**.

See also
make-instance (page 236)

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>
```

make-lock `&key` *name* ⇒ *lock* [*Function*]

Purpose
Create a lock.

Package `:portable-threads`

Module `:portable-threads`

Arguments
name A string.

Returns
The newly created lock.

See also
make-condition-variable (page 109)
make-recursive-lock (page 111)
thread-holds-lock-p (page 131)
with-lock-held (page 134)

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.

Returns
The newly created recursive lock.

See also
make-condition-variable (page 109)
make-lock (page 110)
thread-holds-lock-p (page 131)
with-lock-held (page 134)

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.

make-scheduled-function *function* `&key` *name* ⇒ *scheduled-function* [*Function*]

Purpose
Create a scheduled function.

Package `:portable-threads`

Module `:portable-threads`

Arguments
function A function of one argument

name An object (typically a string or a symbol; default is *function*, if *function* is a symbol, otherwise `nil`)

Returns
The newly created scheduled function.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

Description
Unless the run time of *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.

See also
schedule-function-verbose (page 85)
all-scheduled-functions (page 87)
schedule-function (page 115)
schedule-function-relative (page 117)
scheduled-function-name (page 119)
scheduled-function-repeat-interval (page 120)
unschedule-function (page 132)

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]>
```

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]>
```

---

make-scheduled-function

restart-scheduled-function-scheduler <no arguments> ⇒ *thread* [*Function*]

Purpose
Restart the scheduled-function scheduling thread.

Package
`:portable-threads`

Module
`:portable-threads`

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 accidently, this function can be used to start a new scheduler thread.

See also
schedule-function (page 115)
scheduled-function-repeat-interval (page 120)
unschedule-function (page 132)

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
```

schedule-function *name-or-scheduled-function invocation-time* `&key` *repeat-interval verbose* [*Function*]

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
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* object is unscheduled, it is added to the list of currently scheduled scheduled functions with the specified *invocation-time* and *repeat-interval*. If the *scheduled-function* object is currently scheduled, it is first unscheduled and then rescheduled with the specified *invocation-time* and *repeat-interval*.

See also

schedule-function-verbose	(page 85)
all-scheduled-functions	(page 87)
encode-time-of-day	(page 105)
make-scheduled-function	(page 112)
restart-scheduled-function-scheduler	(page 114)
schedule-function-relative	(page 117)
scheduled-function-repeat-interval	(page 120)
unschedule-function	(page 132)

Examples
Schedule a scheduled function that simply prints `"Happy New Year!"` at midnight (local time) on January 1, 2010:

```
> (schedule-function
    (make-scheduled-function
      #'(lambda (scheduled-function)
          (declare (ignore scheduled-function))
          (print "Happy New Year!")))
     (encode-universal-time 0 0 0 1 1 2010))
> (all-scheduled-functions)
(#<scheduled-function nil [Jan 1, 2010 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 17 0 0) :repeat-interval #.(* 24 60 60))
```

Verbosely change `quitting-time` to 5:30pm every day:

```
> (schedule-function 'quitting-time (encode-time-of-day 17 30 0)
    :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-function

schedule-function-relative *name-or-scheduled-function seconds* `&key` *repeat-interval verbose* [*Function*]

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
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* object is unscheduled, it is added to the list of currently scheduled scheduled functions with an invocation time of *interval* seconds from the current time and the specified *repeat-interval*. If the *scheduled-function* object is currently scheduled, it is first unscheduled and then rescheduled with an invocation time of *interval* seconds from the current time and the specified *repeat-interval*.

See also

schedule-function-verbose	(page 85)
all-scheduled-functions	(page 87)
make-scheduled-function	(page 112)
restart-scheduled-function-scheduler	(page 114)
schedule-function	(page 115)
scheduled-function-repeat-interval	(page 120)
unschedule-function	(page 132)

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)
```

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-name *scheduled-function* [*Function*]

Purpose
Return the name of a scheduled function.

Setf syntax
(setf (scheduled-function-repeat-interval *scheduled-function*) *name*)

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 87)
make-scheduled-function (page 112)
schedule-function (page 115)
schedule-function-relative (page 117)
scheduled-function-name (page 119)

Example
Return the names of all currently scheduled scheduled functions:

```
> (mapcar #'scheduled-function-name (all-scheduled-functions))
(quitting-time)
```

scheduled-function-repeat-interval *scheduled-function* [*Function*]

Purpose
Return the repeat interval of a scheduled function.

Setf syntax
(setf (scheduled-function-repeat-interval *scheduled-function*) *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 87)
make-scheduled-function (page 112)
schedule-function (page 115)
schedule-function-relative (page 117)
scheduled-function-name (page 119)

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))
```

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 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***)

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 86)
all-threads	(page 88)
kill-periodic-function	(page 107)
kill-thread	(page 108)
make-scheduled-function	(page 112)
schedule-function-relative	(page 117)

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

thread-alive-p *thread* ⇒ *boolean* [*Function*]

Purpose
Return a value indicating whether a thread is alive.

Package :portable-threads

Module :portable-threads

Arguments
thread A thread

Returns
True if *thread* is alive; `nil` otherwise.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
all-threads (page 88)
kill-thread (page 108)
spawn-thread (page 129)

Examples
```
> (defparameter *silly-thread* (spawn-thread "Sleeper" 'sleep 10000))
#<thread Sleeper>
> (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*)

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-thread (page 129)

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.

Package
`:portable-threads`

Module
`:portable-threads`

Arguments
thread A thread

Returns
The whostate string of the thread or `nil`.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
spawn-thread (page 129)

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.

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

Returns
True if *object* is an object representing a thread; `nil` otherwise.

See also
all-threads (page 88)
spawn-thread (page 129)
thread-alive-p (page 123)

Example
```
> (threadp (car (all-threads)))
t
```

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
args Arguments to the function

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also
spawn-thread (page 129)

Example
```
(run-in-thread thread
               #'(lambda (result) (throw ':exit result))
               result)
```

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
args Arguments to the function

Returns
The object representing the new thread.

Errors
Threads (multiprocessing) is not supported on the Common Lisp implementation.

See also

all-threads (page 88)
awaken-thread (page 98)
current-thread (page 104)
hibernate-thread (page 106)
kill-thread (page 108)
thread-alive-p (page 123)
thread-name (page 124)
thread-whostate (page 125)
threadp (page 127)
run-in-thread (page 128)
symbol-value-in-thread (page 130)

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

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-thread (page 129)

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-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

Returns
True if the current thread holds *lock*; `nil` otherwise.

See also
make-condition-variable (page 109)
make-lock (page 110)
make-recursive-lock (page 111)
with-lock-held (page 134)

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
```

unschedule-function *name-or-scheduled-function* `&key` *verbose* ⇒ *boolean* [*Function*]

Purpose

Cancel the upcoming invocation 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
verbose	A generalized boolean (default is ***schedule-function-verbose***)

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 scheduled, it is removed from the list of currently scheduled scheduled functions.

See also

schedule-function-verbose	(page 85)
all-scheduled-functions	(page 87)
make-scheduled-function	(page 112)
schedule-function	(page 115)
schedule-function-relative	(page 117)

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, 2010 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

with-lock-held (*lock* `&key` *whostate*) *form** ⇒ *value** [*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

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 109)
make-lock (page 110)
make-recursive-lock (page 111)
thread-holds-lock-p (page 131)
thread-whostate (page 125)

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** ⇒ *value** [*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 completed before *seconds* seconds have elapsed

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

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.

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 103)

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 2 seconds)
```

3.4 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
priority A fixnum (default is `0`)

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

See also
describe-all-polling-functions (page 139)
remove-all-polling-functions (page 141)
remove-polling-function (page 140)
run-polling-functions (page 142)

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 138)
remove-all-polling-functions (page 141)
remove-polling-function (page 140)
run-polling-functions (page 142)

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

See also
add-polling-function (page 138)
describe-all-polling-functions (page 139)
remove-all-polling-functions (page 141)
run-polling-functions (page 142)

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 138)
describe-all-polling-functions (page 139)
remove-polling-function (page 140)
run-polling-functions (page 142)

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 138)
describe-all-polling-functions (page 139)
remove-all-polling-functions (page 141)
remove-polling-function (page 140)
start-control-shell (page 310)

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.

3.5 Portable Sockets

The `:portable-sockets` module provides a uniform interface to commonly used socket entitites.

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`)

Returns
A socket stream.

See also
shutdown-socket-stream (page 150)
start-connection-server (page 151)

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)
#<stream socket 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 147)

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`)

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 148)
remote-hostname-and-port (page 149)
with-open-connection (page 153)

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
```

Again, 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`)

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 144)
start-connection-server (page 151)

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

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 150)

with-open-connection (page 153)

Example
Open a socket connection to the GBBopen Project web server:

```
> (open-connection "GBBopen.org" 80)
#<stream socket 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* ⇒ *hostname*, *port* [*Function*]

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 (default is `nil`)

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 148)
local-hostname-and-port (page 146)
with-open-connection (page 153)

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
```

Again, 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 151)
open-connection (page 148)
with-open-connection (page 153)

Example
Tell the other end of a socket connection that we are done sending output on the socket stream (send an end-of-file indictation):

```
(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* ⇒ *thread* [*Function*]

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 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`)

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 108)
open-connection (page 148)
with-open-connection (page 153)

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
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 148)
shutdown-socket-stream (page 150)

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
```

3.6 OS Interface

The `:os-interface` module provides a uniform interface to commonly used operating-system entitites.

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

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 5) for details).

See also
preferred-browser (page 7)

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 157)

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"
```

run-external-program *program args* &key *input output wait* ⇒ *bidirectional-stream*, *process-id* [*Function*]

Purpose
Run an external program.

Package
`:gbbopen-tools`

Module
`:os-interface`

Arguments
program	A string specifying the name of the program to be run
arguments	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`)

Returns
Two values:

- an input, output, or bi-directional stream or `nil`
- a process identifier, 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 156)

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"
```

3.7 Queue Management

The :queue module provides queue-management objects and operators.

do-queue (*var queue*) *declaration** {*tag* | *form*}* [*Macro*]

Purpose
Iterate over each queue element on the specified *queue*.

Package :gbbopen-tools

Module :queue

Arguments
var A variable symbol
queue A GBBopen queue
declaration A declare expression
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 164)
queue (page 170)
ordered-queue (page 168)

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-tools`

Module `:queue`

Arguments
queue A GBBopen queue

Returns
The first queue element on *queue*.

See also
last-queue-element (page 162)
nth-queue-element (page 166)

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-tools`

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
make-queue (page 163)

remove-from-queue (page 173)

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-tools`

Module `:queue`

Arguments
queue A GBBopen queue

Returns
The last queue element on *queue*.

See also
first-queue-element (page 160)
nth-queue-element (page 166)

Example
```
> (last-queue-element pending-ksas)
#<ksa 2372>
```

make-queue `&rest` *initargs* ⇒ *queue* [*Function*]

Purpose
Make a GBBopen queue.

Package `:gbbopen-tools`

Module `:queue`

Arguments
initargs An initialization argument list

Returns
The newly created queue.

See also
queue (page 170)
ordered-queue (page 168)

Example
```
> (setf pending-ksas (make-queue :class 'ordered-queue
                                 :key #'rating-of))
#<ordered-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-tools`

Module `:queue`

Arguments
function A function 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 159)
queue (page 170)
ordered-queue (page 168)

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-tools`

Module `:queue`

Arguments
queue-element A GBBopen queue element object

Returns
The queue element that follows *queue-element*.

See also
previous-queue-element (page 169)

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*

Package `:gbbopen-tools`

Module `:queue`

Arguments
n A fixnum
queue A GBBopen queue

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 160)
last-queue-element (page 162)

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* [*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-tools`

Module `:queue`

Arguments
queue-element A GBBopen queue element object

Returns
The queue queue on which *queue-element* resides or `nil` if *queue-element* is not on a queue.

See also
queue-element (page 171)
show-queue (page 174)

Example
Return the queue on which `ksa` resides:

```
> (on-queue-p ksa)
#<ordered-queue>
```

ordered-queue [*Class*]

Package

`:gbb-tools`

Module

`:queue`

Description

The mixin-class used as the header of ordered (sorted) GBBopen queues.

See also

make-queue (page 163)
queue (page 170)
queue-element (page 171)
show-queue (page 174)

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-tools`

Module `:queue`

Arguments
queue-element A GBBopen queue element object

Returns
The queue element that precedes *queue-element*.

See also
next-queue-element (page 165)

Example
```
> (previous-queue-element ksa)
#<ksa 2166>
```

queue [*Class*]

Package :gbb-tools

Module :queue

Description

The mixin-class used as the header of GBBopen queues.

See also

make-queue (page 163)
ordered-queue (page 168)
queue-element (page 171)
show-queue (page 174)

queue-element [*Class*]

Package :gbb-tools

Module :queue

Description

Objects that inherit from the mixin-class **queue-element** can be elements of GBBopen queues.

See also

on-queue-p (page 167)
ordered-queue (page 168)
queue (page 170)

Example

Define a KS activation class whose unit 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-tools`

Module `:queue`

Arguments
queue A GBBopen queue
recount-p If true, actually counts the individual queue elements (default is `nil`)

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-tools`

Module `:queue`

Arguments
queue-element A GBBopen queue element object

Returns
The supplied *queue-element*.

See also
insert-on-queue (page 161)

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-tools`

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 two-argument function used to print each queue element line (default is `#'standard-show-queue-element`)

See also

on-queue-p (page 167)

Example

Show the first five KSAs on the queue `pending-ksas`:

```
> (show-queue pending-ksas :end 5)
    0. #<ksa 2217>
    1. #<ksa 2293>
    2. #<ksa 2303>
    3. #<ksa 2280>
    4. #<ksa 2249>
```

4 GBBopen Core

The GBBopen Core module, `:gbbopen-core`, provides support for the blackboard repository, unit and space classes and instances, and event signaling.

warn-about-unusual-requests [*Variable*]

Purpose

Control warning messages of "unusual" **find-instances** and **filter-instances** requests that are likely to be mistakes.

Package

`:gbbopen`

Module

`:gbbopen-core`

Value type

A generalized boolean

Initial value

True

See also

filter-instances (page 218)
find-instances (page 221)

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
```

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
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`

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`

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 247)
remove-all-event-functions (page 246)

Examples

Add the event function evfn-printv to the set of functions to be invoked when `create-instance-event` is signalled on a `hyp` unit instance:

```
(add-event-function 'evfn-printv 'create-instance-event 'hyp)
```

Add the event function evfn-printv to the set of functions to be invoked when `create-instance-event` is signalled on a `hyp` unit instance or its subclasses:

```
(add-event-function 'evfn-printv 'create-instance-event '(hyp
:plus-subclasses))
```

Note

Unit-instance-specific event functions are not yet implemented in GBBopen.

---

add-event-function

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 `add-instance-to-space-instance-event` is signaled.

See also

define-unit-class (page 190)
make-instance (page 236)
make-space-instance (page 238)
remove-instance-from-space-instance (page 248)

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 119 (1835 4791) .85>
```

or

```
> (add-instance-to-space-instance good-hyp '(bb hyps))
#<hyp 119 (1835 4791) .85>
```

allowed-unit-classes *space-instance* ⇒ *extended-unit-classes-specification-list* [*Generic Function*]

Purpose

Return the extended unit-classes specifications of unit classes whose unit instances are allowed on a space instance.

Method signatures

allowed-unit-classes (*space-instance* `standard-space-instance`) ⇒
extended-unit-class-specification-list

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

space-instance A space instance

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 238)

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 '(bb hyps))
((hyp :plus-subclasses))
```

check-link-consistency `&optional` *silent* ⇒ *boolean* [*Function*]

Purpose
Check for consistency in link-slot definitions of unit classes to be linked.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
silent If true, suppress warning/success printing (default is `nil`)

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-consistency** should be used again to check for additional inconsistencies.

See also
define-unit-class (page 190)

Examples
Check for consistency in link-slot definitions in all unit classes:

```
> (check-link-consistency)
;; All link definitions are consistent.
t
```

Create a link-slot inconsistency:

```
> (define-unit-class bad ()
     ((mismatched-link :link (missing inverse))))
bad
> (check-link-consistency)
Warning: The inverse of link MISMATCHED-LINK in unit-class BAD refers
         to unit-class MISSING, which is not defined.
nil
```

Define the missing unit class, but incorrectly:

```
(define-unit-class missing () ((mismatched-link :link (missing bad))))
missing
> (check-link-consistency)
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-consistency)
;; All link definitions are consistent.
t
```

Check again, but silently:

```
> (check-link-consistency 't)
t
```

---

check-link-consistency

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-spec* `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

Returns

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

map-instances-of-class (page 240)

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
```

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 `remove-instance-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 208)
map-instances-on-space-instances (page 242)

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)))
```

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

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*
documentation	A documentation string
slot-specifiers	See below
class-options	See below

Returns
The newly defined 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-class-name` *boolean*) |
(`:export-accessors` *boolean*) |
(`: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
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-class**.

The `:metaclass` *class-name*, 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 same 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 256)
standard-event-class (page 261)
with-generate-accessors-format (page 67)

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>
```

define-event-class

define-space-class *space-class-name* (*{superclass-name}**) [*documentation*] [*Macro*]
(*{slot-specifier}**) *{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*
documentation	A documentation string
slot-specifiers	See below
class-options	See below

Returns
The newly defined 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
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` *dimensional-value-spec*`*)` |
`(:documentation` *string*`)` |
`(:export-class-name` *boolean*`)` |
`(:export-accessors` *boolean*`)` |
`(: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-comparision-test` *instance-name-comparision-test*`)` |
`(:metaclass` *class-name*`)`

initial-space-instance-specifier ::= {*space-instance-path*`+` | *function*}

dimensional-value-specifier ::= *incomposite-dv-specifier* | *composite-dv-specifier*

incomposite-dv-specifier ::= `(`*dimension-name dimension-value-type dimension-value-place*`)`

composite-dv-specifier ::= `(`*dimension-name dimension-value-type composite-type dimension-value-place*`)`

composite-type ::= `:set` | `:sequence` |
{`:ascending-series` *ordering-dimension-name*} |
{`:descending-series` *ordering-dimension-name*}

dimension-value-type ::= `:point` | `:interval` | `:mixed` | `:element` | `:boolean`

dimension-value-place ::= {*slot-name* [*slot-name*]} | {*function* [*slot-name*]}

dimensional-value-specifier ::= `(`*dimension-name dimension-value-type dimension-value-place*`)`

dimension-value-type ::= `:point` | `:interval` | `:mixed` | `:element` | `:boolean`

dimension-value-place ::= *slot-name* | *slot-name slot-name* | {*function* [*slot-name*]}

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
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-class**.

The `:metaclass` *class-name*, if specified, must be a subclass of **standard-space-class**. The default metaclass value is also **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 dimensional index

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. 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 initializtion value associated with the metaclass is used.

See also

define-unit-class (page 190)
make-space-instance (page 238)
standard-space-class (page 264)
with-generate-accessors-format (page 67)

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-unit-class *unit-class-name* (*{superclass-name}**) [*documentation*] [*Macro*]
(*{slot-specifier}**) *{class-option}** ⇒ *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*
documentation	A documentation string
slot-specifiers	See below
class-options	See below

Returns
The newly defined 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
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` *dimensional-value-specifier*`*)` |
`(:documentation` *string*`)` |
`(:export-class-name` *boolean*`)` |
`(:export-accessors` *boolean*`)` |
`(: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-comparision-test` *instance-name-comparision-test*`)` |
`(:metaclass` *class-name*`)`

initial-space-instance-specifier ::= {*space-instance-path*`+` | *function*}

dimensional-value-specifier ::= *incomposite-dv-specifier* | *composite-dv-specifier*

incomposite-dv-specifier ::= `(`*dimension-name dimension-value-type dimension-value-place*`)`

composite-dv-specifier ::= `(`*dimension-name dimension-value-type*
composite-type dimension-value-place`)`

composite-type ::= `:set` | `:sequence` |
{`:ascending-series` *ordering-dimension-name*} |
{`:descending-series` *ordering-dimension-name*}

dimension-value-type ::= `:point` | `:interval` | `:mixed` | `:element` | `:boolean`

dimension-value-place ::= {*slot-name* [*slot-name*]} | {*function* [*slot-name*]}

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
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.

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-class**.

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 dimensional index 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. 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 initializtion value associated with the metaclass is used.

See also

define-space-class (page 187)
define-class (page 36)
find-instance-by-name (page 220)
link-slot-p (page 235)
make-instance (page 236)
standard-unit-class (page 266)
with-generate-accessors-format (page 67)

Example

```
> (define-unit-class hyp ()
    ((belief :initform 0.0)
     (location :initform nil)
     (classification :initform :unknown)
     (supporting-hyps
      :link (hyp supported-hyps))
     (supported-hyps
      :link (hyp supporting-hyps)))
    (:dimensional-values
     (belief :point belief)
     (classification :element classification)
     (x :point #'location.x location)
     (y :point #'location.y location))
    (:initial-space-instances (bb hyps)))
#<standard-unit-class hyp>
```

---

define-unit-class

delete-instance *unit-instance* ⇒ *unit-instance* [*Generic Function*]

Purpose
Delete a unit instance.

Method signatures
delete-instance (*unit-instance* `standard-unit-instance`) ⇒ *unit-instance*
delete-instance (*space-instance* `standard-space-instance`) ⇒ *space-instance*

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
unit-instance The unit instance to be deleted

Returns
The (deleted) unit instance, *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`
- `remove-instance-from-space-instance-event`

See also
delete-all-space-instances (page 194)
delete-space-instance (page 195)
make-instance (page 236)
reset-gbbopen (page 250)

Example
Create, then delete, a `hyp` unit instance:

```
> (delete-instance (make-instance 'hyp)))
#<hyp 311 (896 388) .68 [Deleted]>
```

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`
- `remove-instance-from-space-instance-event`

See also

delete-space-instance (page 195)
make-space-instance (page 238)
reset-gbbopen (page 250)

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

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`
- `remove-instance-from-space-instance-event`

See also
delete-all-space-instances (page 194)
delete-instance (page 193)
make-space-instance (page 238)
reset-gbbopen (page 250)

Examples
Delete the `(bb hyps)` space instance:

```
> (delete-space-instance (find-space-instance-by-path '(bb hyps))
#<standard-space-instance (bb hyps) [Deleted]>
```

or simply

```
> (delete-space-instance '(bb hyps)
#<standard-space-instance (bb hyps) [Deleted]>
```

describe-blackboard-repository <no arguments> [*Function*]

Purpose

Print information 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.

Package

`:gbbopen`

Module

`:gbbopen-core`

Description

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
----------                    ---------
hyp                                1695
ks                                   13
ksa                                 891
sensor-report                     13744
standard-space-instance               3
```

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`

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`

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 204)
enable-event-printing (page 214)
resume-event-printing (page 253)
suspend-event-printing (page 268)

Example
Describe all event printing:

```
> (describe-event-printing 'instance-event)
instance-event
  standard-unit-instance
  uc-2 [suspended]
  uc-1 [suspended]
  ksa
  ks
  root-space-instance
  standard-space-instance
```

Note

Unit-instance-specific event functions are not yet implemented in GBBopen.

---

describe-event-printing

describe-instance *instance* [*Generic Function*]

Purpose
Describe a unit instance (or a space instance, as a unit instance).

Method signatures
describe-instance (*instance* `standard-unit-instance`)

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
instance A unit instance (or space instance)

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

See also
describe-space-instance (page 200)
make-instance (page 236)
make-space-instance (page 238)

Example
Describe the `hyp` unit instance:

```
> (describe-instance hyp)
Hyp #<hyp 119>
  Instance name: 119
  Space instances: (#<standard-space-instance (bb hyps)>)
  Dimensional values:
    belief 0.85
    identity (:car :truck :bus :motorcycle :duck-boat)
    x 1835
    y 4791
  Non-link slots:
    belief 0.85
    identity (:car :truck :bus :motorcycle :duck-boat)
    location (1835 4791)
  Link slots:
    supporting-hyps (#<hyp 183 (1835 4791) .82>
                     #<hyp 233 (1835 4791) .89>)
```

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 199)
make-space-instance (page 238)

Example
Describe the `hyps` space instance:

```
> (describe-space-instance '(hyps))
  Standard-space-instance #<standard-space-instance (bb hyps)>
    Path: (bb hyps)
    Dimensions: None
    Allowed Unit Classes:
      (hyp :plus-subclasses)
```

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-spec* `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`

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-class (abstract)
  Direct subclasses: None
  Direct slots:
    belief
      :allocation :instance
      :initargs (:belief)
      :initform 0.0
      :readers (belief-of)
      :writers ((setf belief-of))
    location
      :allocation :instance
      :initargs (:location)
      :initform nil
      :readers (location-of)
      :writers ((setf location-of))
    classification
      :allocation :instance
      :initargs (:classification)
      :initform :unknown
```

```
    :readers (classification-of)
    :writers ((setf classification-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 slots:
    instance-name
      :allocation :instance
      :initargs (:instance-name)
    belief
      :allocation :instance
      :initargs (:belief)
      :initform 0.0
    location
      :allocation :instance
      :initargs (:location)
      :initform nil
    classification
      :allocation :instance
      :initargs (:classification)
      :initform :unknown
  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
      classification :element
      x :point
      y :point
  Effective dimensional values:
      belief :point
      classification :element
      x :point
      y :point
```

```
  Initial space instances:
    (bb hyps)
  Effective initial space instances:
    (bb hyps)
```

---

describe-unit-class

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`

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`

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 197)
enable-event-printing (page 214)
resume-event-printing (page 253)
suspend-event-printing (page 268)

Example
Disable all event printing:

```
(disable-event-printing)
```

Note

Unit-instance-specific event functions are not yet implemented in GBBopen.

---

disable-event-printing

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
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`

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 183)
clear-space-instances (page 184)
do-instances-on-space-instances (page 208)
find-instances-of-class (page 223)
map-instances-of-class (page 240)
map-instances-on-space-instances (page 242)
map-sorted-instances-of-class (page 244)

Examples
Delete all unit instances of the class `hyp`:

```
(do-instances-of-class (instance 'hyp)
  (delete-instance instance))
```

Delete all unit instances of the class `hyp` and instances of subclasses of `hyp`:

```
(do-instances-of-class (instance '(hyp :plus-subclasses))
  (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.

do-instances-on-space-instances (*var unit-classes-specifier space-instances* `&key` *pattern filter-before filter-after*) {*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
declaration	A declare expression
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`

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-pattern-op
ordered-dimension-pattern-op ::= `<` | `<=` | `>=` | `>` | `=` | `/=` |
`within` | `covers` | `overlaps` | `starts` | `ends`
enumerated-dimension-pattern-op ::= `eq` | `eql` | `equal` | `equalp`
boolean-dimension-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 72) pattern operators are also supported, for example: `=&`, `=$&`, `=$`, `=$$`, and `=$$$` and `within&`, `within$&`, `within$`, `within$$`, and `within$$$`.

See also

do-instances-of-class (page 206)
find-instances (page 221)
find-instances-of-class (page 223)
map-instances-of-class (page 240)
map-instances-on-space-instances (page 242)
with-find-stats (page 276)
Declared numerics (page 72)

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 of less than 0.5:

```
(do-instances-on-space-instances (instance 'hyp '(bb probable-hyps)
                                   :pattern '(< belief .5))
   (delete-instance instance))
```

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 of two arguments that returns a generalized boolean
key	A function of one argument, or `nil` (default is `nil`)
declaration	A declare expression
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`

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 206)
find-instances-of-class (page 223)
map-instances-of-class (page 240)
map-sorted-instances-of-class (page 244)

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)))
```

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
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 225)
map-space-instances (page 245)

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)))
```

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

effective-nonlink-slot-definition (page 213)
gbbopen-effective-slot-definition (page 226)

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

effective-link-definition (page 212)
gbbopen-effective-slot-definition (page 226)

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`

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`

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 197)
disable-event-printing (page 204)
resume-event-printing (page 253)
suspend-event-printing (page 268)

Example
Enable event printing on all space-instance events of `hyp` unit instances:

```
(enable-event-printing '(space-instance-event :plus-subevents)
                       '(hyp :plus-subclasses))
```

Note

Unit-instance-specific event functions are not yet implemented in GBBopen.

---

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 `create-instance-event` is signalled on a `hyp` unit instance:

```
(add-event-function 'evfn-printv 'create-instance-event 'hyp)
```

expand-interval *interval amount* ⇒ *new-interval* [*Function*]

Purpose
Expand an interval by *amount*.

Package :gbbopen

Module :gbbopen-core

Arguments
interval An interval
amount A number

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 always result in a zero-width *new-interval* at the center point of the original *interval*.

See also
interval-start (page 232)
interval-end (page 231)
shift-interval (page 255)

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)
```

filter-instances *unit-instances pattern* `&key` *filter-before filter-after* ⇒ *matching-unit-instances* [*Function*]

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

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-pattern-op
ordered-dimension-pattern-op ::= `<` | `<=` | `>=` | `>` | `=` | `/=` |
`within` | `covers` | `overlaps` | `starts` | `ends`
enumerated-dimension-pattern-op ::= `eq` | `eql` | `equal` | `equalp`
boolean-dimension-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 72) pattern operators are also supported, for example: `=&`, `=$&`, `=$`, `=$$`, and `=$$$` and `within&`, `within$&`, `within$`, `within$$`, and `within$$$`.

See also

warn-about-unusual-requests (page 176)
find-instance-by-name (page 220)
find-instances (page 221)
Declared numerics (page 72)

Examples

```
> (filter-instances (supporting-hyps-of hyp) '(> belief .8))
(#<hyp 119 (1835 4791) .85>
 #<hyp 183 (1835 4791) .82>
 #<hyp 233 (1835 4791) .89>)
> (filter-instances (supporting-hyps-of hyp) `(within belief (.85 ,infinity)))
(#<hyp 119 (1835 4791) .85>
 #<hyp 233 (1835 4791) .89>)
```

find-instance-by-name *name* `&optional` *unit-class-specifier* ⇒ *unit-instance* [*Function*]

Purpose
Retrieve a unit instance by its name.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
name	The name of a unit instance
unit-class-specifier	An extended unit-class specification (see below; default is `t`)

Returns
The unit instance with the specified *name* of the specified class if one exists; `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`

Description
The `:instance-name-comparison-test` function (`eq`, `eql`, `equal`, or `equalp`) specified in **define-unit-class** is used to match *name* with the unit-instance name. If you are using strings as the names of unit instances, you should specify `equal` or `equalp` as the comparision function in the unit classes of those unit instances.

See also
define-unit-class (page 190)
filter-instances (page 218)
find-instances (page 221)

Example
Find the `hyp` unit instance 119:

```
> (find-instance-by-name 119 'hyp)
#<hyp 119 (1835 4791) .85>
```

find-instances *unit-classes-specifier space-instances pattern* [*Function*]
`&key` *filter-before filter-after* ⇒ *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

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`

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-pattern-op
ordered-dimension-pattern-op ::= `<` | `<=` | `>=` | `>` | `=` | `/=` |
`within` | `covers` | `overlaps` | `starts` | `ends`
enumerated-dimension-pattern-op ::= `eq` | `eql` | `equal` | `equalp`
boolean-dimension-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.

Declared numeric (see page 72) pattern operators are also supported, for example: `=&`, `=$&`, `=$`, `=$$`, and `=$$$` and `within&`, `within$&`, `within$`, `within$$`, and `within$$$`.

See also

warn-about-unusual-requests (page 176)
do-instances-of-class (page 206)
do-sorted-instances-of-class (page 210)
filter-instances (page 218)
find-instance-by-name (page 220)
find-instances-of-class (page 223)
map-instances-of-class (page 240)
map-sorted-instances-of-class (page 244)
Declared numerics (page 72)

Examples

```
> (find-instances 'hyp (find-space-instance-by-path '(bb hyps))
    '(and (= x 1835) (> belief .8)))
(#<hyp 319 (1835 8419) .91>
 #<hyp 119 (1835 4791) .85>
 #<hyp 331 (1835 8419) .88>
 #<hyp 335 (1835 8419) .92>
 #<hyp 183 (1835 4791) .82>
 #<hyp 233 (1835 4791) .89>)
> (find-instances 'hyp '(bb hyps)
    `(and (= x 1835) (> belief ,(find-instance-by-name 331 'hyp))))
(#<hyp 319 (1835 8419) .91>
 #<hyp 335 (1835 8419) .92>
 #<hyp 233 (1835 4791) .89>)
> (find-instances '(hyp :no-subclasses) '(bb hyps)
    '(= (x y) (1835 8419)))
(#<hyp 319 (1835 8419) .91>
 #<hyp 331 (1835 8419) .88>
 #<hyp 335 (1835 8419) .92>)
```

find-instances-of-class *function 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
function	A function of one argument
unit-classes-specifier	An extended unit-classes specification (see below)

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`

See also

class-instances-count (page 183)
do-instances-of-class (page 206)
map-instances-of-class (page 240)
map-instances-on-space-instances (page 242)
map-sorted-instances-of-class (page 244)

Example
Return the list of all `hyp` unit instances:

```
> (find-instances-of-class 'hyp)
(#<hyp 319 (1835 8419) .91>
 #<hyp 119 (1835 4791) .85>
 #<hyp 331 (1835 8419) .88>
   ...
 #<hyp 183 (1835 4791) .82>
 #<hyp 233 (1835 4791) .89>)
```

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-space-instance-by-path *space-instance-path* ⇒ *space-instance* [*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

Returns
The specified space instance if it exists; `nil` otherwise.

See also
find-space-instances (page 225)

Example
Find the space instance with path `(bb hyps)`:

```
> (find-space-instance-by-path '(bb hyps))
#<standard-space-instance (bb hyps)>
```

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

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 `(find-space-instances '(*))` and `(find-space-instances 't)` return all space instances.
`*` 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 is returned unchanged.

See also
find-space-instance-by-path (page 224)
map-space-instances (page 245)

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)>)
```

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 212)
effective-nonlink-slot-definition (page 213)

gbbopen-implementation-version <no arguments> ⇒ *string* [*Function*]

Purpose
Return the GBBopen implementation version.

Package :gbbopen

Module :gbbopen-core

Returns
The GBBopen implementation-version string

Example
Return the GBBopen implementation-version string:

```
> (gbbopen-implementation-version)
"0.9.3"
```

instance-deleted-p *unit-instance* ⇒ *boolean* [*Function*]

Purpose
Determine whether a unit instance has been deleted.

Package :gbbopen

Module :gbbopen-core

Arguments
unit-instance A unit instance

Returns
True if the unit instance is deleted; `nil` otherwise.

See also
delete-instance (page 193)

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* ⇒ *dimension-value, dimension-value-type, composite-type* [*Function*]

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*

Returns
Four values:

- the value of the specified dimension
- the dimension-value type of the returned dimension value
- the composite type of the returned dimension value if it is a composite dimension value; `nil` if it is a incomposite dimension value
- the name of the ordering dimension if the dimension value is a series composite; `nil` otherwise

Errors
The dimension *dimension-name* is not defined for *unit-instance*.

Examples
Return the `x` dimension value of the unit instance, `hyp`:

```
> (instance-dimension-value hyp 'x)
1835
:point
nil
nil
```

Return the `identity` dimension value:

```
> (instance-dimension-value hyp 'identity)
(:truck :bus :duck-boat)
:element
:set
nil
```

instance-name-of *instance* ⇒ *name* [*Generic Function*]

Purpose
Return the name of a unit instance.

Setf syntax
(setf (instance-name-of *instance*) *name*)

Method signatures
instance-name-of (*instance* `standard-event-instance`) ⇒ *name*
instance-name-of (*instance* `standard-unit-instance`) ⇒ *name*

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
instance A unit instance or an event instance
name An object

Returns
The name of the *instance*.

See also
make-instance (page 236)

Examples
Return the 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 name of `hyp 180` to `"bogus-180"`:

```
> (setf (instance-name-of (find-instance-by-name 180 'hyp)) "bogus-180")
"bogus-180"
```

interval-end *interval* ⇒ *end-value* [*Function*]

Purpose
Obtain the end value of an interval.

Setf syntax
(setf (interval-end *interval*) *end-value*)

Package :gbbopen

Module :gbbopen-core

Arguments
interval An interval
end-value A number

Returns
The end value of the interval.

See also
expand-interval (page 217)
interval-end (page 231)
shift-interval (page 255)

Examples
```
> (interval-end '(1 2))
2
> (interval-end '(1 . 2))
2
> (interval-end #(1  2))
2
```

interval-start *interval* ⇒ *start-value* [*Function*]

Purpose
Obtain the start value of an interval.

Setf syntax
(setf (interval-start *interval*) *start-value*)

Package :gbbopen

Module :gbbopen-core

Arguments
interval An interval
start-value A number

Returns
The start value of the interval.

See also
expand-interval (page 217)
interval-start (page 232)
shift-interval (page 255)

Examples
```
> (interval-start '(1 2))
1
> (interval-start '(1 . 2))
1
> (interval-start #(1  2))
1
```

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 or a list of unit instances

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

See also
link-setf (page 234)
unlinkf (page 271)
unlinkf-all (page 272)

Example
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) .63>
```

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 or a list of unit instances

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
- 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. Then links to any additional unit instances in *unit-instance-or-instances* are added.

See also

linkf (page 233)
unlinkf (page 271)
unlinkf-all (page 272)

Example

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) .63>
```

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-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

Returns
The *slot* if it is a link slot; nil otherwise.

See also
define-unit-class (page 190)

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)
```

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 or a symbol that names a class
initargs An initialization argument list

Returns
The newly created instance of *class*.

Events
When a unit instance is created, events are signaled in the following sequence:

1. An `update-nonlink-slot-event` or `link-event` is signaled for each 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 `add-instance-to-space-instance-event` is signaled for each space instance on which the newly created unit instance is added.
3. A `create-instance-event` is signaled.

Errors
Use of an initialization argument that has not been declared as valid.

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-of` initialization argument causes that value to be used as the name of the newly created unit instance instead of the instance-name counter value associated with the unit class.

See also
define-event-class (page 185)
define-unit-class (page 190)
define-space-class (page 187)
delete-instance (page 193)
describe-instance (page 199)
instance-name-of (page 230)
make-space-instance (page 238)

Example

Create a new `hyp` unit instance:

```
> (make-instance 'hyp
     :location (list x y)
     :identity '(:car :truck :bus :motorcycle :duck-boat)
     :belief .85
     :supporting-hyps supporting-hyps)
#<hyp 119 (1835 4791) .85>
```

Note

The function **make-space-instance** provides a clear and convenient shorthand for creating space instances.

---

make-space-instance *path* `&rest` *initargs* [*Function*]
`&key` *allowed-unit-classes storage make-parents class*
⇒ *space-instance*

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`)
storage	A storage specification (see below; default is `(t t unstructured)` or `nil` if *allowed-unit-classes* is `nil`)
dimensions	A list of dimension name, dimension type pairs (default 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`)

Returns

The created space instance.

Events

When a space instance is created, events are signaled in the following sequence:

1. An `update-nonlink-slot-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 `add-instance-to-space-instance-event` is signaled for each space instance on which the newly created space instance is added.
3. A `create-instance-event` is signaled.

Detailed syntax

allowed-unit-classes ::= *unit-classes-specifier* | `nil`
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` [`:test` *hashed-test*]
dimension-buckets-specification ::= `(`*start-value end-value bucket-width*`)`
hashed-test ::= `eq` | `eql` | `equal` | `equalp`

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.

See also

allowed-unit-classes (page 180)
define-space-class (page 187)
delete-all-space-instances (page 194)
delete-space-instance (page 195)
describe-instance (page 199)
describe-space-instance (page 200)
make-instance (page 236)
space-instance-children (page 257)
space-instance-dimensions (page 258)
space-instance-parent (page 259)

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 dimensional values between 0–10,000 in the `x` and `y` dimensions:

```
> (make-space-instance '(bb hyps)
     :dimensions (unit-class-dimensions '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 using `eq` to retrive match candidates via `classification` dimensional values:

```
> (make-space-instance '(bb hyps)
     :dimensions (unit-class-dimensions '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 :test eq)))
#<standard-space-instance (bb hyps)>
```

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 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`

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 183)
clear-space-instances (page 184)
do-instances-of-class (page 206)
find-instances-of-class (page 223)
map-instances-on-space-instances (page 242)
map-sorted-instances-of-class (page 244)

Examples
Delete all unit instances of the class `hyp`:

```
(map-instances-of-class #'delete-instance 'hyp)
```

Delete all unit instances of the class `hyp` and instances of subclasses of `hyp`:

```
(map-instances-of-class #'delete-instance '(hyp :plus-subclasses))
```

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.

---

map-instances-of-class

map-instances-on-space-instances *function unit-classes-specifier space-instances* `&key` *pattern filter-before filter-after* [*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 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

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`

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-pattern-op
ordered-dimension-pattern-op ::= `<` | `<=` | `>=` | `>` | `=` | `/=` |
`within` | `covers` | `overlaps` | `starts` | `ends`
enumerated-dimension-pattern-op ::= `eq` | `eql` | `equal` | `equalp`
boolean-dimension-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 72) pattern operators are also supported, for example: `=&`, `=$&`, `=$`, `=$$`, and `=$$$` and `within&`, `within$&`, `within$`, `within$$`, and `within$$$`.

See also

do-instances-of-class (page 206)
do-instances-on-space-instances (page 208)
find-instances (page 221)
find-instances-of-class (page 223)
map-instances-of-class (page 240)
with-find-stats (page 276)
Declared numerics (page 72)

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 of less than 0.5:

```
(map-instances-on-space-instances
   #'delete-instance
   'hyp '(bb probable-hyps) :pattern '(< belief .5))
```

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 of one argument
unit-classes-specifier	An extended unit-classes specification (see below)
predicate	A function of two arguments that returns a generalized boolean
key	A function 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`

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 210)
map-instances-of-class (page 240)

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)
```

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 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 211)
find-space-instances (page 225)

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 +))
```

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`

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`

See also
add-event-function (page 177)
remove-event-function (page 247)

Examples
Remove all event functions associated with a `create-instance-event` on a `hyp` unit instance:

```
(remove-all-event-functions 'create-instance-event 'hyp)
```

Remove all event functions associated with a `create-instance-event` on a `hyp` unit instance or its subclasses:

```
(remove-all-event-functions 'create-instance-event '(hyp :plus-subclasses))
```

Note
Unit-instance-specific event functions are not yet implemented in GBBopen.

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
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`

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`

See also
add-event-function (page 177)
remove-all-event-functions (page 246)

Examples
Remove the event function evfn-printv from the set of functions to be invoked when `create-instance-event` is signalled on a `hyp` unit instance:

```
(remove-event-function 'evfn-printv 'create-instance-event 'hyp)
```

Remove the event function evfn-printv from the set of functions to be invoked when `create-instance-event` is signalled on a `hyp` unit instance or its subclasses:

```
(remove-event-function 'evfn-printv 'create-instance-event '(hyp
:plus-subclasses))
```

Note
Unit-instance-specific event functions are not yet implemented in GBBopen.

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 `remove-instance-from-space-instance-event` is signaled.

See also

add-instance-to-space-instance (page 179)

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) .68>
```

or

```
> (remove-instance-from-space-instance incorrect-hyp '(bb hyps))
#<hyp 311 (896 388) .68>
```

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 276)

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) .91>
 #<hyp 331 (1835 8419) .88>)
> (report-find-stats)
;; No find/map statistics are available.
```

reset-gbbopen `&key` *disable-events retain-classes retain-event-functions retain-event-printing* [*Function*]

Purpose
Delete all unit instances, space instances, and more.

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
disable-events	A generalized boolean (default is `t`)
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`)

Events
If *disable-events* is `nil`, the following events may be signaled as unit instances and space instances are deleted:

- `unlink-event`
- `remove-instance-from-space-instance-event`
- `delete-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 **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 1. **Reset-gbbopen** does not undefine any class definitions, functions, methods, etc.

See also

delete-instance (page 193)
delete-all-space-instances (page 194)
delete-space-instance (page 195)

Examples
Delete all unit instances and space instances, with event-signaling disabled:

```
(reset-gbbopen)
```

Delete all unit instances and space instances, except for KS unit instances, with event-signaling disabled:

```
(reset-gbbopen :retain-classes '((ks :plus-subclasses)))
```

As above, but also retain all event functions and event printing:

```
  (reset-gbbopen :retain-classes '((ks :plus-subclasses))
 :retain-event-functions 't
 :retain-event-printing 't)
```

Note

This is the only GBBopen function that disables 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-unit-class *unit-class-or-name* [*Generic Function*]

Purpose
Resets the unit-class instance-name counter of *unit-class* to 1.

Method signatures
reset-unit-class (*unit-class-name* `symbol`)

reset-unit-class (*unit-class-spec* `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 is reset to 0, but only if the unit class is not abstract and if there are no existing instances of that class. If instances do exist, a warning is issued.

Example
Reset the instance-name counters of all unit classes to 1:

```
> (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.

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`

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`

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 197)
disable-event-printing (page 204)
enable-event-printing (page 214)
suspend-event-printing (page 268)

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

shift-interval *interval amount* ⇒ *new-interval* [*Function*]

Purpose
Shift an interval by *amount*.

Package :gbbopen

Module :gbbopen-core

Arguments
interval An interval
amount A number

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
expand-interval (page 217)
interval-end (page 231)
interval-start (page 232)

Examples
```
> (shift-interval '(2 5) 2)
(4 7)
> (shift-interval '(2 . 5) -1)
(1 . 4)
> (shift-interval #(2 5) .5)
#(2.5 5.5)
```

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-slot-definition*

See also
define-event-class (page 185)
with-events-disabled (page 274)
with-events-enabled (page 275)

Example
```
(signal-event 'my-event :my-event-arg1 3)
```

space-instance-children *space-instance* ⇒ *space-instances* [*Generic Function*]

Purpose
Return the child space instances of a space instance.

Method signatures
space-instance-children (*space-instance* `root-space-instance`) ⇒ *space-instances*

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
space-instance A space instance

Returns
A list of the child space instances.

See also
make-space-instance (page 238)
space-instance-parent (page 259)

Example
Return the child space instances of the `(bb)` space instance:

```
> (space-instance-children (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.

space-instance-dimensions *space-instance* ⇒ *dimension-list* [*Generic Function*]

Purpose
Return the dimension specifications of a space instance.

Method signatures
space-instance-dimensions (*space-instance* `standard-space-instance`) ⇒ *dimension-list*

Package `:gbbopen`

Module `:gbbopen-core`

Arguments
space-instance A space instance

Returns
A list of dimension name, dimension type pairs.

See also
make-space-instance (page 238)
unit-class-dimensions (page 270)

Example
Return the dimensions of the `(bb hyps)` space instance:

```
> (space-instance-dimensions (find-space-instance-by-path '(bb hyps))
((x :ordered) (y :ordered) (belief :ordered) (classification :enumerated))
```

Note
The returned list of dimension specifications should not be destructively altered.

space-instance-parent *space-instance* ⇒ *space-instances* [*Generic Function*]

Purpose
Return the parent space instance of a space instance.

Method signatures
space-instance-parent (*space-instance* `standard-space-instance`) ⇒ *space-instance*

Package
`:gbbopen`

Module
`:gbbopen-core`

Arguments
space-instance A space instance

Returns
The parent space instance or the special `root-space-instance`, if *space-instance* does not have a parent.

Description
If *space-instance* does not have a parent, the `root-space-instance` is returned. The function **space-instance-children** can be called on the returned `root-space-instance` to obtain siblings of a parent-less *space-instance*.

See also
make-space-instance (page 238)
space-instance-children (page 257)

Examples
Return the parent space instances of the `(bb hyp)` space instance:

```
> (space-instance-parent (find-space-instance-by-path '(bb hyp))
#<standard-space-instance (bb)>
```

Return the parent space instances (the `root-space-instance`) of the `(bb)` space instance:

```
> (space-instance-parent (find-space-instance-by-path '(bb))
#<root-space-instance root-space-instance>
```

Return the siblings of the `(bb)` space instance:

```
> (space-instance-children
    (space-instance-parent (find-space-instance-by-path '(bb))
(#<standard-space-instance (bb)>
 #<standard-space-instance (control-shell)>)
```

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

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-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 185)
standard-event-instance (page 262)

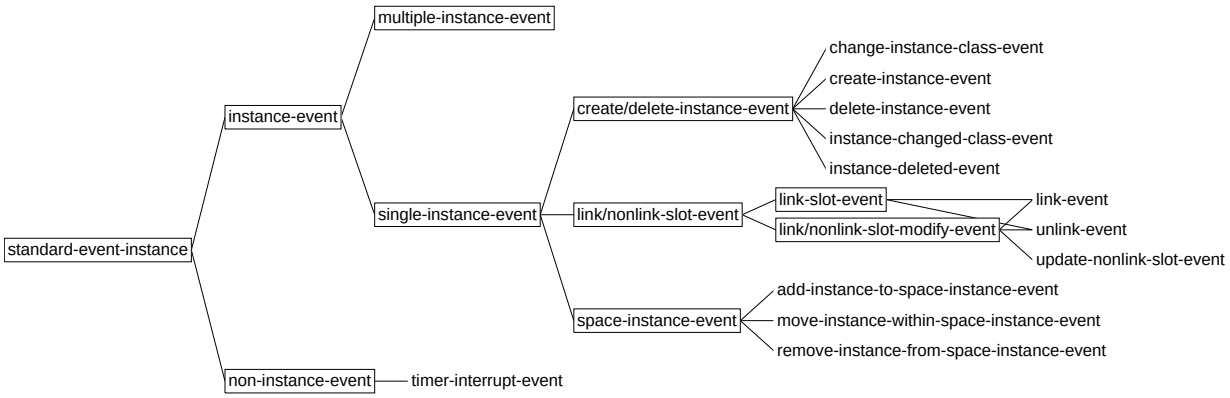
Package :gbbopen

Module :gbbopen-core

Description

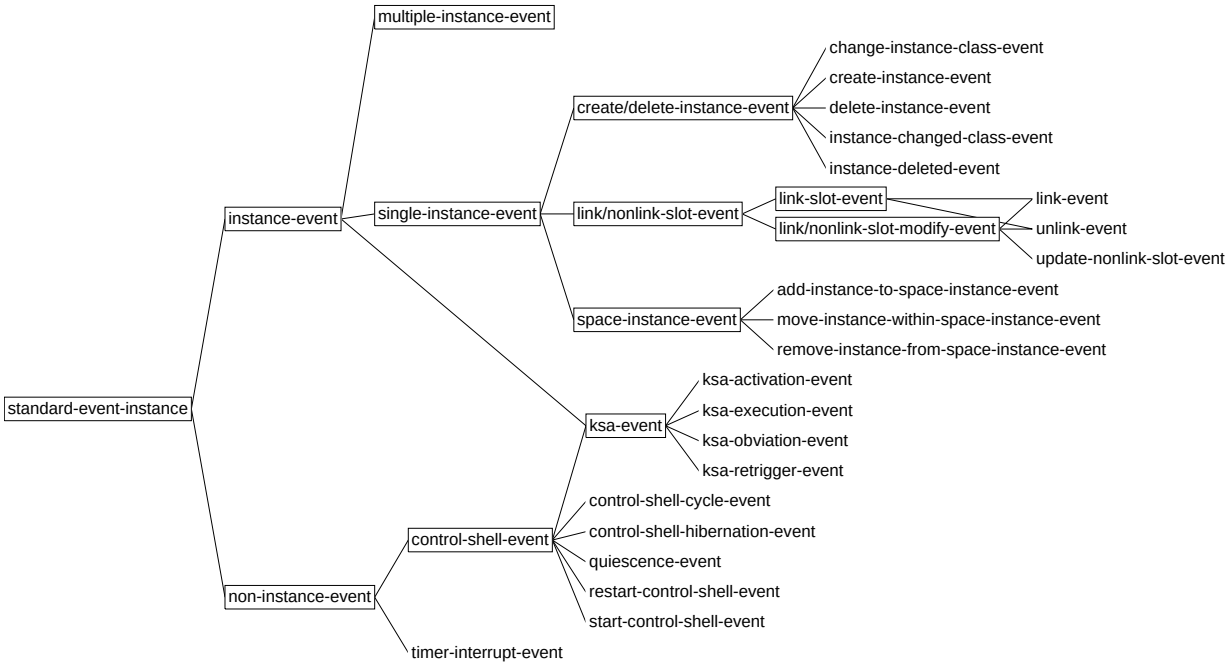
The class **standard-event-instance** is an instance of **standard-event-class** and is a superclass of every event class that is an instance of **standard-event-class** except itself. It is a subclass of **standard-gbbopen-instance**.

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 signalled.

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 signalled by the Agenda Shell. Again classes shown within rectangles are abstract classes that cannot be signalled.

See also

print-instance-slots (page 48)
standard-gbbopen-instance (page 60)
standard-event-class (page 261)

---

standard-event-instance

standard-space-class [*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 265)
standard-unit-class (page 266)

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

print-instance-slots (page 48)
standard-space-class (page 264)
standard-unit-instance (page 267)

standard-unit-class [*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 190)
standard-unit-instance (page 267)
standard-space-class (page 264)

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

print-instance-slots (page 48)
standard-gbbopen-instance (page 60)
standard-space-instance (page 265)
standard-unit-class (page 266)

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`

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`

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 197)
disable-event-printing (page 204)
enable-event-printing (page 214)
resume-event-printing (page 253)

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.

---

unit-class-dimensions *unit-classes-specifier* ⇒ *dimension-list* [*Generic Function*]

Purpose

Return the dimension specifications for instances of a unit class.

Method signatures

unit-class-dimensions (*unit-classes-specifier* `cons`) ⇒ *dimension-list*

unit-class-dimensions (*unit-class* `standard-unit-class`) ⇒ *dimension-list*

Package

`:gbbopen`

Module

`:gbbopen-core`

Arguments

unit-classes-specifier An extended unit-classes specification or a list of extended unit-classes specifications

Returns

A list of dimension name, dimension type pairs.

See also

define-unit-class (page 190)

space-instance-dimensions (page 258)

Example

Return the dimensions defined for instances of unit class `hyp`:

```
> (unit-class-dimensions 'hyp)
((x :ordered) (y :ordered) (belief :ordered) (classification :enumerated))
```

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 or a list of unit instances

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
linkf (page 233)
link-setf (page 234)
unlinkf-all (page 272)

Example
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) .63>
```

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 233)
link-setf (page 234)
unlinkf (page 271)

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))
```

with-changing-dimension-values (*unit-instance* *dimension-name**) *declaration** *form** ⇒ *result** [*Macro*]

Purpose

Inform GBBopen that the dimensional values of a unit instance will potentially 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
forms	An implicit **progn** of forms to be evaluated

Returns

The values returned by evaluating the last *form*.

Description

The indexing for *unit-instance* is updated following the evaluation of the last *form*. Any retrieval operations performed during the evaluation of *forms* may operate with indexes as they existed before evaluation of the first *form*; therefore, retrievals should not be included within the scope of **with-changing-dimension-values**. Retrievals performed by separate threads also should be synchronized with **with-changing-dimension-values**.

If *dimension-names* are specified, only the indexes for those dimensions of *unit-instance* will be updated. 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*.

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))
```

Notify GBBopen that some dimension values of `hyp` might be changed:

```
> (with-changing-dimension-values (hyp)
    (setf (location-of hyp) '(36 52))
    (incf (belief-of hyp) 0.05))
```

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

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

See also
signal-event (page 256)
with-events-enabled (page 275)

Example
Create a `hyp` without signaling any events:

```
> (with-events-disabled ()
    (make-instance 'hyp
      :location (list x y)
      :identity '(:car :truck :bus :motorcycle :duck-boat)
      :belief .85
      :supporting-hyps supporting-hyps))
#<hyp 119 (1835 4791) .85>
```

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

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

See also
signal-event (page 256)
with-events-disabled (page 274)

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)
                 :identity '(:car :truck :bus :motorcycle :duck-boat)
                 :belief .85)))
      (with-events-enabled ()
        (linkf (supporting-hyps-of hyp) supporting-hyps))
      hyp))
#<hyp 119 (1835 4791) .85>
```

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

Returns

The values returned by evaluating the last *form*.

See also

find-instances (page 221)
map-instances-on-space-instances (page 242)
report-find-stats (page 249)
without-find-stats (page 277)

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 119 (1835 4791) .85>
 #<hyp 233 (1835 4791) .89>)
```

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

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

See also
with-find-stats (page 276)

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) .91>
 #<hyp 331 (1835 8419) .88>)
```

5 Agenda Control Shell

The Agenda Shell module, `:agenda-shell`, provides a responsive, agenda-based control shell.

abort-ks-execution <no arguments> [*Function*]

Purpose

Abort the currently executing KSA.

Package

`:agenda-shell`

Module

`:agenda-shell`

See also

exit-control-shell (page 297)

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

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 300)

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, 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

Returns
The list of trigger unit instances

See also
sole-trigger-instance-of (page 308)

Example
Return the trigger unit instances of a KSA:

```
> (collect-trigger-instances ksa)
(#<hyp 119 (1835 4791) .85>
 #<hyp 233 (1835 4791) .89>)
```

control-shell-running-p <no arguments> ⇒ *boolean* [*Function*]

Purpose
Return a value indicating whether a control shell is running.

Package :agenda-shell

Module :agenda-shell

Returns
True if the control shell is running; `nil` otherwise.

See also
start-control-shell (page 310)
restart-control-shell (page 306)

Example
See if the control shell is running:

```
> (control-shell-running-p)
nil
```

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 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 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 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 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 of two arguments (the KSA unit instance and the event object) or `nil` (default is `nil`)
revalidation-predicate	A function 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)

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`

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`

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 typicial 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 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 287)
define-ksa-class (page 290)
describe-ks (page 293)
ensure-ks (page 294)
ks (page 299)
ks-enabled-p (page 301)
standard-event-instance (page 262)
undefine-ks (page 315)

Examples

Define an initial KS that is triggered when the control shell is started:

```
(define-ks inital
   :trigger-events ((start-control-shell-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 ((create-instance-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-class *ks-class-name* (*{superclass-name}**) [*documentation*] [*Macro*]
(*{slot-specifier}**) *{class-option}** ⇒ *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*

documentation A documentation string

slot-specifiers See below

class-options See below

Returns
The newly defined 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
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 *dimensional-value-spec**) |
(:documentation *string*) |
(:export-class-name *boolean*) |
(:export-accessors *boolean*) |
(: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-comparision-test *instance-name-comparision-test*) |
(:metaclass *class-name*)
initial-space-instance-specifier ::= {*space-instance-path*+ | *function*}
dimensional-value-specifier ::= *incomposite-dv-specifier* | *composite-dv-specifier*
incomposite-dv-specifier ::= (*dimension-name dimension-value-type dimension-value-place*)
composite-dv-specifier ::= (*dimension-name dimension-value-type*
composite-type dimension-value-place)
composite-type ::= :set | :sequence |
{:ascending-series *ordering-dimension-name*} |
{:descending-series *ordering-dimension-name*}
dimension-value-type ::= :point | :interval | :mixed | :element | :boolean
dimension-value-place ::= {*slot-name* [*slot-name*]} | {*function* [*slot-name*]}
dimensional-value-specifier ::= (*dimension-name dimension-value-type dimension-value-place*)
dimension-value-type ::= :point | :interval | :mixed | :element | :boolean
dimension-value-place ::= *slot-name* | *slot-name slot-name* | {*function* [*slot-name*]}
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
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*, 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 dimensional index

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. 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 initializtion value associated with the metaclass is used.

See also

define-ks (page 284)
standard-unit-class (page 266)
with-generate-accessors-format (page 67)

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 lock-mixin (ks my-mixin)
    ((lock :initform (make-lock :name "KS Lock"))))
#<standard-unit-class lock-mixin>
> (define-ks-class ks-with-lock (ks lock-mixin)
    ())
#<standard-unit-class ks-with-lock>
```

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

documentation A documentation string

slot-specifiers See below

class-options See below

Returns
The newly defined 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 *dimensional-value-spec**) |
(:documentation *string*) |
(:export-class-name *boolean*) |
(:export-accessors *boolean*) |
(: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-comparision-test *instance-name-comparision-test*) |
(:metaclass *class-name*)
initial-space-instance-specifier ::= {*space-instance-path*+ | *function*}
dimensional-value-specifier ::= *incomposite-dv-specifier* | *composite-dv-specifier*
incomposite-dv-specifier ::= (*dimension-name dimension-value-type dimension-value-place*)
composite-dv-specifier ::= (*dimension-name dimension-value-type*
composite-type dimension-value-place)
composite-type ::= :set | :sequence |
{:ascending-series *ordering-dimension-name*} |
{:descending-series *ordering-dimension-name*}
dimension-value-type ::= :point | :interval | :mixed | :element | :boolean
dimension-value-place ::= {*slot-name* [*slot-name*]} | {*function* [*slot-name*]}
dimensional-value-specifier ::= (*dimension-name dimension-value-type dimension-value-place*)
dimension-value-type ::= :point | :interval | :mixed | :element | :boolean
dimension-value-place ::= *slot-name* | *slot-name slot-name* | {*function* [*slot-name*]}
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
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*, 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 dimensional index

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. 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 initializtion value associated with the metaclass is used.

See also

define-ks (page 284)
standard-ksa-class (page 309)
with-generate-accessors-format (page 67)

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 lock-mixin (ks my-mixin)
    ((lock :initform (make-lock :name "KS Lock"))))
#<standard-unit-class lock-mixin>
> (define-ksa-class ksa-with-lock (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 284)
ks (page 299)

Example
```
> (describe-ks 'start-control-shell-ks)

KS: start-control-shell-ks
  Trigger events:        ((start-control-shell-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

Programatically 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 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 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 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 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 of two arguments (the KSA unit instance and the event object) or `nil` (default is `nil`)
revalidation-predicate	A function of one argument (the KSA unit instance) that returns a generalized boolean or `nil` (default is `nil`)
trigger-events	An *event-specification* (see below)

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`

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`

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 284)
ks (page 299)
ks-enabled-p (page 301)
undefine-ks (page 315)

Example

Define an initial KS that is triggered when the control shell is started:

```
(ensure-ks 'inital
   :trigger-events '((start-control-shell-event))
   :execution-function #'initial-ks-function)
```

execution-cycle-of *ksa* ⇒ *cycle-number* [*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

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 300)

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 280)
control-shell-running-p (page 283)
restart-control-shell (page 306)
start-control-shell (page 310)

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* [*Function*]

Purpose
Return a KS unit instance given its name

Package
`:agenda-shell`

Module
`:agenda-shell`

Arguments
ks-name A symbol naming the KS.

Returns
The KS unit instance named *ks-name* or `nil`, if none has been defined.

See also
define-ks (page 284)
ks (page 299)

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 284)
ksa (page 300)

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 299)

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*)

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 284)
ks (page 299)
undefine-ks (page 315)

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

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 299)
ksa (page 300)

Example
Return the KS of a KSA:

```
> (ks-of ksa)
#<ks start-control-shell-ks>
```

obviation-cycle-of *ksa* ⇒ *cycle-number* [*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

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 300)

Example
Return the obviation cycle of `ksa`:

```
> (obviation-cycle-of ksa)
1211
```

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*)

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 284)
ks (page 299)
ksa (page 300)

Example
Return the rating of a KSA:

```
> (rating-of ksa)
58
```

Note
The rating of a pending KSA can be changed by using **setf** or related macros with this accessor.

restart-control-shell <no arguments>⇒ *value** [*Function*]

Purpose
Restart the agenda shell.

Package :agenda-shell

Module :agenda-shell

Returns
One of the following values:

- `:quiescense`—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 `restart-control-shell-event` is signaled.

See also

abort-ks-execution (page 280)
control-shell-running-p (page 283)
exit-control-shell (page 297)
run-polling-functions (page 142)
start-control-shell (page 310)

Example
Restart the Agenda Shell (in this case, without any KSs defined):

```
> (restart-control-shell)
;; Control shell restarting after cycle 2
;; No executable KSAs remain, exiting control shell
;; Control shell 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.

sole-trigger-event-of *ksa* ⇒ *event* [*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

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 308)

Example
Return the (sole) trigger event of a KSA:

```
> (sole-trigger-event-of ksa)
#<create-instance-event hyp>
```

sole-trigger-instance-of *source* ⇒ *trigger-instance* [*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

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 282)
sole-trigger-event-of (page 307)

Example
Return the (sole) trigger unit instance of a KSA:

```
> (sole-trigger-instance-of ksa)
#<hyp 119 (1835 4791) .85>
```

standard-ksa-class [*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 290)
print-instance-slots (page 48)
standard-unit-class (page 266)

start-control-shell `&rest` *initargs* ⇒ *value** [*Function*]

Purpose
Start the Agenda Shell.

Package `:agenda-shell`

Module `:agenda-shell`

Arguments
initargs An initialization argument list (see below)

Returns
One of the following values:

- `:quiescense`—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 `start-control-shell-event` is signaled.

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`)
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 signalled
`: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 137)) 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 283)
exit-control-shell (page 297)
run-polling-functions (page 142)
restart-control-shell (page 306)

Examples

Start the Agenda Shell (in this case, without any KSs defined):

```
> (start-control-shell)
;; Control shell started
;; No executable KSAs remain, exiting control shell
;; Control shell 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 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 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 Function*]

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

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 284)
ks (page 299)
ksa (page 300)

Example
Return the events that triggered a KSA:

```
> (trigger-events-of ksa)
(#<create-instance-event #<hyp 233 (1835 4791) .89>)
```

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 remainding initialization arguments are ignored

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 284)
ks-enabled-p (page 301)

Examples
Undefine the KS named `initial`. The following forms are all equivalent:

```
> (undefine-ks inital
     :trigger-events '((start-control-shell-event))
     :execution-function #'initial-ks-function)
#<ks initial [Deleted]>

> (undefine-ks inital)
#<ks initial [Deleted]>

> (delete-instance
     (find-instance-by-name 'initial '(ks :plus-subclasses)))
#<ks initial [Deleted]>
```

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.

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`).

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 option An option that refers to a class as a whole or to all the slots of the class.

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. 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 instance, space instance, or retrieval pattern; one of `:ordered`, `:enumerated`, or `:boolean`.

dimension value The value that is used to position a unit instance on a dimension of 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.

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.

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.

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.

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.

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.

link A bi-directional pair of pointers between two unit instances. Link operators maintain bi-directional consistency of link pointers.

link slot A slot designated for the outgoing pointers of links associated with that slot.

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`.

obviated KS activation An unexecuted KS activation that has been deemed unnecessary and will therefore never be executed.

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.

ordered queue A doubly linked, ordered queue. A GBBopen queue is headed by an object that is a subclass of **ordered-queue**.

passive socket A socket that is used to accept a connection initiation to a specific service port.

path expression A regular expression representing one or more space-instance paths.

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.

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.

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 Mini Module system (see page 11).

retrieval pattern A list argument to **filter-instances**, **find-instances**, and **map-instances-on-space-instances** specifying the dimensional requirements for selecting unit instances to be returned.

root directory A fixed anchor directory for a tree of relative directory definitions. Part of the Mini Module system (see page 11).

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.

slot A component of an object that can store a value.

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.

subclasses The classes that inherit from a class.

subevents The classes that inherit from an event class.

thread A thread in a multi-threaded Common Lisp implementation or a Lisp process in a Common Lisp that provides multiprocessing.

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 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