GNU Guix topmost-level modules

version 1.4.0rc2, updated updated

This manual is for GNU Guix topmost-level modules (version 1.4.0rc2, updated updated)

Short Contents

1 (gnu artwork)

1.1 Overview

Common place for the definition of the Guix artwork repository.

1.2 Usage

%artwork-repository [Variable]

2 (gnu bootloader)

2.1 Overview

2.2 Usage

`menu-entry` [Special Form]

`menu-entry?` [Special Form]

`menu-entry-label` [Special Form]

`menu-entry-device` [Special Form]

`menu-entry-linux` [Special Form]

`menu-entry-linux-arguments` [Special Form]

`menu-entry-initrd` [Special Form]

`menu-entry-device-mount-point` [Special Form]

`menu-entry-multiboot-kernel` [Special Form]

`menu-entry-multiboot-arguments` [Special Form]

`menu-entry-multiboot-modules` [Special Form]

`menu-entry-chain-loader` [Special Form]

`menu-entry->sexp` *entry* [Function]
Return ENTRY serialized as an sexp.

`sexp->menu-entry` *sexp* [Function]
Turn SEXP, an sexp as returned by 'menu-entry->sexp', into a <menu-entry> record.

`bootloader` [Special Form]

`bootloader?` [Special Form]

`bootloader-name` [Special Form]

`bootloader-package` [Special Form]

`bootloader-installer` [Special Form]

`bootloader-disk-image-installer` [Special Form]

`bootloader-configuration-file` [Special Form]

`bootloader-configuration-file-generator` [Special Form]

`bootloader-configuration` [Special Form]

`bootloader-configuration?` [Special Form]

`bootloader-configuration-bootloader` [Special Form]

`bootloader-configuration-target` [Special Form]

`bootloader-configuration-targets` *config* [Function]

`bootloader-configuration-menu-entries` [Special Form]

`bootloader-configuration-default-entry` [Special Form]

`bootloader-configuration-timeout` [Special Form]

`bootloader-configuration-keyboard-layout` [Special Form]

`bootloader-configuration-theme` [Special Form]

`bootloader-configuration-terminal-outputs` [Special Form]

`bootloader-configuration-terminal-inputs` [Special Form]

`bootloader-configuration-serial-unit` [Special Form]

`bootloader-configuration-serial-speed` [Special Form]

`bootloader-configuration-device-tree-support?` [Special Form]

`%bootloaders` [Variable]

`lookup-bootloader-by-name` *name* [Function]
Return the bootloader called NAME.

`efi-bootloader-chain` *files final-bootloader* [*#:hooks*] [*#:installer*] [Function]
Define a bootloader chain with FINAL-BOOTLOADER as the final bootloader and certain directories and files from the store given in the list of FILES.

FILES may contain file like objects produced by functions like plain-file, local-file, etc., or package contents produced with file-append. They will be collected inside a directory collection/ inside a generated bootloader profile, which will be passed to the INSTALLER.

If a directory name in FILES ends with '/', then the directory content instead of the directory itself will be symlinked into the collection/ directory.

The procedures in the HOOKS list can be used to further modify the bootloader profile. It is possible to pass a single function instead of a list.

If the INSTALLER argument is used, then this function will be called to install the bootloader. Otherwise the installer of the FINAL-BOOTLOADER will be called.

3 (gnu ci)

3.1 Overview

```
This file defines build jobs for Cuirass.
```

3.2 Usage

`derivation->job` *name drv* [*#:max-silent-time*] [*#:timeout*] [Function]
Return a Cuirass job called NAME and describing DRV.

MAX-SILENT-TIME and TIMEOUT are build options passed to the daemon when building the derivation.

`image->job` *store image* [*#:name*] [*#:system*] [Function]
Return the job for IMAGE on SYSTEM. If NAME is passed, use it as job name, otherwise use the IMAGE name.

`%core-packages` [Variable]

`arguments->systems` *arguments* [Function]
Return the systems list from ARGUMENTS.

`cuirass-jobs` *store arguments* [Function]
Register Cuirass jobs.

4 (gnu compression)

4.1 Overview

4.2 Usage

`compressor` [Special Form]

`compressor?` [Special Form]

`compressor-name` [Special Form]

`compressor-extension` [Special Form]

`compressor-command` [Special Form]

`%compressors` [Variable]

`lookup-compressor` *name* [Function]

Return the compressor object called NAME. Error out if it could not be found.

5 (gnu home)

5.1 Overview

5.2 Usage

`home-environment` [Special Form]

`home-environment?` [Special Form]

`this-home-environment` [Special Form]
Return the record being defined. This macro may only be used in the context of the definition of a thunked field.

`home-environment-derivation` *he* [Function]
Return a derivation that builds home environment.

`home-environment-user-services` [Special Form]

`home-environment-essential-services` [Special Form]

`home-environment-services` *he* [Function]
Return all the services of home environment.

`home-environment-location` [Special Form]

`home-environment-with-provenance` *he config-file* [Function]
Return a variant of HE that stores its own provenance information, including CONFIG-FILE, if available. This is achieved by adding an instance of HOME-PROVENANCE-SERVICE-TYPE to its services.

`home-generation-base` *file* [Function]
If FILE is a Home generation GC root such as "guix-home-42-link", return its corresponding base—e.g., "guix-home". Otherwise return #f.

This is similar to the 'generation-profile' procedure but applied to Home generations.

6 (gnu image)

6.1 Overview

6.2 Usage

`partition` [Special Form]

`partition?` [Special Form]

`partition-device` [Variable]
[unbound!]

`partition-size` [Special Form]

`partition-offset` [Special Form]

`partition-file-system` [Special Form]

`partition-file-system-options` [Special Form]

`partition-label` [Special Form]

`partition-uuid` [Special Form]

`partition-flags` [Special Form]

`partition-initializer` [Special Form]

`image` [Special Form]

`image?` [Special Form]

`image-name` [Special Form]

`image-format` [Special Form]

`image-platform` [Special Form]

`image-size` [Special Form]

`image-operating-system` [Special Form]

`image-partition-table-type` [Special Form]

`image-partitions` [Special Form]

`image-compression?` [Special Form]

`image-volatile-root?` [Special Form]

`image-shared-store?` [Special Form]

`image-shared-network?` [Special Form]

`image-substitutable?` [Special Form]

`image-type` [Special Form]

`image-type?` [Special Form]

`image-type-name` [Special Form]

`image-type-constructor` [Special Form]

`os->image` *os* [*#:type*] [Function]
Use the image constructor from TYPE, an <image-type> record to turn the given OS, an <operating-system> record into an image and return it.

`os+platform->image` *os platform* [*#:type*] [Function]
Use the image constructor from TYPE, an <image-type> record to turn the given OS, an <operating-system> record into an image targeting PLATFORM, a <platform> record and return it.

7 (gnu installer)

7.1 Overview

7.2 Usage

`installer-program` [*#:dry-run?*] [Function]
Return a file-like object that runs the given INSTALLER.

`installer-steps` [*#:dry-run?*] [Function]

`run-installer` [*#:dry-run?*] [Function]
To run the installer from Guile without building it:
```
./pre-inst-env guile -c '((@@ (gnu installer) run-installer) #:dry-run? #t)'
```
when using #:dry-run? #t, no root access is required and the LOCALE, KEYMAP, and PARTITION pages are skipped.

8 (gnu machine)

8.1 Overview

This module provides the types used to declare individual machines in a heterogeneous Guix deployment. The interface allows users to specify system configurations and the means by which resources should be provisioned on a per-host basis.

8.2 Usage

`environment-type` [Special Form]

`environment-type?` [Special Form]

`environment-type-name` [Special Form]

`environment-type-description` [Special Form]

`environment-type-location` [Special Form]

`machine` [Special Form]

`machine?` [Special Form]

`machine-operating-system` *machine* [Function]
Return the operating system of MACHINE.

`machine-environment` [Special Form]

`machine-configuration` [Special Form]

`machine-display-name` *machine* [Function]
Return the host-name identifying MACHINE.

`deploy-machine` *machine* [Function]
Monadic procedure transferring the new system's OS closure to the remote MACHINE, activating it on MACHINE and switching MACHINE to the new generation.

`roll-back-machine` *machine* [Function]
Monadic procedure rolling back to the previous system generation on MACHINE. Return the number of the generation that was current before switching and the new generation number.

`machine-remote-eval` *machine exp* [Function]
Evaluate EXP, a gexp, on MACHINE. Ensure that all the elements EXP refers to are built and deployed to MACHINE beforehand.

`&deploy-error` [Variable]

`deploy-error?` *obj* [Function]

`deploy-error-should-roll-back` *obj* [Function]

`deploy-error-captured-args` *obj* [Function]

9 (gnu packages)

9.1 Overview

General utilities for the software distribution---i.e., the modules under (gnu packages ...).

9.2 Usage

`search-patch` *file-name* [Function]
Search the patch FILE-NAME. Raise an error if not found.

`search-patches` *file-name ...* [Special Form]
Return the list of absolute file names corresponding to each FILE-NAME found in %PATCH-PATH.

`search-auxiliary-file` *file-name* [Function]
Search the auxiliary FILE-NAME. Return #f if not found.

`%patch-path` [Variable]

`%auxiliary-files-path` [Variable]

`%package-module-path` [Variable]

`%default-package-module-path` [Variable]

`cache-is-authoritative?` [Function]
Return true if the pre-computed package cache is authoritative. It is not authoritative when entries have been added via GUIX_PACKAGE_PATH or '-L' flags.

`fold-packages` *proc init* [*modules*] [*#:select?*] [Function]
Call (PROC PACKAGE RESULT) for each available package defined in one of MODULES that matches SELECT?, using INIT as the initial value of RESULT. It is guaranteed to never traverse the same package twice.

`fold-available-packages` *proc init* [Function]
Fold PROC over the list of available packages. For each available package, PROC is called along these lines:

```
(PROC NAME VERSION RESULT
      #:outputs OUTPUTS
      #:location LOCATION
      ...)
```

PROC can use #:allow-other-keys to ignore the bits it's not interested in. When a package cache is available, this procedure does not actually load any package module.

`find-newest-available-packages` [Special Form]

`find-packages-by-name` *name* [*version*] [Function]
Return the list of packages with the given NAME. If VERSION is not #f, then only return packages whose version is prefixed by VERSION, sorted in decreasing version order.

`find-package-locations` *name* [*version*] [Function]
Return a list of version/location pairs corresponding to each package matching NAME and VERSION.

`find-best-packages-by-name` *name version* [Function]
If version is #f, return the list of packages named NAME with the highest version numbers; otherwise, return the list of packages named NAME and at VERSION.

`specification->package` *spec* [Function]
Return a package matching SPEC. SPEC may be a package name, or a package name followed by an at-sign and a version number. If the version number is not present, return the preferred newest version.

`specification->package+output` *spec* [*output*] [Function]
```
Return the package and output specified by SPEC, or #f and #f; SPEC may
optionally contain a version number and an output name, as in these examples:

  guile
  guile@@2.0.9
  guile:debug
  guile@@2.0.9:debug

If SPEC does not specify a version number, return the preferred newest
version; if SPEC does not specify an output, return OUTPUT.

When OUTPUT is false and SPEC does not specify any output, return #f as the
output.
```

`specification->location` *spec* [Function]
```
Return the location of the highest-numbered package matching SPEC, a
specification such as "guile@@2" or "emacs".
```

`specifications->manifest` *specs* [Function]
```
Given SPECS, a list of specifications such as "emacs@@25.2" or
"guile:debug", return a profile manifest.
```

`specifications->packages` *specs* [Function]
```
Given SPECS, a list of specifications such as "emacs@@25.2" or
"guile:debug", return a list of package/output tuples.
```

`package-unique-version-prefix` *name version* [Function]
Search among all the versions of package NAME that are available, and return the shortest unambiguous version prefix to designate VERSION. If only one version of the package is available, return the empty string.

`generate-package-cache` *directory* [Function]

Generate under DIRECTORY a cache of all the available packages.

The primary purpose of the cache is to speed up package lookup by name such that we don't have to traverse and load all the package modules, thereby also reducing the memory footprint.

10 (gnu)

10.1 Overview

This composite module re-exports core parts the (gnu ...) public modules.

10.2 Usage

`use-package-modules` *module ...* [Special Form]

`use-service-modules` *module ...* [Special Form]

`use-system-modules` *module ...* [Special Form]

11 (gnu services)

11.1 Overview

11.2 Usage

`service-extension` [Special Form]

`service-extension?` [Special Form]

`service-extension-target` [Special Form]

`service-extension-compute` [Special Form]

`service-type` [Special Form]

`service-type?` [Special Form]

`service-type-name` [Special Form]

`service-type-extensions` [Special Form]

`service-type-compose` [Special Form]

`service-type-extend` [Special Form]

`service-type-default-value` [Special Form]

`service-type-description` [Special Form]

`service-type-location` [Special Form]

`%service-type-path` [Variable]

`fold-service-types` *proc seed* [*modules*] [Function]
For each service type exported by one of MODULES, call (PROC RESULT). SEED is used as the initial value of RESULT.

`lookup-service-types` *name* [Function]
Return the list of services with the given NAME (a symbol).

`service` *type value* [Special Form]
`service` *type* [Special Form]
Return a service instance of TYPE. The service value is VALUE or, if omitted, TYPE's default value.

`service?` [Special Form]

`service-kind` [Special Form]

`service-value` [Special Form]

`service-parameters` *s* [Function]

`simple-service` *name target value* [Function]
Return a service that extends TARGET with VALUE. This works by creating a singleton service type NAME, of which the returned service is an instance.

`modify-services` *services clauses ...* [Special Form]

```
Modify the services listed in SERVICES according to CLAUSES and return
the resulting list of services.  Each clause must have the form:

  (TYPE VARIABLE => BODY)

where TYPE is a service type, such as 'guix-service-type', and VARIABLE is an
identifier that is bound within BODY to the value of the service of that
TYPE.

Clauses can also remove services of a given type:

  (delete TYPE)

Consider this example:

  (modify-services %base-services
    (guix-service-type config =>
                       (guix-configuration
                        (inherit config)
                        (use-substitutes? #f)
                        (extra-options '("--gc-keep-derivations"))))
    (mingetty-service-type config =>
                           (mingetty-configuration
                            (inherit config)
                            (motd (plain-file "motd" "Hi there!"))))
    (delete udev-service-type))

It changes the configuration of the GUIX-SERVICE-TYPE instance, and that of
all the MINGETTY-SERVICE-TYPE instances, and it deletes instances of the
UDEV-SERVICE-TYPE.

This is a shorthand for (filter-map (lambda (svc) ...) %base-services).
```

`service-back-edges` *services* [Function]

Return a procedure that, when passed a <service>, returns the list of <service> objects that depend on it.

`instantiate-missing-services` *services* [Function]

Return SERVICES, a list, augmented with any services targeted by extensions and missing from SERVICES. Only service types with a default value can be instantiated; other missing services lead to a '&missing-target-service-error'.

`fold-services` *services* [*#:target-type*] [Function]

Fold SERVICES by propagating their extensions down to the root of type TARGET-TYPE; return the root service adjusted accordingly.

`service-error?` *obj* [Function]

`missing-value-service-error?` *obj* [Function]

`missing-value-service-error-type` *obj* [Function]

`missing-value-service-error-location` *obj* [Function]

`missing-target-service-error?` *obj* [Function]

`missing-target-service-error-service` *obj* [Function]

`missing-target-service-error-target-type` *obj* [Function]

`ambiguous-target-service-error?` *obj* [Function]

`ambiguous-target-service-error-service` *obj* [Function]

`ambiguous-target-service-error-target-type` *obj* [Function]

`system-service-type` [Variable]

`provenance-service-type` [Variable]

`sexp->system-provenance` *sexp* [Function]
Parse SEXP, an s-expression read from /run/current-system/provenance or similar, and return two values: the list of channels listed therein, and the OS configuration file or #f.

`system-provenance` *system* [Function]
Given SYSTEM, the file name of a system generation, return two values: the list of channels SYSTEM is built from, and its configuration file. If that information is missing, return the empty list (for channels) and possibly #false (for the configuration file).

`boot-service-type` [Variable]

`cleanup-service-type` [Variable]

`activation-service-type` [Variable]

`activation-service->script` *service* [Function]
Return as a monadic value the activation script for SERVICE, a service of ACTIVATION-SCRIPT-TYPE.

`%linux-bare-metal-service` [Variable]

`%hurd-rc-script` [Variable]

`%hurd-startup-service` [Variable]

`special-files-service-type` [Variable]

`extra-special-file` *file target* [Function]
```
Use TARGET as the "special file" FILE.  For example, TARGET might be
  (file-append coreutils "/bin/env")
and FILE could be "/usr/bin/env".
```

`etc-service-type` [Variable]

`etc-directory` *service* [Function]
Return the directory for SERVICE, a service of type ETC-SERVICE-TYPE.

`setuid-program-service-type` [Variable]

`profile-service-type` [Variable]

`firmware-service-type` [Variable]

`gc-root-service-type` [Variable]

`linux-builder-service-type` [Variable]

`linux-builder-configuration` [Special Form]

`linux-builder-configuration?` [Special Form]

`linux-builder-configuration-kernel` [Special Form]

`linux-builder-configuration-modules` [Special Form]

`linux-loadable-module-service-type` [Variable]

`%boot-service` [Variable]

`%activation-service` [Variable]

`etc-service` *files* [Function]
Return a new service of ETC-SERVICE-TYPE that populates /etc with FILES. FILES must be a list of name/file-like object pairs.

`delete` _ _ [_] [Function]

```
- Scheme Procedure: delete item lst
     Return a newly-created copy of LST with elements `equal?' to ITEM
     removed.  This procedure mirrors `member': `delete' compares
     elements of LST against ITEM with `equal?'.
```

12 (gnu system)

12.1 Overview

This module supports whole-system configuration.

12.2 Usage

`operating-system` [Special Form]

`operating-system?` [Special Form]

`this-operating-system` [Special Form]
Return the record being defined. This macro may only be used in the context of the definition of a thunked field.

`operating-system-bootloader` [Special Form]

`operating-system-services` *os* [Function]
Return all the services of OS, including "essential" services.

`operating-system-essential-services` [Special Form]

`operating-system-default-essential-services` *os* [Function]
Return the list of essential services for OS. These are special services that implement part of what's declared in OS are responsible for low-level bookkeeping.

`operating-system-user-services` [Special Form]

`operating-system-packages` [Special Form]

`operating-system-host-name` [Special Form]

`operating-system-hosts-file` [Special Form]

`operating-system-hurd` [Special Form]

`operating-system-kernel` [Special Form]

`operating-system-kernel-file` *os* [Function]
Return an object representing the absolute file name of the kernel image of OS.

`operating-system-kernel-arguments` *os root-device* [*#:version*] [Function]
Return all the kernel arguments, including the ones not specified directly by the user. VERSION should match that of the target <boot-parameters> record object that will contain the kernel parameters.

`operating-system-label` [Special Form]

`operating-system-default-label` *os* [Function]
Return the default label for OS, as it will appear in the bootloader menu entry.

`operating-system-initrd-modules` [Special Form]

`operating-system-initrd` [Special Form]

`operating-system-users` [Special Form]

`operating-system-groups` [Special Form]

`operating-system-issue` [Special Form]

`operating-system-timezone` [Special Form]

`operating-system-locale` [Special Form]

`operating-system-locale-definitions` [Special Form]

`operating-system-locale-libcs` [Special Form]

`operating-system-mapped-devices` [Special Form]

`operating-system-file-systems` [Special Form]

`operating-system-store-file-system` *os* [Function]
Return the file system that contains the store of OS.

`operating-system-user-mapped-devices` *os* [Function]
Return the subset of mapped devices that can be installed in user-land–i.e., those not needed during boot.

`operating-system-boot-mapped-devices` *os* [Function]
Return the subset of mapped devices that must be installed during boot, from the initrd.

`operating-system-bootloader-crypto-devices` *os* [Function]

`operating-system-activation-script` *os* [Function]
Return the activation script for OS—i.e., the code that "activates" the stateful part of OS, including user accounts and groups, special directories, etc.

`operating-system-user-accounts` *os* [Function]
Return the list of user accounts of OS.

`operating-system-shepherd-service-names` *os* [Function]
Return the list of Shepherd service names for OS.

`operating-system-user-kernel-arguments` [Special Form]

`operating-system-firmware` [Special Form]

`operating-system-keyboard-layout` [Special Form]

`operating-system-name-service-switch` [Special Form]

`operating-system-pam-services` [Special Form]

`operating-system-setuid-programs` [Special Form]

`operating-system-skeletons` [Special Form]

`operating-system-sudoers-file` [Special Form]

`operating-system-swap-devices` [Special Form]

`operating-system-kernel-loadable-modules` [Special Form]

`operating-system-location` [Special Form]

`operating-system-derivation` *os* [Function]
Return a derivation that builds OS.

`operating-system-profile` *os* [Function]
Return a derivation that builds the system profile of OS.

`operating-system-bootcfg` *os* [*old-entries*] [Function]
Return the bootloader configuration file for OS. Use OLD-ENTRIES, a list of <menu-entry>, to populate the "old entries" menu.

`operating-system-etc-directory` *os* [Function]
Return that static part of the /etc directory of OS.

`operating-system-locale-directory` *os* [Function]
Return the directory containing the locales compiled for the definitions listed in OS. The C library expects to find it under /run/current-system/locale.

`operating-system-boot-script` *os* [Function]
Return the boot script for OS—i.e., the code started by the initrd once we're running in the final root.

`operating-system-uuid` *os* [*type*] [Function]
Compute UUID object with a deterministic "UUID" for OS, of the given TYPE (one of 'iso9660 or 'dce). Return a UUID object.

`system-linux-image-file-name` [*target*] [Function]
Return the basename of the kernel image file for TARGET.

`operating-system-with-gc-roots` *os roots* [Function]
Return a variant of OS where ROOTS are registered as GC roots.

`operating-system-with-provenance` *os* [*config-file*] [Function]
Return a variant of OS that stores its own provenance information, including CONFIG-FILE, if available. This is achieved by adding an instance of PROVENANCE-SERVICE-TYPE to its services.

`hurd-default-essential-services` *os* [Function]

`boot-parameters` [Special Form]

`boot-parameters?` [Special Form]

`boot-parameters-label` [Special Form]

`boot-parameters-root-device` [Special Form]

`boot-parameters-bootloader-name` [Special Form]

`boot-parameters-bootloader-menu-entries` [Special Form]

`boot-parameters-store-crypto-devices` [Special Form]

`boot-parameters-store-device` [Special Form]

`boot-parameters-store-directory-prefix` [Special Form]

`boot-parameters-store-mount-point` [Special Form]

`boot-parameters-locale` [Special Form]

`boot-parameters-kernel` [Special Form]

`boot-parameters-kernel-arguments` [Special Form]

`boot-parameters-initrd` [Special Form]

`boot-parameters-multiboot-modules` [Special Form]

`boot-parameters-version` [Special Form]

`%boot-parameters-version` [Variable]

`read-boot-parameters` *port* [Function]
Read boot parameters from PORT and return the corresponding <boot-parameters> object. Raise an error if the format is unrecognized.

`read-boot-parameters-file` *system* [Function]
Read boot parameters from SYSTEM's (system or generation) "parameters" file and returns the corresponding <boot-parameters> object or #f if the format is unrecognized. The object has its kernel-arguments extended in order to make it bootable.

`boot-parameters->menu-entry` *conf* [Function]
Return a <menu-entry> instance given CONF, a <boot-parameters> instance.

`local-host-aliases` *host-name* [Function]
Return aliases for HOST-NAME, to be used in /etc/hosts.

`%root-account` [Variable]

`%setuid-programs` [Variable]

`%sudoers-specification` [Variable]

`%base-packages` [Variable]

`%base-packages-artwork` [Variable]

`%base-packages-interactive` [Variable]

`%base-packages-linux` [Variable]

`%base-packages-networking` [Variable]

`%base-packages-disk-utilities` [Special Form]

`%base-packages-utils` [Variable]

`%base-firmware` [Variable]

`%default-kernel-arguments` [Variable]

13 (gnu tests)

13.1 Overview

```
This module provides the infrastructure to run operating system tests.
The most important part of that is tools to instrument the OS under test,
essentially allowing it to run in a virtual machine controlled by the host
system--hence the name "marionette".
```

13.2 Usage

`marionette-configuration` [Special Form]

`marionette-configuration?` [Special Form]

`marionette-configuration-device` [Special Form]

`marionette-configuration-imported-modules` [Special Form]

`marionette-configuration-requirements` [Special Form]

`marionette-service-type` [Variable]

`marionette-operating-system` *os* [*#:imported-modules*] [*#:extensions*] [*#:requirements*] [Function]

Return a marionetteed variant of OS such that OS can be used as a marionette in a virtual machine–i.e., controlled from the host system. The marionette service in the guest is started after the Shepherd services listed in REQUIREMENTS. The packages in the list EXTENSIONS are made available from the backdoor REPL.

`define-os-with-source` (*os source*) (*use-modules modules ...*) (*operating-system fields ...*) [Special Form]

Define two variables: OS containing the given operating system, and SOURCE containing the source to define OS as an sexp.

This is convenient when we need both the <operating-system> object so we can instantiate it, and the source to create it so we can store in in a file in the system under test.

`%simple-os` [Variable]

`simple-operating-system` *user-services ...* [Special Form]

Return an operating system that includes USER-SERVICES in addition to %BASE-SERVICES.

`system-test` [Special Form]

`system-test?` [Special Form]

`system-test-name` [Special Form]

`system-test-value` [Special Form]

`system-test-description` [Special Form]

`system-test-location` [Special Form]

`fold-system-tests` *proc seed* [Function]
Invoke PROC on each system test, passing it the test and the previous result.

`all-system-tests` [Function]
Return the list of system tests.

14 (guix android-repo-download)

14.1 Overview

```
An <origin> method that fetches a specific commit from an Android repo
repository.
The repository's manifest (URL and revision) can be specified with a
<android-repo-reference> object.
```

14.2 Usage

`android-repo-reference` [Special Form]

`android-repo-reference?` [Special Form]

`android-repo-reference-manifest-url` [Special Form]

`android-repo-reference-revision` [Variable]
[unbound!]

`android-repo-fetch` *ref hash-algo hash* [*name*] [*#:system*] [*#:guile*] [*#:git-repo*] [Function]
Return a fixed-output derivation that fetches REF, an <android-repo-reference> object. The output is expected to have recursive hash HASH of type HASH-ALGO (a symbol). Use NAME as the file name, or a generic name if unset.

`android-repo-version` *version revision* [Function]
Return the version string for packages using android-repo-download.

`android-repo-file-name` *name version* [Function]
Return the file-name for packages using android-repo-download.

15 (guix avahi)

15.1 Overview

15.2 Usage

`avahi-service` [Special Form]

`avahi-service?` [Special Form]

`avahi-service-name` [Special Form]

`avahi-service-type` [Special Form]

`avahi-service-interface` [Special Form]

`avahi-service-local-address` [Special Form]

`avahi-service-address` [Special Form]

`avahi-service-port` [Special Form]

`avahi-service-txt` [Special Form]

`avahi-publish-service-thread` *name* [*#:type*] [*#:port*] [*#:stop-loop?*] [*#:timeout*] [*#:txt*] [Function]

Publish the service TYPE using Avahi, for the given PORT, on all interfaces and for all protocols. Also, advertise the given TXT record list.

This procedure starts a new thread running the Avahi event loop. It exits when STOP-LOOP? procedure returns true.

`avahi-browse-service-thread` *proc* [*#:types*] [*#:ignore-local?*] [*#:family*] [*#:stop-loop?*] [*#:timeout*] [Function]

Browse services which type is part of the TYPES list, using Avahi. The search is restricted to services with the given FAMILY. Each time a service is found or removed, PROC is called and passed as argument the corresponding AVAHI-SERVICE record. If a service is available on multiple network interfaces, it will only be reported on the first interface found.

This procedure starts a new thread running the Avahi event loop. It exits when STOP-LOOP? procedure returns true.

16 (guix base16)

16.1 Overview

16.2 Usage

`bytevector->base16-string` *bv* [Function]
Return the hexadecimal representation of BV's contents.

`base16-string->bytevector` *s* [Function]
Return the bytevector whose hexadecimal representation is string S.

17 (guix base32)

17.1 Overview

A generic, customizable to convert bytevectors to/from a base32 representation.

17.2 Usage

`bytevector-quintet-length` *bv* [Function]
Return the number of quintets (including truncated ones) available in BV.

`bytevector->base32-string` *bv* [Function]
Return a base32 encoding of BV using BASE32-CHARS as the alphabet.

`bytevector->nix-base32-string` *bv* [Function]
Return a base32 encoding of BV using BASE32-CHARS as the alphabet.

`base32-string->bytevector` *s* [Function]
Return the binary representation of base32 string S as a bytevector.

`nix-base32-string->bytevector` *s* [Function]
Return the binary representation of base32 string S as a bytevector.

`%nix-base32-charset` [Variable]

`%rfc4648-base32-charset` [Variable]

`&invalid-base32-character` [Variable]

`invalid-base32-character?` *obj* [Function]

`invalid-base32-character-value` *obj* [Function]

`invalid-base32-character-string` *obj* [Function]

18 (guix base64)

18.1 Overview

18.2 Usage

`base64-encode` *bv* [Function]

`base64-decode` *str* [Function]

`base64-alphabet` [Variable]

`base64url-alphabet` [Variable]

`get-delimited-base64` *port* [Function]

`put-delimited-base64` *port type bv line-length* [Function]

19 (guix build-system)

19.1 Overview

19.2 Usage

`build-system` [Special Form]

`build-system?` [Special Form]

`build-system-name` [Special Form]

`build-system-description` [Special Form]

`build-system-lower` [Special Form]

`bag` [Special Form]

`bag?` [Special Form]

`bag-name` [Special Form]

`bag-system` [Special Form]

`bag-target` [Special Form]

`bag-build-inputs` [Special Form]

`bag-host-inputs` [Special Form]

`bag-target-inputs` [Special Form]

`bag-outputs` [Special Form]

`bag-arguments` [Special Form]

`bag-build` [Special Form]

`make-bag` *build-system name* [*#:source*] [*#:inputs*] [*#:native-inputs*] [*#:outputs*] [*#:arguments*] [*#:system*] [*#:target*] [Function]

Ask BUILD-SYSTEM to return a 'bag' for NAME, with the given SOURCE, INPUTS, NATIVE-INPUTS, OUTPUTS, and additional ARGUMENTS. If TARGET is not #f, it must be a string with the GNU triplet of a cross-compilation target.

This is the mechanism by which a package is `"lowered"` to a bag, which is the intermediate representation just above derivations.

`build-system-with-c-toolchain` *bs toolchain* [Function]

Return a variant of BS, a build system, that uses TOOLCHAIN instead of the default GNU C/C++ toolchain. TOOLCHAIN must be a list of inputs (label/package tuples) providing equivalent functionality, such as the 'gcc-toolchain' package.

20 (guix bzr-download)

20.1 Overview

An <origin> method that fetches a specific revision from a Bazaar repository. The repository URL and revision identifier are specified with a <bzr-reference> object.

20.2 Usage

bzr-reference [Special Form]

bzr-reference? [Special Form]

bzr-reference-url [Special Form]

bzr-reference-revision [Special Form]

bzr-fetch *ref hash-algo hash* [*name*] [*#:system*] [*#:guile*] [*#:bzr*] [Function]
Return a fixed-output derivation that fetches REF, a <bzr-reference> object. The output is expected to have recursive hash HASH of type HASH-ALGO (a symbol). Use NAME as the file name, or a generic name if #f.

21 (guix cache)

21.1 Overview

`This module provides tools to manage a simple on-disk cache consisting of individual files.`

21.2 Usage

`obsolete?` *date now ttl* [Function]
Return #t if DATE is obsolete compared to NOW + TTL seconds.

`delete-file*` *file* [Function]
Like 'delete-file', but does not raise an error when FILE does not exist.

`file-expiration-time` *ttl* [*timestamp*] [Function]
Return a procedure that, when passed a file, returns its "expiration time" computed as its timestamp + TTL seconds. Call TIMESTAMP to obtain the relevant timestamp from the result of 'stat'.

`remove-expired-cache-entries` *entries* [*#:now*] [Function]
[*#:entry-expiration*] [*#:delete-entry*]
Given ENTRIES, a list of file names, remove those whose expiration time, as returned by ENTRY-EXPIRATION, has passed. Use DELETE-ENTRY to delete them.

`maybe-remove-expired-cache-entries` *cache cache-entries* [Function]
[*#:entry-expiration*] [*#:delete-entry*] [*#:cleanup-period*]
Remove expired narinfo entries from the cache if deemed necessary. Call CACHE-ENTRIES with CACHE to retrieve the list of cache entries.

ENTRY-EXPIRATION must be a procedure that, when passed an entry, returns the expiration time of that entry in seconds since the Epoch. DELETE-ENTRY is a procedure that removes the entry passed as an argument. Finally, CLEANUP-PERIOD denotes the minimum time between two cache cleanups.

22 (guix channels)

22.1 Overview

```
This module implements "channels."  A channel is usually a source of
package definitions.  There's a special channel, the 'guix' channel, that
provides all of Guix, including its commands and its documentation.
User-defined channels are expected to typically provide a bunch of .scm
files meant to be added to the '%package-search-path'.

This module provides tools to fetch and update channels from a Git
repository and to build them.
```

22.2 Usage

`channel` [Special Form]

`channel?` [Special Form]

`channel-name` [Special Form]

`channel-url` [Special Form]

`channel-branch` [Special Form]

`channel-commit` [Special Form]

`channel-introduction` [Special Form]

`channel-location` [Special Form]

`channel-introduction?` [Special Form]

`make-channel-introduction` *commit signer* [Function]
Return a new channel introduction: COMMIT is the introductory where authentication starts, and SIGNER is the OpenPGP fingerprint (a bytevector) of the signer of that commit.

`channel-introduction-first-signed-commit` [Special Form]

`channel-introduction-first-commit-signer` [Special Form]

`openpgp-fingerprint->bytevector` *str* [Function]
Convert STR, an OpenPGP fingerprint (hexadecimal string with whitespace), to the corresponding bytevector.

`openpgp-fingerprint` [Special Form]
Convert STR, an OpenPGP fingerprint (hexadecimal string with whitespace), to the corresponding bytevector.

`%default-guix-channel` [Variable]

`%default-channels` [Variable]

`guix-channel?` *channel* [Function]
Return true if CHANNEL is the 'guix' channel.

`repository->guix-channel` *directory* [*#:introduction*] [Function]
Look for a Git repository in DIRECTORY or its ancestors and return a channel that uses that repository and the commit HEAD currently points to; use INTRODUCTION as the channel's introduction. Return #f if no Git repository could be found at DIRECTORY or one of its ancestors.

`channel-instance?` [Special Form]

`channel-instance-channel` [Special Form]

`channel-instance-commit` [Special Form]

`channel-instance-checkout` [Special Form]

`authenticate-channel` *channel checkout commit* [*#:keyring-reference-prefix*] [Function]
Authenticate the given COMMIT of CHANNEL, available at CHECKOUT, a directory containing a CHANNEL checkout. Raise an error if authentication fails.

`latest-channel-instances` *store channels* [*#:current-channels*] [*#:authenticate?*] [*#:validate-pull*] [Function]
Return a list of channel instances corresponding to the latest checkouts of CHANNELS and the channels on which they depend.

When AUTHENTICATE? is true, authenticate the subset of CHANNELS that has a "channel introduction".

CURRENT-CHANNELS is the list of currently used channels. It is compared against the newly-fetched instances of CHANNELS, and VALIDATE-PULL is called for each channel update and can choose to emit warnings or raise an error, depending on the policy it implements.

`checkout->channel-instance` *checkout* [*#:commit*] [*#:url*] [*#:name*] [Function]
Return a channel instance for CHECKOUT, which is assumed to be a checkout of COMMIT at URL. Use NAME as the channel name.

`latest-channel-derivation` [*channels*] [*#:current-channels*] [*#:validate-pull*] [Function]
Return as a monadic value the derivation that builds the profile for the latest instances of CHANNELS. CURRENT-CHANNELS and VALIDATE-PULL are passed to 'latest-channel-instances'.

`channel-instance->sexp` *instance* [Function]
Return an sexp representation of INSTANCE, a channel instance.

`channel-instances->manifest` *instances* [*#:system*] [Function]
Return a profile manifest with entries for all of INSTANCES, a list of channel instances. By default, build for the current system, or SYSTEM if specified.

`%channel-profile-hooks` [Variable]

`channel-instances->derivation` *instances* [Function]
Return the derivation of the profile containing INSTANCES, a list of channel instances.

`ensure-forward-channel-update` *channel start commit relation* [Function]
Raise an error if RELATION is not 'ancestor, meaning that START is not an ancestor of COMMIT, unless CHANNEL specifies a commit.

This procedure implements a channel update policy meant to be used as a #:validate-pull argument.

`profile-channels` *profile* [Function]
Return the list of channels corresponding to entries in PROFILE. If PROFILE is not a profile created by 'guix pull', return the empty list.

`manifest-entry-channel` *entry* [Function]
Return the channel ENTRY corresponds to, or #f if that information is missing or unreadable. ENTRY must be an entry created by 'channel-instances->manifest', with the 'source' property.

`sexp->channel` *sexp* [*name*] [Function]
Read SEXP, a provenance sexp as created by 'channel-instance->sexp'; use NAME as the channel name if SEXP does not specify it. Return #f if the sexp does not have the expected structure.

`channel->code` *channel* [*#:include-introduction?*] [Function]
Return code (an sexp) to build CHANNEL. When INCLUDE-INTRODUCTION? is true, include its introduction, if any.

`channel-news-entry?` [Special Form]

`channel-news-entry-commit` [Special Form]

`channel-news-entry-tag` [Special Form]

`channel-news-entry-title` [Special Form]

`channel-news-entry-body` [Special Form]

`channel-news-for-commit` *channel new* [*old*] [Function]
Return a list of <channel-news-entry> for CHANNEL between commits OLD and NEW. When OLD is omitted or is #f, return all the news entries of CHANNEL.

23 (guix ci)

23.1 Overview

This module provides a client to the HTTP interface of the Hydra and Cuirass continuous integration (CI) tools.

23.2 Usage

`build-product?` [Special Form]

`build-product-id` [Special Form]

`build-product-type` [Special Form]

`build-product-file-size` [Special Form]

`build-product-path` [Special Form]

`build?` [Special Form]

`build-id` [Special Form]

`build-derivation` [Special Form]

`build-evaluation` [Special Form]

`build-system` [Special Form]

`build-status` [Special Form]

`build-timestamp` [Special Form]

`build-start-time` [Special Form]

`build-stop-time` [Special Form]

`build-duration` *build* [Function]

Return the duration in seconds of BUILD.

`build-products` [Special Form]

`checkout?` [Special Form]

`checkout-commit` [Special Form]

`checkout-channel` [Special Form]

`evaluation?` [Special Form]

`evaluation-id` [Special Form]

`evaluation-spec` [Special Form]

`evaluation-complete?` [Special Form]

`evaluation-checkouts` [Special Form]

`job?` [Special Form]

`job-build-id` [Special Form]

`job-status` [Special Form]

`job-name` [Special Form]

`history?` [Special Form]

`history-evaluation` [Special Form]

`history-checkouts` [Special Form]

`history-jobs` [Special Form]

`%query-limit` [Variable]

`queued-builds` *url* [*limit*] [Function]
Return the list of queued derivations on URL.

`latest-builds` *url* [*limit*] [*#:evaluation*] [*#:system*] [*#:job*] [*#:jobset*] [*#:status*] [Function]
Return the latest builds performed by the CI server at URL. If EVALUATION is an integer, restrict to builds of EVALUATION. If SYSTEM is true (a system string such as "x86_64-linux"), restrict to builds for SYSTEM.

`evaluation` *url evaluation* [Function]
Return the given EVALUATION performed by the CI server at URL.

`evaluation-jobs` *url evaluation-id* [Function]
Return the list of jobs of evaluation EVALUATION-ID.

`build` *url id* [Function]
Look up build ID at URL and return it. Raise &http-get-error if it is not found (404).

`job-build` *url job* [Function]
Return the build associated with JOB.

`jobs-history` *url jobs* [*#:specification*] [*#:limit*] [Function]
Return the job history for the SPECIFICATION jobs which names are part of the JOBS list, from the CI server at URL. Limit the history to the latest LIMIT evaluations.

`latest-evaluations` *url* [*limit*] [*#:spec*] [Function]
Return the latest evaluations performed by the CI server at URL. If SPEC is passed, only consider the evaluations for the given SPEC specification.

`evaluations-for-commit` *url commit* [*limit*] [Function]
Return the evaluations among the latest LIMIT evaluations that have COMMIT as one of their inputs.

`channel-with-substitutes-available` *chan url* [Function]

Return a channel inheriting from CHAN but which commit field is set to the latest commit with available substitutes for the Guix package definitions at URL. The current system is taken into account.

If no commit with available substitutes were found, the commit field is set to false and a warning message is printed.

24 (guix colors)

24.1 Overview

This module provides tools to produce colored output using ANSI escapes.

24.2 Usage

`color` *colors ...* [Special Form]
Return a list of color attributes that can be passed to 'colorize-string'.

`color?` [Special Form]

`coloring-procedure` *color* [Function]
Return a procedure that applies COLOR to the given string.

`colorize-string` *str color* [Function]
Return a copy of STR colorized using ANSI escape sequences according to COLOR. At the end of the returned string, the color attributes are reset such that subsequent output will not have any colors in effect.

`highlight` *str* [*port*] [Function]
Return STR with extra ANSI color attributes if PORT supports it.

`highlight/warn` *str* [*port*] [Function]
Return STR with extra ANSI color attributes if PORT supports it.

`dim` *str* [*port*] [Function]
Return STR with extra ANSI color attributes if PORT supports it.

`colorize-full-matches` *rules* [Function]
Return a procedure that, given a string, colorizes according to RULES. RULES must be a list of regexp/color pairs; the whole match of a regexp is colorized with the corresponding color.

`color-rules` (*regexp colors ...*) *...* [Special Form]
Return a procedure that colorizes the string it is passed according to the given rules. Each rule has the form:

```
(REGEXP COLOR1 COLOR2 ...)
```

where COLOR1 specifies how to colorize the first submatch of REGEXP, and so on.

`color-output?` *port* [Function]
Return true if we should write colored output to PORT.

`isatty?*` *port* [Function]

`supports-hyperlinks?` [*port*] [Function]
Return true if PORT is a terminal that supports hyperlink escapes.

`file-hyperlink` *file* [*text*] [Function]
Return TEXT with escapes for a hyperlink to FILE.

`hyperlink` *uri text* [Function]
Return a string that denotes a hyperlink using an OSC escape sequence as documented at <https://gist.github.com/egmontkob/eb114294efbcd5adb1944c9f3cb5feda>.

25 (guix combinators)

25.1 Overview

```
This module provides useful combinators that complement SRFI-1 and
friends.
```

25.2 Usage

`fold2` *proc seed1 seed2 lst* [Function]

`fold-tree` *proc init children roots* [Function]
Call (PROC NODE RESULT) for each node in the tree that is reachable from ROOTS, using INIT as the initial value of RESULT. The order in which nodes are traversed is not specified, however, each node is visited only once, based on an eq? check. Children of a node to be visited are generated by calling (CHILDREN NODE), the result of which should be a list of nodes that are connected to NODE in the tree, or '() or #f if NODE is a leaf node.

`fold-tree-leaves` *proc init children roots* [Function]
Like fold-tree, but call (PROC NODE RESULT) only for leaf nodes.

`compile-time-value` *exp* [Special Form]
Evaluate the given expression at compile time. The expression must evaluate to a simple datum.

`procedure-call-location` [Special Form]

`define-compile-time-procedure` (*proc* (*arg pred*) ...) *body ...* [Special Form]
Define PROC as a macro such that, if every actual argument in a "call" matches PRED, then BODY is evaluated at macro-expansion time. BODY must return a single value in a type that has read syntax–e.g., numbers, strings, lists, etc.

BODY can refer to 'procedure-call-location', which is bound to a source property alist corresponding to the call site.

This macro is meant to be used primarily for small procedures that validate or process its arguments in a way that may be equally well performed at macro-expansion time.

26 (guix config)

26.1 Overview

Compile-time configuration of Guix. When adding a substitution variable here, make sure to equip (guix scripts pull) to substitute it.

26.2 Usage

%guix-package-name [Variable]

%guix-version [Variable]

%guix-bug-report-address [Variable]

%guix-home-page-url [Variable]

%channel-metadata [Variable]

%storedir [Variable]

%localstatedir [Variable]

%sysconfdir [Variable]

%store-directory [Variable]

%state-directory [Variable]

%store-database-directory [Variable]

%config-directory [Variable]

%system [Variable]

%gzip [Variable]

%bzip2 [Variable]

%xz [Variable]

27 (guix cpio)

27.1 Overview

```
This module implements the cpio "new ASCII" format, bit-for-bit identical
to GNU cpio with the '-H newc' option.
```

27.2 Usage

`cpio-header?` [Special Form]

`make-cpio-header` [*#:inode*] [*#:mode*] [*#:uid*] [*#:gid*] [*#:nlink*] [*#:mtime*] [*#:size*] [*#:dev*] [*#:rdev*] [*#:name-size*] [Function]
Return a new cpio file header.

`file->cpio-header` *file* [*file-name*] [*#:stat*] [Function]
Return a cpio header corresponding to the info returned by STAT for FILE, using FILE-NAME as its file name.

`file->cpio-header*` *file* [*file-name*] [*#:stat*] [Function]
Similar to 'file->cpio-header', but return a header with a zeroed modification time, inode number, UID/GID, etc. This allows archives to be produced in a deterministic fashion.

`special-file->cpio-header*` *file device-type device-major device-minor permission-bits* [*file-name*] [Function]
Create a character or block device header.

DEVICE-TYPE is either 'char-special or 'block-special.

The number of hard links is assumed to be 1.

`write-cpio-header` *obj port* [Function]

`read-cpio-header` *port* [Function]

`write-cpio-archive` *files port* [*#:file->header*] [Function]
Write to PORT a cpio archive in "new ASCII" format containing all of FILES.

The archive written to PORT is intended to be bit-identical to what GNU cpio produces with the '-H newc' option.

28 (guix cpu)

28.1 Overview

```
This module provides tools to determine the micro-architecture supported
by the CPU and to map it to a name known to GCC's '-march'.
```

28.2 Usage

`current-cpu` [Function]

`cpu?` [Special Form]

`cpu-architecture` [Special Form]

`cpu-vendor` [Special Form]

`cpu-family` [Special Form]

`cpu-model` [Special Form]

`cpu-flags` [Special Form]

`cpu->gcc-architecture` *cpu* [Function]

Return the architecture name, suitable for GCC's '-march' flag, that corresponds to CPU, a record as returned by 'current-cpu'.

29 (guix cve)

29.1 Overview

This modules provides the tools to fetch, parse, and digest part of the Common Vulnerabilities and Exposures (CVE) feeds provided by the US NIST at <https://nvd.nist.gov/vuln/data-feeds>.

29.2 Usage

`json->cve-items` *json* [Function]
Parse JSON, an input port or a string, and return a list of <cve-item> records.

`cve-item?` [Special Form]

`cve-item-cve` [Special Form]

`cve-item-configurations` [Special Form]

`cve-item-published-date` [Special Form]

`cve-item-last-modified-date` [Special Form]

`cve?` [Special Form]

`cve-id` [Special Form]

`cve-data-type` [Special Form]

`cve-data-format` [Special Form]

`cve-references` [Special Form]

`cve-reference?` [Special Form]

`cve-reference-url` [Special Form]

`cve-reference-tags` [Special Form]

`vulnerability?` [Special Form]

`vulnerability-id` [Special Form]

`vulnerability-packages` [Special Form]

`json->vulnerabilities` *json* [Function]
Parse JSON, an input port or a string, and return the list of vulnerabilities found therein.

`current-vulnerabilities` [*#:timeout*] [Function]
Return the current list of Common Vulnerabilities and Exposures (CVE) as published by the US NIST. TIMEOUT specifies the timeout in seconds for connection establishment.

`vulnerabilities->lookup-proc` *vulnerabilities* [Function]
Return a lookup procedure built from VULNERABILITIES that takes a package name and optionally a version number. When the version is omitted, the lookup procedure returns a list of vulnerabilities; otherwise, it returns a list of vulnerabilities affecting the given package version.

30 (guix cvs-download)

30.1 Overview

An <origin> method that fetches a specific revision or date from a CVS repository. The CVS-ROOT-DIRECTORY, MODULE and REVISION are specified with a <cvs-reference> object. REVISION should be specified as either a date string in ISO-8601 format (e.g. "2012-12-21") or a CVS tag.

30.2 Usage

`cvs-reference` [Special Form]

`cvs-reference?` [Special Form]

`cvs-reference-root-directory` [Special Form]

`cvs-reference-module` [Special Form]

`cvs-reference-revision` [Special Form]

`cvs-fetch` *ref hash-algo hash* [*name*] [*#:system*] [*#:guile*] [*#:cvs*] [Function]
Return a fixed-output derivation that fetches REF, a <cvs-reference> object. The output is expected to have recursive hash HASH of type HASH-ALGO (a symbol). Use NAME as the file name, or a generic name if #f.

31 (guix deprecation)

31.1 Overview

Provide a mechanism to mark bindings as deprecated.

31.2 Usage

`define-deprecated` [Special Form]

Define a deprecated variable or procedure, along these lines:

```
(define-deprecated foo bar 42)
(define-deprecated old new)
(define-deprecated (baz x y) qux (qux y x))
```

This will write a deprecation warning to GUIX-WARNING-PORT.

`define-deprecated/public` *body ...* [Special Form]

Like 'define/deprecated', but export all the newly introduced bindings.

`define-deprecated/alias` *deprecated replacement* [Special Form]

Define as an alias a deprecated variable, procedure, or macro, along these lines:

```
(define-deprecated/alias nix-server? store-connection?)
```

where 'nix-server?' is the deprecated name for 'store-connection?'.

This will write a deprecation warning to GUIX-WARNING-PORT.

`define-deprecated/public-alias` *deprecated replacement* [Special Form]

Like define-deprecated/alias, but exporting DEPRECATED. It is assumed, that REPLACEMENT is already public.

`warn-about-old-daemon` [Function]

`warn-about-deprecation` *variable properties* [*#:replacement*] [Function]

32 (guix derivations)

32.1 Overview

32.2 Usage

`<derivation>` [Variable]

`derivation?` [Special Form]

`derivation-outputs` [Special Form]

`derivation-inputs` [Special Form]

`derivation-sources` [Special Form]

`derivation-system` [Special Form]

`derivation-builder` [Special Form]

`derivation-builder-arguments` [Special Form]

`derivation-builder-environment-vars` [Special Form]

`derivation-file-name` [Special Form]

`derivation-prerequisites` *drv* [*cut?*] [Function]

Return the list of derivation-inputs required to build DRV, recursively.

CUT? is a predicate that is passed a derivation-input and returns true to eliminate the given input and its dependencies from the search. An example of such a predicate is 'valid-derivation-input?'; when it is used as CUT?, the result is the set of prerequisites of DRV not already in valid.

`derivation-build-plan` *store inputs* [*#:mode*] [*#:substitutable-info*] [Function]

Given INPUTS, a list of derivation-inputs, return two values: the list of derivations to build, and the list of substitutable items that, together, allow INPUTS to be realized.

SUBSTITUTABLE-INFO must be a one-argument procedure similar to that returned by 'substitution-oracle'.

`derivation-prerequisites-to-build` [Special Form]

`<derivation-output>` [Variable]

`derivation-output?` [Special Form]

`derivation-output-path` [Special Form]

`derivation-output-hash-algo` [Special Form]

`derivation-output-hash` [Special Form]

`derivation-output-recursive?` [Special Form]

`<derivation-input>` [Variable]

`derivation-input?` [Special Form]

`derivation-input` *drv* [*outputs*] [Function]
Return a <derivation-input> for the OUTPUTS of DRV.

`derivation-input-path` *input* [Function]
Return the file name of the derivation INPUT refers to.

`derivation-input-derivation` [Special Form]

`derivation-input-sub-derivations` [Special Form]

`derivation-input-output-paths` *input* [Function]
Return the list of output paths corresponding to INPUT, a <derivation-input>.

`derivation-input-output-path` *input* [Function]
Return the output file name of INPUT. If INPUT has more than one outputs, an error is raised.

`valid-derivation-input?` *store input* [Function]
Return true if INPUT is valid–i.e., if all the outputs it requests are in the store.

`&derivation-error` [Variable]

`derivation-error?` *obj* [Function]

`derivation-error-derivation` *obj* [Function]

`&derivation-missing-output-error` [Variable]

`derivation-missing-output-error?` *obj* [Function]

`derivation-missing-output` *obj* [Function]

`derivation-name` *drv* [Function]
Return the base name of DRV.

`derivation-output-names` *drv* [Function]
Return the names of the outputs of DRV.

`fixed-output-derivation?` *drv* [Function]
Return #t if DRV is a fixed-output derivation, such as the result of a download with a fixed hash (aka. 'fetchurl').

`offloadable-derivation?` *drv* [Function]
Return true if DRV can be offloaded, false otherwise.

`substitutable-derivation?` *drv* [Function]
Return #t if DRV can be substituted.

`derivation-input-fold` *proc seed inputs* [*#:cut?*] [Function]
Perform a breadth-first traversal of INPUTS, calling PROC on each input with the current result, starting from SEED. Skip recursion on inputs that match CUT?.

`substitution-oracle` *store inputs-or-drv* [*#:mode*] [Function]
Return a one-argument procedure that, when passed a store file name, returns a 'substitutable?' if it's substitutable and #f otherwise.

The returned procedure knows about all substitutes for all the derivation inputs or derivations listed in INPUTS-OR-DRV, *except* those that are already valid (that is, it won't bother checking whether an item is substitutable if it's already on disk); it also knows about their prerequisites, unless they are themselves substitutable.

Creating a single oracle (thus making a single 'substitutable-path-info' call) and reusing it is much more efficient than calling 'has-substitutes?' or similar repeatedly, because it avoids the costs associated with launching the substituter many times.

`derivation-hash` *drv* [Function]
Return the hash of DRV, modulo its fixed-output inputs, as a bytevector.

`derivation-properties` *drv* [Function]

`read-derivation` *drv-port* [*read-derivation-from-file*] [Function]
Read the derivation from DRV-PORT and return the corresponding <derivation> object. Call READ-DERIVATION-FROM-FILE to read derivations declared as inputs of the derivation being parsed.

Most of the time you'll want to use 'read-derivation-from-file', which caches things as appropriate and is thus more efficient.

`read-derivation-from-file` *file* [Function]
Read the derivation in FILE, a '.drv' file, and return the corresponding <derivation> object.

`write-derivation` *drv port* [Function]
Write the ATerm-like serialization of DRV to PORT. See Section 2.4 of Eelco Dolstra's PhD dissertation for an overview of a previous version of that form.

`derivation->output-path` *drv* [*output*] [Function]
Return the store path of its output OUTPUT. Raise a '&derivation-missing-output-error' condition if OUTPUT is not an output of DRV.

`derivation->output-paths` *drv* [Function]
Return the list of name/path pairs of the outputs of DRV.

`derivation-path->output-path` *path* [*output*] [Function]
Read the derivation from PATH ('/gnu/store/xxx.drv'), and return the store path of its output OUTPUT.

`derivation-path->output-paths` *path* [Function]
Read the derivation from PATH ('/gnu/store/xxx.drv'), and return the list of name/path pairs of its outputs.

`derivation` *store name builder args* [*#:system*] [*#:env-vars*] [*#:inputs*] [*#:sources*] [*#:outputs*] [*#:hash*] [*#:hash-algo*] [*#:recursive?*] [*#:references-graphs*] [*#:allowed-references*] [*#:disallowed-references*] [*#:leaked-env-vars*] [*#:local-build?*] [*#:substitutable?*] [*#:properties*] [*#:%deprecation-warning?*] [Function]

Build a derivation with the given arguments, and return the resulting <derivation> object. When HASH and HASH-ALGO are given, a fixed-output derivation is created—i.e., one whose result is known in advance, such as a file download. If, in addition, RECURSIVE? is true, then that fixed output may be an executable file or a directory and HASH must be the hash of an archive containing this output.

When REFERENCES-GRAPHS is true, it must be a list of file name/store path pairs. In that case, the reference graph of each store path is exported in the build environment in the corresponding file, in a simple text format.

When ALLOWED-REFERENCES is true, it must be a list of store items or outputs that the derivation's outputs may refer to. Likewise, DISALLOWED-REFERENCES, if true, must be a list of things the outputs may not refer to.

When LEAKED-ENV-VARS is true, it must be a list of strings denoting environment variables that are allowed to "leak" from the daemon's environment to the build environment. This is only applicable to fixed-output derivations–i.e., when HASH is true. The main use is to allow variables such as "http_proxy" to be passed to derivations that download files.

When LOCAL-BUILD? is true, declare that the derivation is not a good candidate for offloading and should rather be built locally. This is the case for small derivations where the costs of data transfers would outweigh the benefits.

When SUBSTITUTABLE? is false, declare that substitutes of the derivation's output should not be used.

PROPERTIES must be an association list describing "properties" of the derivation. It is kept as-is, uninterpreted, in the derivation.

`raw-derivation` . *args* [Function]

Build a derivation with the given arguments, and return the resulting <derivation> object. When HASH and HASH-ALGO are given, a fixed-output derivation is created—i.e., one whose result is known in advance, such as a file download. If, in addition, RECURSIVE? is true, then that fixed output may be an executable file or a directory and HASH must be the hash of an archive containing this output.

When REFERENCES-GRAPHS is true, it must be a list of file name/store path pairs. In that case, the reference graph of each store path is exported in the build environment in the corresponding file, in a simple text format.

When ALLOWED-REFERENCES is true, it must be a list of store items or outputs that the derivation's outputs may refer to. Likewise, DISALLOWED-REFERENCES, if true, must be a list of things the outputs may not refer to.

When LEAKED-ENV-VARS is true, it must be a list of strings denoting environment variables that are allowed to "leak" from the daemon's environment to the build environment. This is only applicable to fixed-output derivations–i.e., when HASH

is true. The main use is to allow variables such as "http_proxy" to be passed to derivations that download files.

When LOCAL-BUILD? is true, declare that the derivation is not a good candidate for offloading and should rather be built locally. This is the case for small derivations where the costs of data transfers would outweigh the benefits.

When SUBSTITUTABLE? is false, declare that substitutes of the derivation's output should not be used.

PROPERTIES must be an association list describing "properties" of the derivation. It is kept as-is, uninterpreted, in the derivation.

`invalidate-derivation-caches!` [Function]
Invalidate internal derivation caches. This is mostly useful for long-running processes that know what they're doing. Use with care!

`map-derivation` *store drv mapping* [*#:system*] [Function]
Given MAPPING, a list of pairs of derivations, return a derivation based on DRV where all the 'car's of MAPPING have been replaced by its 'cdr's, recursively.

`build-derivations` *store derivations* [*mode*] [Function]
Build DERIVATIONS, a list of <derivation> or <derivation-input> objects, .drv file names, or derivation/output pairs, using the specified MODE.

`built-derivations` . *args* [Function]
Build DERIVATIONS, a list of <derivation> or <derivation-input> objects, .drv file names, or derivation/output pairs, using the specified MODE.

`file-search-error?` *obj* [Function]

`file-search-error-file-name` *obj* [Function]

`file-search-error-search-path` *obj* [Function]

`search-path*` *path file* [Function]

`module->source-file-name` *module* [Function]
Return the file name corresponding to MODULE, a Guile module name (a list of symbols.)

`build-expression->derivation` [Special Form]

`%guile-for-build` [Variable]

33 (guix describe)

33.1 Overview

This module provides supporting code to allow a Guix instance to find, at run time, which profile it's in (profiles created by 'guix pull'). That allows it to read meta-information about itself (e.g., repository URL and commit ID) and to find other channels available in the same profile. It's a bit like ELPA's pkg-info.el.

33.2 Usage

`current-profile` [Function]

`current-profile-date` [Function]
Return the creation date of the current profile (produced by 'guix pull'), as a number of seconds since the Epoch, or #f if it could not be determined.

`current-profile-entries` [Function]

`current-channels` [Function]

`package-path-entries` [Function]
Return two values: the list of package path entries to be added to the package search path, and the list to be added to %LOAD-COMPILED-PATH. These entries are taken from the 'guix pull' profile the calling process lives in, when applicable.

`package-provenance` *package* [Function]
Return the provenance of PACKAGE as an sexp for use as the 'provenance' property of manifest entries, or #f if it could not be determined.

`package-channels` *package* [Function]
Return the list of channels providing PACKAGE or an empty list if it could not be determined.

`manifest-entry-with-provenance` *entry* [Function]
Return ENTRY with an additional 'provenance' property if it's not already there.

`manifest-entry-provenance` *entry* [Function]
Return the list of channels ENTRY comes from. Return the empty list if that information is missing.

34 (guix diagnostics)

34.1 Overview

This module provides the tools to report diagnostics to the user in a consistent way: errors, warnings, and notes.

34.2 Usage

`warning` [Special Form]

`info` [Special Form]

`report-error` [Special Form]

`leave` *args ...* [Special Form]
Emit an error message and exit.

`<location>` [Variable]

`location` *file line column* [Function]
Return the <location> object for the given FILE, LINE, and COLUMN.

`location?` [Special Form]

`location-file` [Special Form]

`location-line` [Special Form]

`location-column` [Special Form]

`source-properties->location` *loc* [Function]
Return a location object based on the info in LOC, an alist as returned by Guile's 'source-properties', 'frame-source', 'current-source-location', etc.

`location->source-properties` *loc* [Function]
Return the source property association list based on the info in LOC, a location object.

`location->string` *loc* [Function]
Return a human-friendly, GNU-standard representation of LOC.

`location->hyperlink` *location* [Function]
Return a string corresponding to LOCATION, with escapes for a hyperlink.

`&error-location` [Variable]

`error-location?` *obj* [Function]

`error-location` *obj* [Function]

`formatted-message` [Special Form]
Return a '&formatted-message' error condition.

`formatted-message?` *obj* [Function]

`formatted-message-string` *obj* [Function]

`formatted-message-arguments` *obj* [Function]

`&fix-hint` [Variable]

`fix-hint?` *obj* [Function]

`condition-fix-hint` *obj* [Function]

`guix-warning-port` [Variable]

`program-name` [Variable]

`define-with-syntax-properties` [Special Form]

Define BINDING to be a syntax form replacing each VALUE-IDENTIFIER and SYNTAX-PROPERTIES-IDENTIFIER in body by the syntax and syntax-properties, respectively, of each ensuing syntax object.

35 (guix discovery)

35.1 Overview

```
This module provides tools to discover Guile modules and the variables
they export.
```

35.2 Usage

`scheme-files` *directory* [Function]
Return the list of Scheme files found under DIRECTORY, recursively. The returned list is sorted in alphabetical order. Return the empty list if DIRECTORY is not accessible.

`scheme-modules` *directory* [*sub-directory*] [*#:warn*] [Function]
Return the list of Scheme modules available under DIRECTORY. Optionally, narrow the search to SUB-DIRECTORY.

WARN is called when a module could not be loaded. It is passed the module name and the exception key and arguments.

`scheme-modules*` *directory* [*sub-directory*] [Function]
Return the list of module names found under SUB-DIRECTORY in DIRECTORY. This is a source-only variant that does not try to load files.

`fold-modules` *proc init path* [*#:warn*] [Function]
Fold over all the Scheme modules present in PATH, a list of directories. Call (PROC MODULE RESULT) for each module that is found.

`all-modules` *path* [*#:warn*] [Function]
Return the list of package modules found in PATH, a list of directories to search. Entries in PATH can be directory names (strings) or (DIRECTORY . SUB-DIRECTORY) pairs, in which case modules are searched for beneath SUB-DIRECTORY. Modules are listed in the order they appear on the path.

`fold-module-public-variables` *proc init modules* [Function]
Call (PROC OBJECT RESULT) for each variable exported by one of MODULES, using INIT as the initial value of RESULT. It is guaranteed to never traverse the same object twice.

`fold-module-public-variables*` *proc init modules* [Function]
Call (PROC MODULE SYMBOL VARIABLE RESULT) for each variable exported by one of MODULES, using INIT as the initial value of RESULT. It is guaranteed to never traverse the same object twice.

36 (guix docker)

36.1 Overview

36.2 Usage

`build-docker-image` *image paths prefix* [*#:repository*] [*#:extra-files*] [*#:transformations*] [*#:system*] [*#:database*] [*#:entry-point*] [*#:environment*] [*#:compressor*] [*#:creation-time*] [Function]

Write to IMAGE a Docker image archive containing the given PATHS. PREFIX must be a store path that is a prefix of any store paths in PATHS. REPOSITORY is a descriptive name that will show up in "REPOSITORY" column of the output of "docker images".

When DATABASE is true, copy it to /var/guix/db in the image and create /var/guix/gcroots and friends.

When ENTRY-POINT is true, it must be a list of strings; it is stored as the entry point in the Docker image JSON structure.

ENVIRONMENT must be a list of name/value pairs. It specifies the environment variables that must be defined in the resulting image.

EXTRA-FILES must be a list of directives for 'evaluate-populate-directive' describing non-store files that must be created in the image.

TRANSFORMATIONS must be a list of (OLD -> NEW) tuples describing how to transform the PATHS. Any path in PATHS that begins with OLD will be rewritten in the Docker image so that it begins with NEW instead. If a path is a non-empty directory, then its contents will be recursively added, as well.

SYSTEM is a GNU triplet (or prefix thereof) of the system the binaries in PATHS are for; it is used to produce metadata in the image. Use COMPRESSOR, a command such as '("gzip" "-9n"), to compress IMAGE. Use CREATION-TIME, a SRFI-19 time-utc object, as the creation time in metadata.

37 (guix download)

37.1 Overview

`Produce fixed-output derivations with data fetched over HTTP or FTP.`

37.2 Usage

`%mirrors` [Variable]

`%disarchive-mirrors` [Variable]

`%download-fallback-test` [Variable]

`url-fetch/executable` *url hash-algo hash* [*name*] [*#:system*] [*#:guile*] [Function]

Like 'url-fetch', but make the downloaded file executable.

`url-fetch/tarbomb` *url hash-algo hash* [*name*] [*#:system*] [*#:guile*] [Function]

Similar to 'url-fetch' but unpack the file from URL in a directory of its own. This helper makes it easier to deal with "tar bombs".

`url-fetch/zipbomb` *url hash-algo hash* [*name*] [*#:system*] [*#:guile*] [Function]

Similar to 'url-fetch' but unpack the zip file at URL in a directory of its own. This helper makes it easier to deal with "zip bombs".

`download-to-store` *store url* [*name*] [*#:log*] [*#:recursive?*] [*#:verify-certificate?*] [Function]

Download from URL to STORE, either under NAME or URL's basename if omitted. Write progress reports to LOG. RECURSIVE? has the same effect as the same-named parameter of 'add-to-store'. VERIFY-CERTIFICATE? determines whether or not to validate HTTPS server certificates.

`url-fetch` *url hash-algo hash* [*name*] [*#:system*] [*#:guile*] [*#:executable?*] [Function]

Return a fixed-output derivation that fetches data from URL (a string, or a list of strings denoting alternate URLs), which is expected to have hash HASH of type HASH-ALGO (a symbol). By default, the file name is the base name of URL; optionally, NAME can specify a different file name. When EXECUTABLE? is true, make the downloaded file executable.

When one of the URL starts with mirror://, then its host part is interpreted as the name of a mirror scheme, taken from %MIRROR-FILE.

Alternatively, when URL starts with file://, return the corresponding file name in the store.

38 (guix elf)

38.1 Overview

This file was taken from the Guile 2.1 branch, where it is known as (system vm elf), and renamed to (guix elf). It will be unneeded when Guix switches to Guile 2.1/2.2.

A module to read and write Executable and Linking Format (ELF) files.

This module exports a number of record types that represent the various parts that make up ELF files. Fundamentally this is the main header, the segment headers (program headers), and the section headers. It also exports bindings for symbolic constants and utilities to parse and write special kinds of ELF sections.

See elf(5) for more information on ELF.

38.2 Usage

has-elf-header? *bv* [Function]

elf? [Special Form]

elf-bytes [Special Form]

elf-word-size [Special Form]

elf-byte-order [Special Form]

elf-abi [Special Form]

elf-type [Special Form]

elf-machine-type [Special Form]

elf-entry [Special Form]

elf-phoff [Special Form]

elf-shoff [Special Form]

elf-flags [Special Form]

elf-ehsize [Special Form]

elf-phentsize [Special Form]

elf-phnum [Special Form]

elf-shentsize [Special Form]

elf-shnum [Special Form]

`elf-section-flags` [Special Form]
`elf-section-addr` [Special Form]
`elf-section-offset` [Special Form]
`elf-section-size` [Special Form]
`elf-section-link` [Special Form]
`elf-section-info` [Special Form]
`elf-section-addralign` [Special Form]
`elf-section-entsize` [Special Form]
`elf-section-header-len` *word-size* [Function]
`elf-section-header-addr-offset` *word-size* [Function]
`elf-section-header-offset-offset` *word-size* [Function]
`write-elf-section-header` *bv offset byte-order word-size sec* [Function]
`elf-symbol?` [Special Form]
`elf-symbol-name` [Special Form]
`elf-symbol-value` [Special Form]
`elf-symbol-size` [Special Form]
`elf-symbol-info` [Special Form]
`elf-symbol-other` [Special Form]
`elf-symbol-shndx` [Special Form]
`elf-symbol-binding` *sym* [Function]
`elf-symbol-type` *sym* [Function]
`elf-symbol-visibility` *sym* [Function]
`elf-symbol-len` *word-size* [Function]
`elf-symbol-value-offset` *word-size* [Function]
`write-elf-symbol` *bv offset byte-order word-size sym* [Function]
`SHN_UNDEF` [Variable]
`SHT_NULL` [Variable]
`SHT_PROGBITS` [Variable]
`SHT_SYMTAB` [Variable]
`SHT_STRTAB` [Variable]
`SHT_RELA` [Variable]
`SHT_HASH` [Variable]
`SHT_DYNAMIC` [Variable]

SHT_NOTE [Variable]
SHT_NOBITS [Variable]
SHT_REL [Variable]
SHT_SHLIB [Variable]
SHT_DYNSYM [Variable]
SHT_INIT_ARRAY [Variable]
SHT_FINI_ARRAY [Variable]
SHT_PREINIT_ARRAY [Variable]
SHT_GROUP [Variable]
SHT_SYMTAB_SHNDX [Variable]
SHT_NUM [Variable]
SHT_LOOS [Variable]
SHT_HIOS [Variable]
SHT_LOPROC [Variable]
SHT_HIPROC [Variable]
SHT_LOUSER [Variable]
SHT_HIUSER [Variable]
SHF_WRITE [Variable]
SHF_ALLOC [Variable]
SHF_EXECINSTR [Variable]
SHF_MERGE [Variable]
SHF_STRINGS [Variable]
SHF_INFO_LINK [Variable]
SHF_LINK_ORDER [Variable]
SHF_OS_NONCONFORMING [Variable]
SHF_GROUP [Variable]
SHF_TLS [Variable]
DT_NULL [Variable]
DT_NEEDED [Variable]
DT_PLTRELSZ [Variable]
DT_PLTGOT [Variable]
DT_HASH [Variable]
DT_STRTAB [Variable]

DT_SYMTAB [Variable]

DT_RELA [Variable]

DT_RELASZ [Variable]

DT_RELAENT [Variable]

DT_STRSZ [Variable]

DT_SYMENT [Variable]

DT_INIT [Variable]

DT_FINI [Variable]

DT_SONAME [Variable]

DT_RPATH [Variable]

DT_SYMBOLIC [Variable]

DT_REL [Variable]

DT_RELSZ [Variable]

DT_RELENT [Variable]

DT_PLTREL [Variable]

DT_DEBUG [Variable]

DT_TEXTREL [Variable]

DT_JMPREL [Variable]

DT_BIND_NOW [Variable]

DT_INIT_ARRAY [Variable]

DT_FINI_ARRAY [Variable]

DT_INIT_ARRAYSZ [Variable]

DT_FINI_ARRAYSZ [Variable]

DT_RUNPATH [Variable]

DT_FLAGS [Variable]

DT_ENCODING [Variable]

DT_PREINIT_ARRAY [Variable]

DT_PREINIT_ARRAYSZ [Variable]

DT_NUM [Variable]

DT_LOGUILE [Variable]

DT_GUILE_GC_ROOT [Variable]

DT_GUILE_GC_ROOT_SZ [Variable]

DT_GUILE_ENTRY [Variable]

`STV_DEFAULT` [Variable]

`STV_INTERNAL` [Variable]

`STV_HIDDEN` [Variable]

`STV_PROTECTED` [Variable]

`NT_GNU_ABI_TAG` [Variable]

`NT_GNU_HWCAP` [Variable]

`NT_GNU_BUILD_ID` [Variable]

`NT_GNU_GOLD_VERSION` [Variable]

`parse-elf` *bv* [Function]

`elf-segment` *elf n* [Function]

`elf-segments` *elf* [Function]

`elf-section` *elf n* [Function]

`elf-sections` *elf* [Function]

`elf-section-by-name` *elf name* [Function]

`elf-sections-by-name` *elf* [Function]

`elf-symbol-table-len` *section* [Function]

`elf-symbol-table-ref` *elf section n* [*strtab*] [Function]

`parse-elf-note` *elf section* [Function]

`elf-note-name` [Special Form]

`elf-note-desc` [Special Form]

`elf-note-type` [Special Form]

`make-elf` [*#:bytes*] [*#:byte-order*] [*#:word-size*] [*#:abi*] [*#:type*] [*#:machine-type*] [*#:entry*] [*#:phoff*] [*#:shoff*] [*#:flags*] [*#:ehsize*] [*#:phentsize*] [*#:phnum*] [*#:shentsize*] [*#:shnum*] [*#:shstrndx*] [Function]

`make-elf-section` [*#:index*] [*#:name*] [*#:type*] [*#:flags*] [*#:addr*] [*#:offset*] [*#:size*] [*#:link*] [*#:info*] [*#:addralign*] [*#:entsize*] [Function]

`make-elf-segment` [*#:index*] [*#:type*] [*#:offset*] [*#:vaddr*] [*#:paddr*] [*#:filesz*] [*#:memsz*] [*#:flags*] [*#:align*] [Function]

`make-elf-symbol` [*#:name*] [*#:value*] [*#:size*] [*#:binding*] [*#:type*] [*#:info*] [*#:visibility*] [*#:other*] [*#:shndx*] [Function]

39 (guix ftp-client)

39.1 Overview

```
Simple FTP client (RFC 959).
```

39.2 Usage

`ftp-connection?` [Special Form]

`ftp-connection-addrinfo` [Special Form]

`connect*` *s sockaddr* [*timeout*] [Function]
: When TIMEOUT is omitted or #f, this procedure is equivalent to 'connect'. When TIMEOUT is a number, it is the (possibly inexact) maximum number of seconds to wait for the connection to succeed.

`ftp-open` *host* [*port*] [*#:timeout*] [*#:username*] [*#:password*] [Function]
: Open an FTP connection to HOST on PORT (a service-identifying string, or a TCP port number), and return it.

  When TIMEOUT is not #f, it must be a (possibly inexact) number denoting the maximum duration in seconds to wait for the connection to complete; passed TIMEOUT, an ETIMEDOUT error is raised.

`ftp-close` *conn* [Function]

`ftp-chdir` *conn dir* [Function]
: Change to directory DIR.

`ftp-size` *conn file* [Function]
: Return the size in bytes of FILE.

`ftp-list` *conn* [*directory*] [*#:timeout*] [Function]

`ftp-retr` *conn file* [*directory*] [*#:timeout*] [Function]
: Retrieve FILE from DIRECTORY (or, if omitted, the current directory) from FTP connection CONN. Return a binary port to that file. The returned port must be closed before CONN can be used for other purposes.

40 (guix gexp)

40.1 Overview

This module implements "G-expressions", or "gexps". Gexps are like S-expressions (sexps), with two differences:

1. References (un-quotations) to derivations or packages in a gexp are replaced by the corresponding output file name; in addition, the 'ungexp-native' unquote-like form allows code to explicitly refer to the native code of a given package, in case of cross-compilation;

2. Gexps embed information about the derivations they refer to.

Gexps make it easy to write to files Scheme code that refers to store items, or to write Scheme code to build derivations.

40.2 Usage

`gexp` [Special Form]

`gexp?` [Special Form]

`sexp->gexp` *sexp* [Function]
Turn SEXP into a gexp without any references.

Using this is a way for the caller to tell that SEXP doesn't need to be scanned for file-like objects, thereby reducing processing costs. This is particularly useful if SEXP is a long list or a deep tree.

`with-imported-modules` *modules body ...* [Special Form]
Mark the gexps defined in BODY... as requiring MODULES in their execution environment.

`with-extensions` *extensions body ...* [Special Form]
Mark the gexps defined in BODY... as requiring EXTENSIONS in their execution environment.

`let-system` (*system target*) *exp0 exp ...* [Special Form]
`let-system` *system exp0 exp ...* [Special Form]
Introduce a system binding in a gexp. The simplest form is:

```
(let-system system
  (cond ((string=? system "x86_64-linux") ...)
        (else ...)))
```

which binds SYSTEM to the currently targeted system. The second form is similar, but it also shows the cross-compilation target:

```
(let-system (system target)
  ...)
```

Here TARGET is bound to the cross-compilation triplet or #f.

`gexp->approximate-sexp` *gexp* [Function]
Return the S-expression corresponding to GEXP, but do not lower anything. As a result, the S-expression will be approximate if GEXP has references.

`gexp-input` *thing* [*output*] [*#:native?*] [Function]
Return a new <gexp-input> for the OUTPUT of THING; NATIVE? determines whether this should be considered a "native" input or not.

`gexp-input?` [Special Form]

`gexp-input-thing` [Special Form]

`gexp-input-output` [Special Form]

`gexp-input-native?` [Special Form]

`assume-valid-file-name` *file* [Special Form]
This is a syntactic keyword to tell 'local-file' that it can assume that the given file name is valid, even if it's not a string literal, and thus not warn about it.

`local-file` [Special Form]
Return an object representing local file FILE to add to the store; this object can be used in a gexp. If FILE is a relative file name, it is looked up relative to the source file where this form appears. FILE will be added to the store under NAME–by default the base name of FILE.

When RECURSIVE? is true, the contents of FILE are added recursively; if FILE designates a flat file and RECURSIVE? is true, its contents are added, and its permission bits are kept.

When RECURSIVE? is true, call (SELECT? FILE STAT) for each directory entry, where FILE is the entry's absolute file name and STAT is the result of 'lstat'; exclude entries for which SELECT? does not return true.

This is the declarative counterpart of the 'interned-file' monadic procedure. It is implemented as a macro to capture the current source directory where it appears.

`local-file?` [Special Form]

`local-file-file` [Special Form]

`local-file-absolute-file-name` *file* [Function]
Return the absolute file name for FILE, a <local-file> instance. A 'system-error' exception is raised if FILE could not be found.

`local-file-name` [Special Form]

`local-file-recursive?` [Special Form]

`local-file-select?` [Special Form]

`plain-file` *name content* [Function]
Return an object representing a text file called NAME with the given CONTENT (a string) to be added to the store.

This is the declarative counterpart of 'text-file'.

`plain-file?` [Special Form]

`plain-file-name` [Special Form]

`plain-file-content` [Special Form]

`computed-file` *name gexp* [*#:guile*] [*#:local-build?*] [*#:options*] [Function]
Return an object representing the store item NAME, a file or directory computed by GEXP. When LOCAL-BUILD? is #t (the default), it ensures the corresponding derivation is built locally. OPTIONS may be used to pass additional arguments to 'gexp->derivation'.

This is the declarative counterpart of 'gexp->derivation'.

`computed-file?` [Special Form]

`computed-file-name` [Special Form]

`computed-file-gexp` [Special Form]

`computed-file-options` [Special Form]

`program-file` *name gexp* [*#:guile*] [*#:module-path*] [Function]
Return an object representing the executable store item NAME that runs GEXP. GUILE is the Guile package used to execute that script. Imported modules of GEXP are looked up in MODULE-PATH.

This is the declarative counterpart of 'gexp->script'.

`program-file?` [Special Form]

`program-file-name` [Special Form]

`program-file-gexp` [Special Form]

`program-file-guile` [Special Form]

`program-file-module-path` [Special Form]

`scheme-file` *name gexp* [*#:splice?*] [*#:set-load-path?*] [Function]
Return an object representing the Scheme file NAME that contains GEXP.

This is the declarative counterpart of 'gexp->file'.

`scheme-file?` [Special Form]

`scheme-file-name` [Special Form]

`scheme-file-gexp` [Special Form]

`file-append` *base . suffix* [Function]
Return a <file-append> object that expands to the concatenation of BASE and SUFFIX.

`file-append?` [Special Form]

`file-append-base` [Special Form]

`file-append-suffix` [Special Form]

`raw-derivation-file` [Special Form]

`raw-derivation-file?` [Special Form]

`with-parameters` ((*param value*) ...) *body ...* [Special Form]

```
Bind each PARAM to the corresponding VALUE for the extent during which BODY
is lowered.  Consider this example:

  (with-parameters ((%current-system "x86_64-linux"))
    coreutils)

It returns a <parameterized> object that ensures %CURRENT-SYSTEM is set to
x86_64-linux when COREUTILS is lowered.
```

`parameterized?` [Special Form]

`load-path-expression` *modules* [*path*] [*#:extensions*] [*#:system*] [*#:target*] [*#:guile*] [Function]

Return as a monadic value a gexp that sets '%load-path' and '%load-compiled-path' to point to MODULES, a list of module names. MODULES are searched for in PATH. Return #f when MODULES and EXTENSIONS are empty. Assume MODULES are compiled with GUILE.

`gexp-modules` *gexp* [Function]

Return the list of Guile module names GEXP relies on. If (gexp? GEXP) is false, meaning that GEXP is a plain Scheme object, return the empty list.

`lower-gexp` *exp* [*#:module-path*] [*#:system*] [*#:target*] [*#:graft?*] [*#:guile-for-build*] [*#:effective-version*] [*#:deprecation-warnings*] [Function]

Note: This API is subject to change; use at your own risk!

Lower EXP, a gexp, instantiating it for SYSTEM and TARGET. Return a <lowered-gexp> ready to be used.

Lowered gexps are an intermediate representation that's useful for applications that deal with gexps outside in a way that is disconnected from derivations–e.g., code evaluated for its side effects.

`lowered-gexp?` [Special Form]

`lowered-gexp-sexp` [Special Form]

`lowered-gexp-inputs` [Special Form]

`lowered-gexp-sources` [Special Form]

`lowered-gexp-guile` [Special Form]

`lowered-gexp-load-path` [Special Form]

`lowered-gexp-load-compiled-path` [Special Form]

`with-build-variables` *inputs outputs body* [Function]
Return a gexp that surrounds BODY with a definition of the legacy '%build-inputs', '%outputs', and '%output' variables based on INPUTS, a list of name/gexp-input tuples, and OUTPUTS, a list of strings.

`input-tuples->gexp` *inputs* [*#:native?*] [Function]
Given INPUTS, a list of label/gexp-input tuples, return a gexp that expands to an input alist.

`outputs->gexp` *outputs* [Function]
Given OUTPUTS, a list of output names, return a gexp that expands to an output alist.

`gexp->derivation` *name exp* [*#:system*] [*#:target*] [*#:hash*] [Function]
[*#:hash-algo*] [*#:recursive?*] [*#:env-vars*] [*#:modules*] [*#:module-path*]
[*#:guile-for-build*] [*#:effective-version*] [*#:graft?*] [*#:references-graphs*]
[*#:allowed-references*] [*#:disallowed-references*] [*#:leaked-env-vars*]
[*#:local-build?*] [*#:substitutable?*] [*#:properties*]
[*#:deprecation-warnings*] [*#:script-name*]

```
Return a derivation NAME that runs EXP (a gexp) with GUILE-FOR-BUILD (a
derivation) on SYSTEM; EXP is stored in a file called SCRIPT-NAME.  When
TARGET is true, it is used as the cross-compilation target triplet for
packages referred to by EXP.

MODULES is deprecated in favor of 'with-imported-modules'.  Its meaning is to
make MODULES available in the evaluation context of EXP; MODULES is a list of
names of Guile modules searched in MODULE-PATH to be copied in the store,
compiled, and made available in the load path during the execution of
EXP---e.g., '((guix build utils) (guix build gnu-build-system)).

EFFECTIVE-VERSION determines the string to use when adding extensions of
EXP (see 'with-extensions') to the search path---e.g., "2.2".

GRAFT? determines whether packages referred to by EXP should be grafted when
applicable.

When REFERENCES-GRAPHS is true, it must be a list of tuples of one of the
following forms:

  (FILE-NAME PACKAGE)
  (FILE-NAME PACKAGE OUTPUT)
  (FILE-NAME DERIVATION)
  (FILE-NAME DERIVATION OUTPUT)
  (FILE-NAME STORE-ITEM)

The right-hand-side of each element of REFERENCES-GRAPHS is automatically made
```

an input of the build process of EXP. In the build environment, each FILE-NAME contains the reference graph of the corresponding item, in a simple text format.

ALLOWED-REFERENCES must be either #f or a list of output names and packages. In the latter case, the list denotes store items that the result is allowed to refer to. Any reference to another store item will lead to a build error. Similarly for DISALLOWED-REFERENCES, which can list items that must not be referenced by the outputs.

DEPRECATION-WARNINGS determines whether to show deprecation warnings while compiling modules. It can be #f, #t, or 'detailed.

The other arguments are as for 'derivation'.

`gexp->file` *name exp* [*#:set-load-path?*] [*#:module-path*] [*#:splice?*] [*#:system*] [*#:target*] [Function]

Return a derivation that builds a file NAME containing EXP. When SPLICE? is true, EXP is considered to be a list of expressions that will be spliced in the resulting file.

When SET-LOAD-PATH? is true, emit code in the resulting file to set '%load-path' and '%load-compiled-path' to honor EXP's imported modules. Lookup EXP's modules in MODULE-PATH.

`gexp->script` *name exp* [*#:guile*] [*#:module-path*] [*#:system*] [*#:target*] [Function]

Return an executable script NAME that runs EXP using GUILE, with EXP's imported modules in its search path. Look up EXP's modules in MODULE-PATH.

`text-file*` *name . text* [Function]

Return as a monadic value a derivation that builds a text file containing all of TEXT. TEXT may list, in addition to strings, objects of any type that can be used in a gexp: packages, derivations, local file objects, etc. The resulting store file holds references to all these.

`mixed-text-file` *name* [*#:guile*] *. text* [Function]

Return an object representing store file NAME containing TEXT. TEXT is a sequence of strings and file-like objects, as in:

```
(mixed-text-file "profile"
                 "export PATH=" coreutils "/bin:" grep "/bin")
```

This is the declarative counterpart of 'text-file*'.

`file-union` *name files* [*#:guile*] [Function]

Return a <computed-file> that builds a directory containing all of FILES. Each item in FILES must be a two-element list where the first element is the file name to use in the new directory, and the second element is a gexp denoting the target file. Here's an example:

```
(file-union "etc"
            `(("hosts" ,(plain-file "hosts"
                                    "127.0.0.1 localhost"))
              ("bashrc" ,(plain-file "bashrc"
                                     "alias ls='ls --color'"))
              ("libvirt/qemu.conf" ,(plain-file "qemu.conf" ""))))
```

This yields an 'etc' directory containing these two files.

`directory-union` *name things* [*#:copy?*] [*#:quiet?*] [*#:resolve-collision*] [Function]

Return a directory that is the union of THINGS, where THINGS is a list of file-like objects denoting directories. For example:

```
(directory-union "guile+emacs" (list guile emacs))
```

yields a directory that is the union of the 'guile' and 'emacs' packages.

Call RESOLVE-COLLISION when several files collide, passing it the list of colliding files. RESOLVE-COLLISION must return the chosen file or #f, in which case the colliding entry is skipped altogether.

When COPY? is true, copy files instead of creating symlinks. When QUIET? is true, the derivation will not print anything.

`references-file` *item* [*name*] [*#:guile*] [Function]

Return a file that contains the list of direct and indirect references (the closure) of ITEM.

`imported-files` *files* [*#:name*] [*#:system*] [*#:guile*] [Function]

Import FILES into the store and return the resulting derivation or store file name (a derivation is created if and only if some elements of FILES are file-like objects and not local file names.) FILES must be a list of (FINAL-PATH . FILE) pairs. Each FILE is mapped to FINAL-PATH in the resulting store path. FILE can be either a file name, or a file-like object, as returned by 'local-file' for example.

`imported-modules` *modules* [*#:name*] [*#:system*] [*#:guile*] [*#:module-path*] [Function]

Return a derivation that contains the source files of MODULES, a list of module names such as `(ice-9 q)'. All of MODULES must be either names of modules to be found in the MODULE-PATH search path, or a module name followed by an arrow followed by a file-like object. For example:

```
(imported-modules `((guix build utils)
                    (guix gcrypt)
                    ((guix config) => ,(scheme-file ...))))
```

In this example, the first two modules are taken from MODULE-PATH, and the last one is created from the given <scheme-file> object.

`compiled-modules` *modules* [*#:name*] [*#:system*] [*#:target*] [*#:guile*] [*#:module-path*] [*#:extensions*] [*#:deprecation-warnings*] [*#:optimization-level*] [Function]

Return a derivation that builds a tree containing the '.go' files corresponding to MODULES. All the MODULES are built in a context where they can refer to each other. When TARGET is true, cross-compile MODULES for TARGET, a GNU triplet.

`define-gexp-compiler` (*name* (*param record-type*) *system target*) *body ...* [Special Form]

`define-gexp-compiler` *name record-type compiler* => *compile expander* => *expand* [Special Form]

Define NAME as a compiler for objects matching PREDICATE encountered in gexps.

In the simplest form of the macro, BODY must return (1) a derivation for a record of the specified type, for SYSTEM and TARGET (the latter of which is #f except when cross-compiling), (2) another record that can itself be compiled down to a derivation, or (3) an object of a primitive data type.

The more elaborate form allows you to specify an expander:

```
(define-gexp-compiler something-compiler <something>
  compiler => (lambda (param system target) ...)
  expander => (lambda (param drv output) ...))
```

The expander specifies how an object is converted to its sexp representation.

`gexp-compiler?` [Special Form]

`file-like?` *object* [Function]

Return #t if OBJECT leads to a file in the store once unquoted in a G-expression; otherwise return #f.

`lower-object` *obj* [*system*] [*#:target*] [Function]

Return as a value in %STORE-MONAD the derivation or store item corresponding to OBJ for SYSTEM, cross-compiling for TARGET if TARGET is true. OBJ must be an object that has an associated gexp compiler, such as a <package>.

`&gexp-error` [Variable]

`gexp-error?` *obj* [Function]

`&gexp-input-error` [Variable]

`gexp-input-error?` *obj* [Function]

`gexp-error-invalid-input` *obj* [Function]

41 (guix git-authenticate)

41.1 Overview

```
This module provides tools to authenticate a range of Git commits.  A
commit is considered "authentic" if and only if it is signed by an
authorized party.  Parties authorized to sign a commit are listed in the
'.guix-authorizations' file of the parent commit.
```

41.2 Usage

`read-authorizations` *port* [Function]

Read authorizations in the '.guix-authorizations' format from PORT, and return a list of authorized fingerprints.

`commit-signing-key` *repo commit-id keyring* [Function]
[*#:disallowed-hash-algorithms*]

Return the OpenPGP key that signed COMMIT-ID (an OID). Raise an exception if the commit is unsigned, has an invalid signature, has a signature using one of the hash algorithms in DISALLOWED-HASH-ALGORITHMS, or if its signing key is not in KEYRING.

`commit-authorized-keys` *repository commit* [*default-authorizations*] [Function]

Return the list of OpenPGP fingerprints authorized to sign COMMIT, based on authorizations listed in its parent commits. If one of the parent commits does not specify anything, fall back to DEFAULT-AUTHORIZATIONS.

`authenticate-commit` *repository commit keyring* [Function]
[*#:default-authorizations*]

Authenticate COMMIT from REPOSITORY and return the signing key fingerprint. Raise an error when authentication fails. If one of the parent commits does not specify anything, fall back to DEFAULT-AUTHORIZATIONS.

`authenticate-commits` *repository commits* [Function]
[*#:default-authorizations*] [*#:keyring-reference*] [*#:keyring*]
[*#:report-progress*]

Authenticate COMMITS, a list of commit objects, calling REPORT-PROGRESS for each of them. Return an alist showing the number of occurrences of each key. If KEYRING is omitted, the OpenPGP keyring is loaded from KEYRING-REFERENCE in REPOSITORY.

`load-keyring-from-reference` *repository reference* [Function]

Load the '.key' files from the tree at REFERENCE in REPOSITORY and return an OpenPGP keyring.

`previously-authenticated-commits` *key* [Function]

Return the previously-authenticated commits under KEY as a list of commit IDs (hex strings).

`cache-authenticated-commit` *key commit-id* [Function]
Record in ~/.cache, under KEY, COMMIT-ID and its closure as authenticated (only COMMIT-ID is written to cache, though).

`repository-cache-key` *repository* [Function]
Return a unique key to store the authenticate commit cache for REPOSITORY.

`authenticate-repository` *repository start signer* [Function]
[*#:keyring-reference*] [*#:cache-key*] [*#:end*] [*#:authentic-commits*] [*#:historical-authorizations*] [*#:make-reporter*]
Authenticate REPOSITORY up to commit END, an OID. Authentication starts with commit START, an OID, which must be signed by SIGNER; an exception is raised if that is not the case. Commits listed in AUTHENTIC-COMMITS and their closure are considered authentic. Return an alist mapping OpenPGP public keys to the number of commits signed by that key that have been traversed.

The OpenPGP keyring is loaded from KEYRING-REFERENCE in REPOSITORY, where KEYRING-REFERENCE is the name of a branch. The list of authenticated commits is cached in the authentication cache under CACHE-KEY.

HISTORICAL-AUTHORIZATIONS must be a list of OpenPGP fingerprints (bytevectors) denoting the authorized keys for commits whose parent lack the '.guix-authorizations' file.

`git-authentication-error?` *obj* [Function]

`git-authentication-error-commit` *obj* [Function]

`unsigned-commit-error?` *obj* [Function]

`unauthorized-commit-error?` *obj* [Function]

`unauthorized-commit-error-signing-key` *obj* [Function]

`signature-verification-error?` *obj* [Function]

`signature-verification-error-keyring` *obj* [Function]

`signature-verification-error-signature` *obj* [Function]

`missing-key-error?` *obj* [Function]

`missing-key-error-signature` *obj* [Function]

42 (guix git-download)

42.1 Overview

An <origin> method that fetches a specific commit from a Git repository. The repository URL and commit hash are specified with a <git-reference> object.

42.2 Usage

`git-reference` [Special Form]

`git-reference?` [Special Form]

`git-reference-url` [Special Form]

`git-reference-commit` [Special Form]

`git-reference-recursive?` [Special Form]

`git-fetch` *ref hash-algo hash* [*name*] [*#:system*] [*#:guile*] [*#:git*] [Function]
Return a fixed-output derivation that fetches REF, a <git-reference> object. The output is expected to have recursive hash HASH of type HASH-ALGO (a symbol). Use NAME as the file name, or a generic name if #f.

`git-version` *version revision commit* [Function]
Return the version string for packages using git-download.

`git-file-name` *name version* [Function]
Return the file-name for packages using git-download.

`git-predicate` *directory* [*#:recursive?*] [Function]
Return a predicate that returns true if a file is part of the Git checkout living at DIRECTORY. If DIRECTORY does not lie within a Git checkout, and upon Git errors, return #f instead of a predicate.

When RECURSIVE? is true, the predicate also returns true if a file is part of any Git submodule under DIRECTORY. This is enabled by default.

The returned predicate takes two arguments FILE and STAT where FILE is an absolute file name and STAT is the result of 'lstat'.

43 (guix git)

43.1 Overview

43.2 Usage

`%repository-cache-directory` [Variable]

`honor-system-x509-certificates!` [Function]
Use the system's X.509 certificates for Git checkouts over HTTPS. Honor the 'SSL_CERT_FILE' and 'SSL_CERT_DIR' environment variables.

`url-cache-directory` *url* [*cache-directory*] [*#:recursive?*] [Function]
Return the directory associated to URL in %repository-cache-directory.

`with-repository` *directory repository exp ...* [Special Form]
Open the repository at DIRECTORY and bind REPOSITORY to it within the dynamic extent of EXP.

`with-git-error-handling` *body ...* [Special Form]

`false-if-git-not-found` *exp* [Special Form]
Evaluate EXP, returning #false if a GIT_ENOTFOUND error is raised.

`update-cached-checkout` *url* [*#:ref*] [*#:recursive?*] [*#:check-out?*] [Function]
[*#:starting-commit*] [*#:log-port*] [*#:cache-directory*]
Update the cached checkout of URL to REF in CACHE-DIRECTORY. Return three values: the cache directory name, and the SHA1 commit (a string) corresponding to REF, and the relation of the new commit relative to STARTING-COMMIT (if provided) as returned by 'commit-relation'.

REF is pair whose key is [branch | commit | tag | tag-or-commit] and value the associated data: [<branch name> | <sha1> | <tag name> | <string>]. If REF is the empty list, the remote HEAD is used.

When RECURSIVE? is true, check out submodules as well, if any.

When CHECK-OUT? is true, reset the cached working tree to REF; otherwise leave it unchanged.

`url+commit->name` *url sha1* [Function]
Return the string "<REPO-NAME>-<SHA1:7>" where REPO-NAME is the name of the git repository, extracted from URL and SHA1:7 the seven first digits of SHA1 string.

`latest-repository-commit` *store url* [*#:recursive?*] [*#:log-port*] [Function]
[*#:cache-directory*] [*#:ref*]
Return two values: the content of the git repository at URL copied into a store directory and the sha1 of the top level commit in this directory. The reference to be checkout, once the repository is fetched, is specified by REF. REF is pair whose

key is [branch | commit | tag] and value the associated data, respectively [<branch name> | <sha1> | <tag name>]. If REF is the empty list, the remote HEAD is used.

When RECURSIVE? is true, check out submodules as well, if any.

Git repositories are kept in the cache directory specified by %repository-cache-directory parameter.

Log progress and checkout info to LOG-PORT.

`commit-difference` *new old* [*excluded*] [Function]
Return the list of commits between NEW and OLD, where OLD is assumed to be an ancestor of NEW. Exclude all the commits listed in EXCLUDED along with their ancestors.

Essentially, this computes the set difference between the closure of NEW and that of OLD.

`commit-relation` *old new* [Function]
Return a symbol denoting the relation between OLD and NEW, two commit objects: 'ancestor (meaning that OLD is an ancestor of NEW), 'descendant, or 'unrelated, or 'self (OLD and NEW are the same commit).

`commit-descendant?` *new old* [Function]
Return true if NEW is the descendant of one of OLD, a list of commits.

When the expected result is likely #t, this is faster than using 'commit-relation' since fewer commits need to be traversed.

`commit-id?` *str* [Function]
Return true if STR is likely a Git commit ID, false otherwise—e.g., if it is a tag name. This is based on a simple heuristic so use with care!

`remote-refs` *url* [*#:tags?*] [Function]
Return the list of references advertised at Git repository URL. If TAGS? is true, limit to only refs/tags.

`git-checkout` [Special Form]

`git-checkout?` [Special Form]

`git-checkout-url` [Special Form]

`git-checkout-branch` [Special Form]

`git-checkout-commit` [Special Form]

`git-checkout-recursive?` [Special Form]

`git-reference->git-checkout` *reference* [Function]
Convert the <git-reference> REFERENCE to an equivalent <git-checkout>.

44 (guix glob)

44.1 Overview

```
This is a minimal implementation of "glob patterns" (info "(libc)
Globbbing").  It is currently limited to simple patterns and does not
support braces, for instance.
```

44.2 Usage

`string->sglob` *str* [Function]
: Return an sexp, called an "sglob", that represents the compiled form of STR, a glob pattern such as "foo*" or "foo??bar".

`compile-sglob` *sglob* [Function]
: Compile SGLOB into a more efficient representation.

`string->compiled-sglob` *. args* [Function]

`glob-match?` *pattern str* [Function]
: Return true if STR matches PATTERN, a compiled glob pattern as returned by 'compile-sglob'.

45 (guix gnu-maintenance)

45.1 Overview

Code for dealing with the maintenance of GNU packages, such as auto-updates.

45.2 Usage

`gnu-package-name` [Special Form]

`gnu-package-mundane-name` [Special Form]

`gnu-package-copyright-holder` [Special Form]

`gnu-package-savannah` [Special Form]

`gnu-package-fsd` [Special Form]

`gnu-package-language` [Special Form]

`gnu-package-logo` [Special Form]

`gnu-package-doc-category` [Special Form]

`gnu-package-doc-summary` [Special Form]

`gnu-package-doc-description` [Special Form]

`gnu-package-doc-urls` [Special Form]

`gnu-package-download-url` [Special Form]

`official-gnu-packages` [*fetch*] [Function]
Return a list of records, which are GNU packages. Use FETCH, to fetch the list of GNU packages over HTTP.

`find-package` *name* [Function]
Find GNU package called NAME and return it. Return #f if it was not found.

`gnu-package?` *package* [Function]

`uri-mirror-rewrite` *uri* [Function]
Rewrite URI to a mirror:// URI if possible, or return URI unmodified.

`release-file?` *project file* [Function]
Return #f if FILE is not a release tarball of PROJECT, otherwise return true.

`releases` *project* [*#:server*] [*#:directory*] [Function]
Return the list of <upstream-release> of PROJECT as a list of release name/directory pairs.

`latest-release` *package* [*#:server*] [*#:directory*] [Function]
Return the <upstream-source> for the latest version of PACKAGE or #f. PACKAGE must be the canonical name of a GNU package.

`gnu-release-archive-types` [Variable]
[unbound!]

`gnu-package-name->name+version` *name+version* [Function]
Return the package name and version number extracted from NAME+VERSION.

`%gnu-updater` [Variable]

`%gnu-ftp-updater` [Variable]

`%savannah-updater` [Variable]

`%sourceforge-updater` [Variable]

`%xorg-updater` [Variable]

`%kernel.org-updater` [Variable]

`%generic-html-updater` [Variable]

46 (guix gnupg)

46.1 Overview

`GnuPG interface.`

46.2 Usage

`%gpg-command` [Variable]

`%openpgp-key-server` [Variable]

`current-keyring` [Variable]

`gnupg-verify` *sig file* [*keyring*] [Function]
Verify signature SIG for FILE against the keys in KEYRING. All the keys in KEYRING as assumed to be "trusted", whether or not they expired or were revoked. Return a status s-exp if GnuPG failed.

`gnupg-verify*` *sig file* [*#:key-download*] [*#:server*] [*#:keyring*] [Function]
Like 'gnupg-verify', but try downloading the public key if it's missing. Return two values: 'valid-signature and a fingerprint/name pair upon success, 'missing-key and a fingerprint if the key could not be found, and 'invalid-signature with a fingerprint if the signature is invalid.

KEY-DOWNLOAD specifies a download policy for missing OpenPGP keys; allowed values: 'always', 'never', and 'interactive' (default). Return a fingerprint/user name pair on success and #f otherwise.

`gnupg-status-good-signature?` *status* [Function]
If STATUS, as returned by 'gnupg-verify', denotes a good signature, return a fingerprint/user pair; return #f otherwise.

`gnupg-status-missing-key?` *status* [Function]
If STATUS denotes a missing-key error, then return the fingerprint of the missing key or its key id if the fingerprint is unavailable.

47 (guix grafts)

47.1 Overview

47.2 Usage

`graft?` [Special Form]

`graft` [Special Form]

`graft-origin` [Special Form]

`graft-replacement` [Special Form]

`graft-origin-output` [Special Form]

`graft-replacement-output` [Special Form]

`graft-derivation` *store drv grafts* [*#:guile*] [*#:outputs*] [*#:system*] [Function]
Apply GRAFTS to the OUTPUTS of DRV and all their dependencies, recursively. That is, if GRAFTS apply only indirectly to DRV, graft the dependencies of DRV, and graft DRV itself to refer to those grafted dependencies.

`graft-derivation/shallow` *drv grafts* [*#:name*] [*#:outputs*] [*#:guile*] [*#:system*] [Function]
Return a derivation called NAME, which applies GRAFTS to the specified OUTPUTS of DRV. This procedure performs "shallow" grafting in that GRAFTS are not recursively applied to dependencies of DRV.

`%graft-with-utf8-locale?` [Variable]

`%graft?` [Variable]

`grafting?` [Special Form]

`set-grafting` [Special Form]

`without-grafting` *mexp ...* [Special Form]
Bind monadic expressions MEXP in a dynamic extent where '%graft?' is false.

48 (guix graph)

48.1 Overview

This module provides an abstract way to represent graphs and to manipulate them. It comes with several such representations for packages, derivations, and store items. It also provides a generic interface for exporting graphs in an external format, including a Graphviz implementation thereof.

48.2 Usage

`node-type` [Special Form]

`node-type?` [Special Form]

`node-type-identifier` [Special Form]

`node-type-label` [Special Form]

`node-type-edges` [Special Form]

`node-type-convert` [Special Form]

`node-type-name` [Special Form]

`node-type-description` [Special Form]

`node-edges` *type nodes* [Function]
Return, as a monadic value, a one-argument procedure that, given a node of TYPE, returns its edges. NODES is taken to be the sinks of the global graph.

`node-back-edges` *type nodes* [Function]
Return, as a monadic value, a one-argument procedure that, given a node of TYPE, returns its back edges. NODES is taken to be the sinks of the global graph.

`traverse/depth-first` *proc seed nodes node-edges* [Function]
Do a depth-first traversal of NODES along NODE-EDGES, calling PROC with each node and the current result, and visiting each reachable node exactly once. NODES must be a list of nodes, and NODE-EDGES must be a one-argument procedure as returned by 'node-edges' or 'node-back-edges'.

`node-transitive-edges` *nodes node-edges* [Function]
Return the list of nodes directly or indirectly connected to NODES according to the NODE-EDGES procedure. NODE-EDGES must be a one-argument procedure that, given a node, returns its list of direct dependents; it is typically returned by 'node-edges' or 'node-back-edges'.

`node-reachable-count` *nodes node-edges* [Function]
Return the number of nodes reachable from NODES along NODE-EDGES.

`shortest-path` *node1 node2 type* [Function]
Return as a monadic value the shortest path, represented as a list, from NODE1 to NODE2 of the given TYPE. Return #f when there is no path.

`%graph-backends` [Variable]

`%d3js-backend` [Variable]

`%graphviz-backend` [Variable]

`lookup-backend` *name* [Function]
Return the graph backend called NAME. Raise an error if it is not found.

`graph-backend?` [Special Form]

`graph-backend` [Special Form]

`graph-backend-name` [Special Form]

`graph-backend-description` [Special Form]

`export-graph` *sinks port* [*#:reverse-edges?*] [*#:node-type*] [*#:max-depth*] [*#:backend*] [Function]
Write to PORT the representation of the DAG with the given SINKS, using the given BACKEND. Use NODE-TYPE to traverse the DAG. When REVERSE-EDGES? is true, draw reverse arrows. Do not represent nodes whose distance to one of the SINKS is greater than MAX-DEPTH.

49 (guix hash)

49.1 Overview

49.2 Usage

`vcs-file?` *file stat* [Function]
Returns true if FILE is a version control system file.

`file-hash*` *file* [*#:algorithm*] [*#:recursive?*] [*#:select?*] [Function]
Compute the hash of FILE with ALGORITHM.

Symbolic links are only dereferenced if RECURSIVE? is false. Directories are only supported if RECURSIVE? is #true or 'auto'. The executable bit is only recorded if RECURSIVE? is #true. If FILE is a symbolic link, it is only followed if RECURSIVE? is false.

For regular files, there are two different hashes when the executable hash isn't recorded: the regular hash and the nar hash. In most situations, the regular hash is desired and setting RECURSIVE? to 'auto' does the right thing for both regular files and directories.

This procedure must only be used under controlled circumstances; the detection of symbolic links in FILE is racy.

When FILE is a directory, the procedure SELECT? called as (SELECT? FILE STAT) decides which files to include. By default, version control files are excluded. To include everything, SELECT? can be set to (const #true).

50 (guix hg-download)

50.1 Overview

An <origin> method that fetches a specific changeset from a Mercurial repository. The repository URL and changeset ID are specified with a <hg-reference> object.

50.2 Usage

`hg-reference` [Special Form]

`hg-reference?` [Special Form]

`hg-reference-url` [Special Form]

`hg-reference-changeset` [Special Form]

`hg-predicate` *directory* [Function]
This procedure evaluates to a predicate that reports back whether a given *file* - *stat* combination is part of the files tracked by Mercurial.

`hg-fetch` *ref hash-algo hash* [*name*] [*#:system*] [*#:guile*] [*#:hg*] [Function]
Return a fixed-output derivation that fetches REF, a <hg-reference> object. The output is expected to have recursive hash HASH of type HASH-ALGO (a symbol). Use NAME as the file name, or a generic name if #f.

`hg-version` *version revision changeset* [Function]
Return the version string for packages using hg-download.

`hg-file-name` *name version* [Function]
Return the file-name for packages using hg-download.

51 (guix http-client)

51.1 Overview

HTTP client portable among Guile versions, and with proper error condition reporting.

51.2 Usage

`&http-get-error` [Variable]

`http-get-error?` *obj* [Function]

`http-get-error-uri` *obj* [Function]

`http-get-error-code` *obj* [Function]

`http-get-error-reason` *obj* [Function]

`http-get-error-headers` *obj* [Function]

`http-fetch` *uri* [*#:port*] [*#:text?*] [*#:buffered?*] [*#:open-connection*] [*#:keep-alive?*] [*#:verify-certificate?*] [*#:headers*] [*#:log-port*] [*#:timeout*] [Function]

Return an input port containing the data at URI, and the expected number of bytes available or #f. If TEXT? is true, the data at URI is considered to be textual. Follow any HTTP redirection. When BUFFERED? is #f, return an unbuffered port, suitable for use in 'filtered-port'. HEADERS is an alist of extra HTTP headers.

When KEEP-ALIVE? is true, the connection is marked as 'keep-alive' and PORT is not closed upon completion.

When VERIFY-CERTIFICATE? is true, verify HTTPS server certificates.

TIMEOUT specifies the timeout in seconds for connection establishment; when TIMEOUT is #f, connection establishment never times out.

Write information about redirects to LOG-PORT.

Raise an '&http-get-error' condition if downloading fails.

`http-multiple-get` *base-uri proc seed requests* [*#:port*] [*#:verify-certificate?*] [*#:open-connection*] [*#:keep-alive?*] [*#:batch-size*] [Function]

Send all of REQUESTS to the server at BASE-URI. Call PROC for each response, passing it the request object, the response, a port from which to read the response body, and the previous result, starting with SEED, à la 'fold'. Return the final result.

When PORT is specified, use it as the initial connection on which HTTP requests are sent; otherwise call OPEN-CONNECTION to open a new connection for a URI. When KEEP-ALIVE? is false, close the connection port before returning.

`%http-cache-ttl` [Variable]

`http-fetch/cached` *uri* [*#:ttl*] [*#:text?*] [*#:headers*] [*#:write-cache*] [*#:cache-miss*] [*#:log-port*] [*#:timeout*] [Function]

Like 'http-fetch', return an input port, but cache its contents in ~/.cache/guix. The cache remains valid for TTL seconds.

Call WRITE-CACHE with the HTTP input port and the cache output port to write the data to cache. Call CACHE-MISS with URI just before fetching data from URI.

HEADERS is an alist of extra HTTP headers, to which cache-related headers are added automatically as appropriate.

TIMEOUT specifies the timeout in seconds for connection establishment.

Write information about redirects to LOG-PORT.

`open-socket-for-uri` *uri-or-string* [*#:timeout*] [Function]

Return an open input/output port for a connection to URI. When TIMEOUT is not #f, it must be a (possibly inexact) number denoting the maximum duration in seconds to wait for the connection to complete; passed TIMEOUT, an ETIMEDOUT error is raised.

52 (guix i18n)

52.1 Overview

`Internationalization support.`

52.2 Usage

`G_` *t-13c8accbfea35c4d-1d* [Function]

`N_` *t-13c8accbfea35c4d-24 t-13c8accbfea35c4d-25 t-13c8accbfea35c4d-26* [Function]

`P_` *msgid* [Function]
Return the translation of the package description or synopsis MSGID.

`%gettext-domain` [Variable]

`%package-text-domain` [Variable]

53 (guix inferior)

53.1 Overview

```
This module provides a way to spawn Guix "inferior" processes and to talk
to them.  It allows us, from one instance of Guix, to interact with
another instance of Guix coming from a different commit.
```

53.2 Usage

`inferior?` [Special Form]

`open-inferior` *directory* [*#:command*] [*#:error-port*] [Function]
Open the inferior Guix in DIRECTORY, running 'DIRECTORY/COMMAND repl' or equivalent. Return #f if the inferior could not be launched.

`port->inferior` *pipe* [*close*] [Function]
Given PIPE, an input/output port, return an inferior that talks over PIPE. PIPE is closed with CLOSE when 'close-inferior' is called on the returned inferior.

`close-inferior` *inferior* [Function]
Close INFERIOR.

`inferior-eval` *exp inferior* [Function]
Evaluate EXP in INFERIOR.

`inferior-eval-with-store` *inferior store code* [Function]
Evaluate CODE in INFERIOR, passing it STORE as its argument. CODE must thus be the code of a one-argument procedure that accepts a store.

`inferior-object?` [Special Form]

`inferior-exception?` *obj* [Function]

`inferior-exception-arguments` *obj* [Function]

`inferior-exception-inferior` *obj* [Function]

`inferior-exception-stack` *obj* [Function]

`inferior-protocol-error?` *obj* [Function]

`inferior-protocol-error-inferior` *obj* [Function]

`read-repl-response` *port* [*inferior*] [Function]
Read a (guix repl) response from PORT and return it as a Scheme object. Raise '&inferior-exception' when an exception is read from PORT.

`inferior-packages` *inferior* [Function]
Return the list of packages known to INFERIOR.

`inferior-available-packages` *inferior* [Function]
Return the list of name/version pairs corresponding to the set of packages available in INFERIOR.

This is faster and less resource-intensive than calling 'inferior-packages'.

`lookup-inferior-packages` *inferior name* [*version*] [Function]
Return the sorted list of inferior packages matching NAME in INFERIOR, with highest version numbers first. If VERSION is true, return only packages with a version number prefixed by VERSION.

`inferior-package?` [Special Form]

`inferior-package-name` [Special Form]

`inferior-package-version` [Special Form]

`inferior-package-synopsis` *package* [*#:translate?*] [Function]
Return the Texinfo synopsis of PACKAGE, an inferior package. When TRANSLATE? is true, translate it to the current locale's language.

`inferior-package-description` *package* [*#:translate?*] [Function]
Return the Texinfo description of PACKAGE, an inferior package. When TRANSLATE? is true, translate it to the current locale's language.

`inferior-package-home-page` *package* [Function]
Return the home page of PACKAGE.

`inferior-package-location` *package* [Function]
Return the source code location of PACKAGE, either #f or a <location> record.

`inferior-package-inputs` *t-f7bdf8847bdd73-2536* [Function]

`inferior-package-native-inputs` *t-f7bdf8847bdd73-253d* [Function]

`inferior-package-propagated-inputs` *t-f7bdf8847bdd73-2544* [Function]

`inferior-package-transitive-propagated-inputs` [Function]
t-f7bdf8847bdd73-254b

`inferior-package-native-search-paths` *t-f7bdf8847bdd73-255a* [Function]

`inferior-package-transitive-native-search-paths` [Function]
t-f7bdf8847bdd73-2568

`inferior-package-search-paths` *t-f7bdf8847bdd73-2561* [Function]

`inferior-package-replacement` *package* [Function]
Return the replacement for PACKAGE. This will either be an inferior package, or #f.

`inferior-package-provenance` *package* [Function]
Return a "provenance sexp" for PACKAGE, an inferior package. The result is similar to the sexp returned by 'package-provenance' for regular packages.

`inferior-package-derivation` *store package* [*system*] [*#:target*] [Function]
Return the derivation for PACKAGE, an inferior package, built for SYSTEM and cross-built for TARGET if TARGET is true. The inferior corresponding to PACKAGE must be live.

`inferior-package->manifest-entry` *package* [*output*] [*#:properties*] [Function]

Return a manifest entry for the OUTPUT of package PACKAGE.

`gexp->derivation-in-inferior` *name exp guix* [*#:silent-failure?*] . *rest* [Function]

Return a derivation that evaluates EXP with GUIX, an instance of Guix as returned for example by 'channel-instances->derivation'. Other arguments are passed as-is to 'gexp->derivation'.

When SILENT-FAILURE? is true, create an empty output directory instead of failing when GUIX is too old and lacks the 'guix repl' command.

`%inferior-cache-directory` [Variable]

`cached-channel-instance` *store channels* [*#:authenticate?*] [*#:cache-directory*] [*#:ttl*] [Function]

Return a directory containing a guix filetree defined by CHANNELS, a list of channels. The directory is a subdirectory of CACHE-DIRECTORY, where entries can be reclaimed after TTL seconds. This procedure opens a new connection to the build daemon. AUTHENTICATE? determines whether CHANNELS are authenticated.

`inferior-for-channels` *channels* [*#:cache-directory*] [*#:ttl*] [Function]

Return an inferior for CHANNELS, a list of channels. Use the cache at CACHE-DIRECTORY, where entries can be reclaimed after TTL seconds. This procedure opens a new connection to the build daemon.

This is a convenience procedure that people may use in manifests passed to 'guix package -m', for instance.

54 (guix ipfs)

54.1 Overview

This module implements bindings for the HTTP interface of the IPFS gateway, documented here: <https://docs.ipfs.io/reference/api/http/>. It allows you to add and retrieve files over IPFS, and a few other things.

54.2 Usage

`%ipfs-base-url` [Variable]

`add-data` *data* [*#:name*] [*#:recursive?*] [Function]
Add DATA, a bytevector, to IPFS. Return a content object representing it.

`add-file` *file* [*#:name*] [Function]
Add FILE under NAME to the IPFS and return a content object for it.

`content?` [Special Form]

`content-name` [Special Form]

`content-hash` [Special Form]

`content-size` [Special Form]

`add-empty-directory` [*#:name*] [Function]
Return a content object for an empty directory.

`add-to-directory` *directory file name* [Function]
Add FILE to DIRECTORY under NAME, and return the resulting directory. DIRECTORY and FILE must be hashes identifying objects in the IPFS store.

`read-contents` *object* [*#:offset*] [*#:length*] [Function]
Return an input port to read the content of OBJECT from.

`publish-name` *object* [Function]
Publish OBJECT under the current peer ID.

55 (guix least-authority)

55.1 Overview

```
This module provides tools to execute programs with the least authority
necessary, using Linux namespaces.
```

55.2 Usage

`least-authority-wrapper` *program* [*#:name*] [*#:guest-uid*] [*#:guest-gid*] [*#:mappings*] [*#:namespaces*] [*#:directory*] [*#:preserved-environment-variables*] [Function]

Return a wrapper of PROGRAM that executes it with the least authority.

PROGRAM is executed in separate namespaces according to NAMESPACES, a list of symbols; it runs with GUEST-UID and GUEST-GID. MAPPINGS is a list of <file-system-mapping> records indicating directories mirrored inside the execution environment of PROGRAM. DIRECTORY is the working directory of the wrapped process. Each environment listed in PRESERVED-ENVIRONMENT-VARIABLES is preserved; other environment variables are erased.

56 (guix licenses)

56.1 Overview

Available licenses.

This list is based on these links:
https://github.com/NixOS/nixpkgs/blob/master/lib/licenses.nix
https://www.gnu.org/licenses/license-list

Please update spdx-string->license from guix/import/utils.scm
when modifying this list to avoid mismatches.

56.2 Usage

`license?` [Special Form]

`license-name` [Special Form]

`license-uri` [Special Form]

`license-comment` [Special Form]

`agpl1` [Variable]

`agpl3` [Variable]

`agpl3+` [Variable]

`apsl2` [Variable]

`asl1.1` [Variable]

`asl2.0` [Variable]

`boost1.0` [Variable]

`bsd-0` [Variable]

`bsd-1` [Variable]

`bsd-2` [Variable]

`bsd-3` [Variable]

`bsd-4` [Variable]

`non-copyleft` *uri* [*comment*] [Function]
Return a lax, permissive, non-copyleft license (for example a variant of the 3-clause BSD license or the Expat license), whose full text can be found at URI, which may be a file:// URI pointing the package's tree.

`cc0` [Variable]

`cc-by2.0` [Variable]

`cc-by3.0` [Variable]

`cc-by4.0` [Variable]

`cc-by-sa2.0` [Variable]

`cc-by-sa3.0` [Variable]

`cc-by-sa4.0` [Variable]

`cddl1.0` [Variable]

`cddl1.1` [Variable]

`cecill` [Variable]

`cecill-b` [Variable]

`cecill-c` [Variable]

`artistic2.0` [Variable]

`clarified-artistic` [Variable]

`copyleft-next` [Variable]

`cpl1.0` [Variable]

`cua-opl1.0` [Variable]

`edl1.0` [Variable]

`epl1.0` [Variable]

`epl2.0` [Variable]

`eupl1.1` [Variable]

`eupl1.2` [Variable]

`expat` [Variable]

`expat-0` [Variable]

`freetype` [Variable]

`freebsd-doc` [Variable]

`giftware` [Variable]

`gpl1` [Variable]

`gpl1+` [Variable]

`gpl2` [Variable]

`gpl2+` [Variable]

`gpl3` [Variable]

`gpl3+` [Variable]

`gfl1.0` [Variable]

`fdl1.1+` [Variable]

`fdl1.2+` [Variable]

`fdl1.3+` [Variable]

`opl1.0+` [Variable]

`osl2.1` [Variable]

`isc` [Variable]

`ijg` [Variable]

`ibmpl1.0` [Variable]

`imlib2` [Variable]

`ipa` [Variable]

`knuth` [Variable]

`lal1.3` [Variable]

`lgpl2.0` [Variable]

`lgpl2.0+` [Variable]

`lgpl2.1` [Variable]

`lgpl2.1+` [Variable]

`lgpl3` [Variable]

`lgpl3+` [Variable]

`llgpl` [Variable]

`lppl` [Variable]

`lppl1.0+` [Variable]

`lppl1.1+` [Variable]

`lppl1.2` [Variable]

`lppl1.2+` [Variable]

`lppl1.3` [Variable]

`lppl1.3+` [Variable]

`lppl1.3a` [Variable]

`lppl1.3a+` [Variable]

`lppl1.3b` [Variable]

`lppl1.3b+` [Variable]

`lppl1.3c` [Variable]

`lppl1.3c+` [Variable]

`miros` [Variable]

`mpl1.0` [Variable]

`mpl1.1` [Variable]

`mpl2.0` [Variable]

`ms-pl` [Variable]

`ncsa` [Variable]

`nmap` [Variable]

`ogl-psi1.0` [Variable]

`openldap2.8` [Variable]

`openssl` [Variable]

`perl-license` [Variable]

`psfl` [Variable]

`public-domain` [Variable]

`qpl` [Variable]

`qwt1.0` [Variable]

`repoze` [Variable]

`ruby` [Variable]

`sgifreeb2.0` [Variable]

`silofl1.1` [Variable]

`sleepycat` [Variable]

`tcl/tk` [Variable]

`unicode` [Variable]

`unlicense` [Variable]

`vim` [Variable]

`w3c` [Variable]

`x11` [Variable]

`x11-style` *uri* [*comment*] [Function]
Return an X11-style license, whose full text can be found at URI, which may be a file:// URI pointing the package's tree.

`zpl2.1` [Variable]

`zlib` [Variable]

`fsf-free` *uri* [*comment*] [Function]
Return a license that does not fit any of the ones above or a collection of licenses, approved as free by the FSF. More details can be found at URI.

`wtfpl2` [Variable]

`wxwindows3.1+` [Variable]

`hpnd` [Variable]

`fsdg-compatible` *uri* [*comment*] [Function]

Return a license that does not fit any of the ones above or a collection of licenses, not necessarily free, but in accordance with FSDG as Non-functional Data. More details can be found at URI. See also https://www.gnu.org/distros/free-system-distribution-guidelines.en.html#non-functional-data.

57 (guix lint)

57.1 Overview

57.2 Usage

`check-description-style` *package* [Function]

`check-inputs-should-be-native` *package* [Function]

`check-inputs-should-not-be-an-input-at-all` *package* [Function]

`check-input-labels` *package* [Function]
Emit a warning for labels that differ from the corresponding package name.

`check-wrapper-inputs` *package* [Function]
Emit a warning if PACKAGE uses 'wrap-program' or similar, but `"bash"` or `"bash-minimal"` is not in its inputs. 'wrap-script' is not supported.

`check-patch-file-names` *package* [Function]
Emit a warning if the patches requires by PACKAGE are badly named or if the patch could not be found.

`check-patch-headers` *package* [Function]
Check that PACKAGE's patches start with a comment. Return a list of warnings.

`check-synopsis-style` *package* [Function]

`check-derivation` *package* [*#:store*] [Function]
Emit a warning if we fail to compile PACKAGE to a derivation.

`check-home-page` *package* [Function]
Emit a warning if PACKAGE has an invalid 'home-page' field, or if that 'home-page' is not reachable.

`check-name` *package* [Function]
Check whether PACKAGE's name matches our guidelines.

`check-source` *package* [Function]
Emit a warning if PACKAGE has an invalid 'source' field, or if that 'source' is not reachable.

`check-source-file-name` *package* [Function]
Emit a warning if PACKAGE's origin has no meaningful file name.

`check-source-unstable-tarball` *package* [Function]
Emit a warning if PACKAGE's source is an autogenerated tarball.

`check-optional-tests` *package* [Function]
Emit a warning if the test suite is run unconditionally.

`check-mirror-url` *package* [Function]
Check whether PACKAGE uses source URLs that should be 'mirror://'.

`check-github-url` *package* [*#:timeout*] [Function]
Check whether PACKAGE uses source URLs that redirect to GitHub.

`check-license` *package* [Function]
Warn about type errors of the 'license' field of PACKAGE.

`check-vulnerabilities` *package* [*package-vulnerabilities*] [Function]
Check for known vulnerabilities for PACKAGE. Obtain the list of vulnerability records for PACKAGE by calling PACKAGE-VULNERABILITIES.

`check-for-updates` *package* [Function]
Check if there is an update available for PACKAGE.

`check-formatting` *package* [Function]
Check the formatting of the source code of PACKAGE.

`check-archival` *package* [Function]
Check whether PACKAGE's source code is archived on Software Heritage. If it's not, and if its source code is a VCS snapshot, then send a "save" request to Software Heritage.

Software Heritage imposes limits on the request rate per client IP address. This checker prints a notice and stops doing anything once that limit has been reached.

`check-profile-collisions` *package* [*#:store*] [Function]
Check for collisions that would occur when installing PACKAGE as a result of the propagated inputs it pulls in.

`check-haskell-stackage` *package* [Function]
Check whether PACKAGE is a Haskell package ahead of the current Stackage LTS version.

`check-tests-true` *package* [Function]
Check whether PACKAGE explicitly requests to run tests, which is superfluous when building natively and incorrect when cross-compiling.

`lint-warning` [Special Form]

`lint-warning?` [Special Form]

`lint-warning-package` [Special Form]

`lint-warning-message` *warning* [Function]

`lint-warning-message-text` [Special Form]

`lint-warning-message-data` [Special Form]

`lint-warning-location` [Special Form]

`%local-checkers` [Variable]

`%network-dependent-checkers` [Variable]

58 (guix man-db)

58.1 Overview

58.2 Usage

`mandb-entry?` [Special Form]

`mandb-entry-file-name` [Special Form]

`mandb-entry-name` [Special Form]

`mandb-entry-section` [Special Form]

`mandb-entry-synopsis` [Special Form]

`mandb-entry-kind` [Special Form]

`mandb-entries` *directory* [Function]
Return mandb entries for the man pages found under DIRECTORY, recursively.

`write-mandb-database` *file entries* [Function]
Write ENTRIES to FILE as a man-db database. FILE is usually ".../index.db", and is a GDBM database.

59 (guix memoization)

59.1 Overview

59.2 Usage

`invalidate-memoization!` *proc* [Function]
Invalidate the memoization cache of PROC.

`memoize` *proc* [Function]
Return a memoizing version of PROC.

This is a generic version of 'mlambda' what works regardless of the arity of 'proc'. It is more expensive since the argument list is always allocated, and the result is returned via (apply values results).

`mlambda` *formals body ...* [Special Form]
Define a memoizing lambda. The lambda's arguments are compared with 'equal?', and BODY is expected to yield a single return value.

`mlambdaq` *formals body ...* [Special Form]
Define a memoizing lambda. If FORMALS lists a single argument, it is compared using 'eq?'; otherwise, the argument list is compared using 'equal?'. BODY is expected to yield a single return value.

60 (guix modules)

60.1 Overview

```
This module provides introspection tools for Guile modules at the source
level.  Namely, it allows you to determine the closure of a module; it
does so just by reading the 'define-module' clause of the module and its
dependencies.  This is primarily useful as an argument to
'with-imported-modules'.
```

60.2 Usage

`missing-dependency-error?` *obj* [Function]

`missing-dependency-module` *obj* [Function]

`missing-dependency-search-path` *obj* [Function]

`file-name->module-name` *file* [Function]
Return the module name (a list of symbols) corresponding to FILE.

`module-name->file-name` *module* [Function]
Return the file name for MODULE.

`source-module-dependencies` *module* [*load-path*] [Function]
Return the modules used by MODULE by looking at its source code.

`source-module-closure` *modules* [*load-path*] [*#:select?*] [Function]
Return the closure of MODULES by reading 'define-module' forms in their source code. MODULES and the result are a list of Guile module names. Only modules that match SELECT? are considered.

`live-module-closure` *modules* [*#:select?*] [Function]
Return the closure of MODULES, determined by looking at live (loaded) module information. MODULES and the result are a list of Guile module names. Only modules that match SELECT? are considered.

`guix-module-name?` *name* [Function]
Return true if NAME (a list of symbols) denotes a Guix module.

61 (guix monad-repl)

61.1 Overview

61.2 Usage

`run-in-store` [Variable]
[unbound!]

`enter-store-monad` [Variable]
[unbound!]

62 (guix monads)

62.1 Overview

```
This module implements the general mechanism of monads, and provides in
particular an instance of the "state" monad.  The API was inspired by that
of Racket's "better-monads" module (see
<http://planet.racket-lang.org/package-source/toups/functional.plt/1/1/planet-docs/better-
The implementation and use case were influenced by Oleg Kysielov's
"Monadic Programming in Scheme" (see
<http://okmij.org/ftp/Scheme/monad-in-Scheme.html>).
```

62.2 Usage

`define-monad` [Special Form]
: Define the monad under NAME, with the given bind and return methods.

`monad?` [Special Form]

`monad-bind` [Special Form]

`monad-return` [Special Form]

`template-directory` [Special Form]
: This is a "stateful macro" to register and lookup templates and template instances.

`>>=` [Special Form]

`return` [Special Form]

`with-monad` [Special Form]
: Evaluate BODY in the context of MONAD, and return its result.

`mlet` [Special Form]

`mlet*` *monad* () *body ...* [Special Form]
`mlet*` *monad* ((*var mval*) *rest ...*) *body ...* [Special Form]
`mlet*` *monad* ((*var* -> *val*) *rest ...*) *body ...* [Special Form]
: Bind the given monadic values MVAL to the given variables VAR. When the form is (VAR -> VAL), bind VAR to the non-monadic value VAL in the same way as 'let'.

`mbegin` *%current-monad mexp* [Special Form]
`mbegin` *%current-monad mexp rest ...* [Special Form]
`mbegin` *monad mexp* [Special Form]
`mbegin` *monad mexp rest ...* [Special Form]
: Bind MEXP and the following monadic expressions in sequence, returning the result of the last expression. Every expression in the sequence must be a monadic expression.

`mwhen` *condition mexp0 mexp* ...* [Special Form]
: When CONDITION is true, evaluate the sequence of monadic expressions MEXP0..MEXP* as in an 'mbegin'. When CONDITION is false, return *unspecified* in the current monad. Every expression in the sequence must be a monadic expression.

`munless` *condition mexp0 mexp* ...* [Special Form]
When CONDITION is false, evaluate the sequence of monadic expressions MEXP0..MEXP* as in an 'mbegin'. When CONDITION is true, return *unspecified* in the current monad. Every expression in the sequence must be a monadic expression.

`mparameterize` *monad* ((*parameter value*) *rest ...*) *body ...* [Special Form]
`mparameterize` *monad* () *body ...* [Special Form]
This form implements dynamic scoping, similar to 'parameterize', but in a monadic context.

`lift0` [Special Form]
Lift PROC to MONAD—i.e., return a monadic function in MONAD.

`lift1` [Special Form]
Lift PROC to MONAD—i.e., return a monadic function in MONAD.

`lift2` [Special Form]
Lift PROC to MONAD—i.e., return a monadic function in MONAD.

`lift3` [Special Form]
Lift PROC to MONAD—i.e., return a monadic function in MONAD.

`lift4` [Special Form]
Lift PROC to MONAD—i.e., return a monadic function in MONAD.

`lift5` [Special Form]
Lift PROC to MONAD—i.e., return a monadic function in MONAD.

`lift6` [Special Form]
Lift PROC to MONAD—i.e., return a monadic function in MONAD.

`lift7` [Special Form]
Lift PROC to MONAD—i.e., return a monadic function in MONAD.

`lift` *proc monad* [Function]
Lift PROC, a procedure that accepts an arbitrary number of arguments, to MONAD—i.e., return a monadic function in MONAD.

`listm` [Special Form]
Return a monadic list in MONAD from the monadic values MVAL.

`foldm` [Special Form]

`mapm` [Special Form]

`sequence` [Special Form]

`anym` [Special Form]

`%identity-monad` [Special Form]

`%state-monad` [Special Form]

`state-return` [Special Form]

`state-bind` [Special Form]

`current-state` [Special Form]

`set-current-state` [Special Form]

`state-push` *value* [Function]
Push VALUE to the current state, which is assumed to be a list, and return the previous state as a monadic value.

`state-pop` [Function]
Pop a value from the current state and return it as a monadic value. The state is assumed to be a list.

`run-with-state` *mval* [*state*] [Function]
Run monadic value MVAL starting with STATE as the initial state. Return two values: the resulting value, and the resulting state.

63 (guix narinfo)

63.1 Overview

63.2 Usage

`narinfo-signature->canonical-sexp` *str* [Function]
Return the value of a narinfo's 'Signature' field as a canonical sexp.

`narinfo?` [Special Form]

`narinfo-path` [Special Form]

`narinfo-uris` [Special Form]

`narinfo-uri-base` [Special Form]

`narinfo-compressions` [Special Form]

`narinfo-file-hashes` [Special Form]

`narinfo-file-sizes` [Special Form]

`narinfo-hash` [Special Form]

`narinfo-size` [Special Form]

`narinfo-references` [Special Form]

`narinfo-deriver` [Special Form]

`narinfo-system` [Special Form]

`narinfo-signature` [Special Form]

`narinfo-contents` [Special Form]

`narinfo-hash-algorithm+value` *narinfo* [Function]
Return two values: the hash algorithm used by NARINFO and its value as a bytevector.

`narinfo-hash->sha256` *hash* [Function]
If the string HASH denotes a sha256 hash, return it as a bytevector. Otherwise return #f.

`narinfo-best-uri` *narinfo* [*#:fast-decompression?*] [Function]
Select the "best" URI to download NARINFO's nar, and return three values: the URI, its compression method (a string), and the compressed file size. When FAST-DECOMPRESSION? is true, prefer substitutes with faster decompression (typically zstd) rather than substitutes with a higher compression ratio (typically lzip).

`valid-narinfo?` *narinfo* [*acl*] [*#:verbose?*] [Function]
Return #t if NARINFO's signature is valid and made by one of the keys in ACL.

`read-narinfo` *port* [*url*] [*#:size*] [Function]
Read a narinfo from PORT. If URL is true, it must be a string used to build full URIs from relative URIs found while reading PORT. When SIZE is true, read at most SIZE bytes from PORT; otherwise, read as much as possible.

No authentication and authorization checks are performed here!

`write-narinfo` *narinfo port* [Function]
Write NARINFO to PORT.

`string->narinfo` *str cache-uri* [Function]
Return the narinfo represented by STR. Assume CACHE-URI as the base URI of the cache STR originates form.

`narinfo->string` *narinfo* [Function]
Return the external representation of NARINFO.

`equivalent-narinfo?` *narinfo1 narinfo2* [Function]
Return true if NARINFO1 and NARINFO2 are equivalent–i.e., if they describe the same store item. This ignores unnecessary metadata such as the Nar URL.

64 (guix nar)

64.1 Overview

64.2 Usage

`nar-invalid-hash-error?` *obj* [Function]

`nar-invalid-hash-error-expected` *obj* [Function]

`nar-invalid-hash-error-actual` *obj* [Function]

`nar-signature-error?` *obj* [Function]

`nar-signature-error-signature` *obj* [Function]

`restore-file-set` *port* [*#:verify-signature?*] [*#:lock?*] [*#:log-port*] [Function]

Restore the file set ("nar bundle") read from PORT to the store. The format of the data on PORT must be as created by 'export-paths'—i.e., a series of Nar-formatted archives with interspersed meta-data joining them together, possibly with a digital signature at the end. Log progress to LOG-PORT. Return the list of files restored.

When LOCK? is #f, assume locks for the files to be restored are already held. This is the case when the daemon calls a build hook.

Note that this procedure accesses the store directly, so it's only meant to be used by the daemon's build hooks since they cannot call back to the daemon while the locks are held.

65 (guix openpgp)

65.1 Overview

```
This module contains code to read OpenPGP messages as described in
<https://tools.ietf.org/html/rfc4880>, with extensions from
<https://tools.ietf.org/html/draft-ietf-openpgp-rfc4880bis-06> (notably
EdDSA support and extra signature sub-packets).

Currently this module does enough to verify detached signatures of binary
data.  It does _not_ perform sanity checks on self-signatures, subkey
binding signatures, etc., among others.  Use only in a context where this
limitations are acceptable!
```

65.2 Usage

`get-openpgp-detached-signature/ascii` *port* [Function]
: Read from PORT an ASCII-armored detached signature. Return an <openpgp-signature> record or the end-of-file object. Raise an error if the data read from PORT does is invalid or does not correspond to a detached signature.

`verify-openpgp-signature` *sig keyring dataport* [Function]
: Verify that the data read from DATAPORT matches SIG, an <openpgp-signature>. Fetch the public key of the issuer of SIG from KEYRING, a keyring as returned by 'get-openpgp-keyring'. Return two values: a status symbol, such as 'bad-signature or 'missing-key, and additional info, such as the issuer's OpenPGP public key extracted from KEYRING.

`port-ascii-armored?` *p* [Function]

`openpgp-error?` *obj* [Function]

`openpgp-unrecognized-packet-error?` *obj* [Function]

`openpgp-unrecognized-packet-error-port` *obj* [Function]

`openpgp-unrecognized-packet-error-type` *obj* [Function]

`openpgp-invalid-signature-error?` *obj* [Function]

`openpgp-invalid-signature-error-port` *obj* [Function]

`openpgp-signature?` [Special Form]

`openpgp-signature-issuer-key-id` [Special Form]

`openpgp-signature-issuer-fingerprint` [Special Form]

`openpgp-signature-public-key-algorithm` [Special Form]

`openpgp-signature-hash-algorithm` [Special Form]

`openpgp-signature-creation-time` *sig* [Function]

`openpgp-signature-expiration-time` *sig* [Function]

`openpgp-user-id?` [Special Form]

`openpgp-user-id-value` [Special Form]

`openpgp-user-attribute?` [Special Form]

`openpgp-public-key?` [Special Form]

`openpgp-public-key-subkey?` [Special Form]

`openpgp-public-key-value` [Special Form]

`openpgp-public-key-fingerprint` [Special Form]

`openpgp-format-fingerprint` *bv* [Function]
Return a string representing BV, a bytevector, in the conventional OpenPGP hexadecimal format for fingerprints.

`openpgp-public-key-id` *k* [Function]

`openpgp-keyring?` [Special Form]

`%empty-keyring` [Variable]

`lookup-key-by-id` *keyring id* [Function]
Return two values: the first key with ID in KEYRING, and a list of associated packets (user IDs, signatures, etc.). Return #f and the empty list of ID was not found. ID must be the 64-bit key ID of the key, an integer.

`lookup-key-by-fingerprint` *keyring fingerprint* [Function]
Return two values: the key with FINGERPRINT in KEYRING, and a list of associated packets (user IDs, signatures, etc.). Return #f and the empty list of FINGERPRINT was not found. FINGERPRINT must be a bytevector.

`get-openpgp-keyring` *port* [*keyring*] [*#:limit*] [Function]
Read from PORT an OpenPGP keyring in binary format; return a keyring based on all the OpenPGP primary keys that were read. The returned keyring complements KEYRING. LIMIT is the maximum number of keys to read, or -1 if there is no limit.

`read-radix-64` *port* [Function]
Read from PORT an ASCII-armored Radix-64 stream, decode it, and return the result as a bytevector as well as the type, a string such as "PGP MESSAGE". Return #f if PORT does not contain a valid Radix-64 stream, and the end-of-file object if the Radix-64 sequence was truncated.

`string->openpgp-packet` *str* [Function]
Read STR, an ASCII-armored OpenPGP packet, and return the corresponding OpenPGP record.

`get-openpgp-packet` *p* [Function]

66 (guix packages)

66.1 Overview

```
This module provides a high-level mechanism to define packages in a
Guix-based distribution.
```

66.2 Usage

`content-hash` [Special Form]
Return a content hash with the given parameters. The default hash algorithm is sha256. If the first argument is a literal string, it is decoded as base32. Otherwise, it must be a bytevector.

`content-hash?` [Special Form]

`content-hash-algorithm` [Special Form]

`content-hash-value` [Special Form]

`origin` *fields ...* [Special Form]
Build an <origin> record, automatically converting 'sha256' field specifications to 'hash'.

`origin?` [Special Form]

`this-origin` [Special Form]
Return the record being defined. This macro may only be used in the context of the definition of a thunked field.

`origin-uri` [Special Form]

`origin-method` [Special Form]

`origin-hash` [Special Form]

`origin-sha256` [Special Form]

`origin-file-name` [Special Form]

`origin-actual-file-name` *origin* [Function]
Return the file name of ORIGIN, either its 'file-name' field or the file name of its URI.

`origin-patches` [Special Form]

`origin-patch-flags` [Special Form]

`origin-patch-inputs` [Special Form]

`origin-patch-guile` [Special Form]

`origin-snippet` [Special Form]

`origin-modules` [Special Form]

`base32` [Special Form]
Return the bytevector corresponding to the given textual representation.

`base64` [Special Form]
Return the bytevector corresponding to the given textual representation.

`package` [Special Form]

`package?` [Special Form]

`this-package` [Special Form]
Return the record being defined. This macro may only be used in the context of the definition of a thunked field.

`package-name` [Special Form]

`package-upstream-name` *package* [Function]
Return the upstream name of PACKAGE, which could be different from the name it has in Guix.

`package-version` [Special Form]

`package-full-name` *package* [*delimiter*] [Function]
```
Return the full name of PACKAGE--i.e., `NAME@@VERSION'.  By specifying
DELIMITER (a string), you can customize what will appear between the name and
the version.  By default, DELIMITER is "@@".
```

`package-source` [Special Form]

`package-build-system` [Special Form]

`package-arguments` [Special Form]

`package-inputs` [Special Form]

`package-native-inputs` [Special Form]

`package-propagated-inputs` [Special Form]

`package-outputs` [Special Form]

`package-native-search-paths` [Special Form]

`package-search-paths` [Special Form]

`package-replacement` [Special Form]

`package-synopsis` [Special Form]

`package-description` [Special Form]

`package-license` [Special Form]

`package-home-page` [Special Form]

`package-supported-systems` [Special Form]

`package-properties` [Special Form]

`package-location` *package* [Function]
Return the source code location of PACKAGE as a <location> record, or #f if it is not known.

`package-definition-location` *package* [Function]
Like 'package-location', but return the location of the definition itself–i.e., that of the enclosing 'define-public' form, if any, or #f.

`hidden-package` *p* [Function]
Return a "hidden" version of P–i.e., one that 'fold-packages' and thus, user interfaces, ignores.

`hidden-package?` *p* [Function]
Return true if P is "hidden"–i.e., must not be visible to user interfaces.

`package-superseded` *p* [Function]
Return the package that supersedes P, or #f if P is still current.

`deprecated-package` *old-name p* [Function]
Return a package called OLD-NAME and marked as superseded by P, a package object.

`package-field-location` *package field* [Function]
Return the source code location of the definition of FIELD for PACKAGE, or #f if it could not be determined.

`this-package-input` *name* [Special Form]
Return the input NAME of the package being defined–i.e., an input from the 'inputs' or 'propagated-inputs' field. Native inputs are not considered. If this input does not exist, return #f instead.

`this-package-native-input` *name* [Special Form]
Return the native package input NAME of the package being defined–i.e., an input from the 'native-inputs' field. If this native input does not exist, return #f instead.

`lookup-package-input` *package name* [Function]
Look up NAME among PACKAGE's inputs. Return it if found, #f otherwise.

`lookup-package-native-input` *package name* [Function]
Look up NAME among PACKAGE's native inputs. Return it if found, #f otherwise.

`lookup-package-propagated-input` *package name* [Function]
Look up NAME among PACKAGE's propagated inputs. Return it if found, #f otherwise.

`lookup-package-direct-input` *package name* [Function]
Look up NAME among PACKAGE's direct inputs. Return it if found, #f otherwise.

`prepend` [Special Form]

`replace` [Special Form]

`modify-inputs` *inputs* (*delete name*) *clauses ...* [Special Form]
`modify-inputs` *inputs* (*delete names ...*) *clauses ...* [Special Form]
`modify-inputs` *inputs* (*prepend lst ...*) *clauses ...* [Special Form]
`modify-inputs` *inputs* (*append lst ...*) *clauses ...* [Special Form]
`modify-inputs` *inputs* (*replace name replacement*) *clauses ...* [Special Form]
`modify-inputs` *inputs* [Special Form]

```
Modify the given package inputs, as returned by 'package-inputs' & co.,
according to the given clauses.  The example below removes the GMP and ACL
inputs of Coreutils and adds libcap:

  (modify-inputs (package-inputs coreutils)
    (delete "gmp" "acl")
    (prepend libcap))

Other types of clauses include 'append' and 'replace'.

The first argument must be a labeled input list; the result is also a labeled
input list.
```

`package-direct-sources` *package* [Function]
Return all source origins associated with PACKAGE; including origins in PACKAGE's inputs.

`package-transitive-sources` *package* [Function]
Return PACKAGE's direct sources, and their direct sources, recursively.

`package-direct-inputs` *package* [Function]
Return all the direct inputs of PACKAGE—i.e, its direct inputs along with their propagated inputs.

`package-transitive-inputs` *package* [Function]
Return the transitive inputs of PACKAGE—i.e., its direct inputs along with their propagated inputs, recursively.

`package-transitive-target-inputs` *package* [Function]
Return the transitive target inputs of PACKAGE—i.e., its direct inputs along with their propagated inputs, recursively. This only includes inputs for the target system, and not native inputs.

`package-transitive-native-inputs` *package* [Function]
Return the transitive native inputs of PACKAGE—i.e., its direct inputs along with their propagated inputs, recursively. This only includes inputs for the host system ("native inputs"), and not target inputs.

`package-transitive-propagated-inputs` *package* [Function]
Return the propagated inputs of PACKAGE, and their propagated inputs, recursively.

`package-transitive-native-search-paths` *package* [Function]
Return the list of search paths for PACKAGE and its propagated inputs, recursively.

`package-transitive-supported-systems` *package* [*system*] [Function]
Return the intersection of the systems supported by PACKAGE and those supported by its dependencies.

`package-mapping` *proc* [*cut?*] [*#:deep?*] [Function]
Return a procedure that, given a package, applies PROC to all the packages depended on and returns the resulting package. The procedure stops recursion when CUT? returns true for a given package. When DEEP? is true, PROC is applied to implicit inputs as well.

`package-input-rewriting` *replacements* [*rewrite-name*] [*#:deep?*] [Function]
Return a procedure that, when passed a package, replaces its direct and indirect dependencies, including implicit inputs when DEEP? is true, according to REPLACEMENTS. REPLACEMENTS is a list of package pairs; the first element of each pair is the package to replace, and the second one is the replacement.

Optionally, REWRITE-NAME is a one-argument procedure that takes the name of a package and returns its new name after rewrite.

`package-input-rewriting/spec` *replacements* [*#:deep?*] [Function]
`Return a procedure that, given a package, applies the given REPLACEMENTS to all the package graph, including implicit inputs unless DEEP? is false. REPLACEMENTS is a list of spec/procedures pair; each spec is a package specification such as "gcc" or "guile@@2", and each procedure takes a matching package and returns a replacement for that package.`

`package-source-derivation` *store source* [*system*] [Function]
Return the derivation or file corresponding to SOURCE, which can be an a file name or any object handled by 'lower-object', such as an <origin>. When SOURCE is a file name, return either the interned file name (if SOURCE is outside of the store) or SOURCE itself (if SOURCE is already a store item.)

`package-derivation` *store . args* [Function]
Return the <derivation> object of PACKAGE for SYSTEM.

`package-cross-derivation` *store . args* [Function]
Cross-build PACKAGE for TARGET (a GNU triplet) from host SYSTEM (a Guix system identifying string).

`package-output` *store package* [*output*] [*system*] [Function]
Return the output path of PACKAGE's OUTPUT for SYSTEM—where OUTPUT is the symbolic output name, such as "out". Note that this procedure calls 'package-derivation', which is costly.

`package-grafts` *store . args* [Function]
Return the list of grafts applicable to PACKAGE as built for SYSTEM and TARGET.

`package-patched-vulnerabilities` *package* [Function]
Return the list of patched vulnerabilities of PACKAGE as a list of CVE identifiers. The result is inferred from the file names of patches.

`package-with-patches` *original patches* [Function]
Return package ORIGINAL with PATCHES applied.

`package-with-extra-patches` *original patches* [Function]
Return package ORIGINAL with all PATCHES appended to its list of patches.

`package-with-c-toolchain` *package toolchain* [Function]
Return a variant of PACKAGE that uses TOOLCHAIN instead of the default GNU C/C++ toolchain. TOOLCHAIN must be a list of inputs (label/package tuples) providing equivalent functionality, such as the 'gcc-toolchain' package.

`package/inherit` *p overrides ...* [Special Form]
Like (package (inherit P) OVERRIDES ...), except that the same transformation is done to the package P's replacement, if any. P must be a bare identifier, and will be bound to either P or its replacement when evaluating OVERRIDES.

`transitive-input-references` *alist inputs* [Function]
Return a list of (assoc-ref ALIST <label>) for each (<label> <package> . _) in INPUTS and their transitive propagated inputs.

`%32bit-supported-systems` [Variable]

`%64bit-supported-systems` [Variable]

`%supported-systems` [Variable]

`%hurd-systems` [Variable]

`%cuirass-supported-systems` [Variable]

`supported-package?` *package* [*system*] [Function]
Return true if PACKAGE is supported on SYSTEM–i.e., if PACKAGE and all its dependencies are known to build on SYSTEM.

`&package-error` [Variable]

`package-error?` *obj* [Function]

`package-error-package` *obj* [Function]

`package-license-error?` *obj* [Function]

`package-error-invalid-license` *obj* [Function]

`&package-input-error` [Variable]

`package-input-error?` *obj* [Function]

`package-error-invalid-input` *obj* [Function]

`&package-cross-build-system-error` [Variable]

`package-cross-build-system-error?` *obj* [Function]

`package->bag` *package* [*system*] [*target*] [*#:graft?*] [Function]
Compile PACKAGE into a bag for SYSTEM, possibly cross-compiled to TARGET, and return it.

`bag->derivation` *bag* [*context*] [Function]
Return the derivation to build BAG for SYSTEM. Optionally, CONTEXT can be a package object describing the context in which the call occurs, for improved error reporting.

`bag-direct-inputs` *bag* [Function]
Same as 'package-direct-inputs', but applied to a bag.

`bag-transitive-inputs` *bag* [Function]
Same as 'package-transitive-inputs', but applied to a bag.

`bag-transitive-host-inputs` *bag* [Function]
Same as 'package-transitive-target-inputs', but applied to a bag.

`bag-transitive-build-inputs` *bag* [Function]
Same as 'package-transitive-native-inputs', but applied to a bag.

`bag-transitive-target-inputs` *bag* [Function]
Return the "target inputs" of BAG, recursively.

`package-development-inputs` *package* [*system*] [*#:target*] [Function]
Return the list of inputs required by PACKAGE for development purposes on SYSTEM. When TARGET is true, return the inputs needed to cross-compile PACKAGE from SYSTEM to TRIPLET, where TRIPLET is a triplet such as "aarch64-linux-gnu".

`package-closure` *packages* [*#:system*] [Function]
Return the closure of PACKAGES on SYSTEM–i.e., PACKAGES and the list of packages they depend on, recursively.

`default-guile` [Function]
Return the default Guile package used to run the build code of derivations.

`default-guile-derivation` [*system*] [Function]
Return the derivation for SYSTEM of the default Guile package used to run the build code of derivation.

`set-guile-for-build` *guile* [Function]
This monadic procedure changes the Guile currently used to run the build code of derivations to GUILE, a package object.

`package-file` *package* [*file*] [*#:system*] [*#:output*] [*#:target*] [Function]
Return as a monadic value the absolute file name of FILE within the OUTPUT directory of PACKAGE. When FILE is omitted, return the name of the OUTPUT directory of PACKAGE. When TARGET is true, use it as a cross-compilation target triplet.

Note that this procedure does _not_ build PACKAGE. Thus, the result might or might not designate an existing file. We recommend not using this procedure unless you know what you are doing.

`package->derivation` *package* [*system*] [*#:graft?*] [Function]
Return the <derivation> object of PACKAGE for SYSTEM.

`package->cross-derivation` *package target* [*system*] [*#:graft?*] [Function]
Cross-build PACKAGE for TARGET (a GNU triplet) from host SYSTEM (a Guix system identifying string).

`origin->derivation` *origin* [*system*] [Function]
Return the derivation corresponding to ORIGIN.

`%current-system` [Variable]

`%current-target-system` [Variable]

`define-public` [Special Form]
Like 'define-public' but set 'current-definition-location' for the lexical scope of its body.

`search-path-specification` [Special Form]

`delete _ _ [_]` [Function]

```
- Scheme Procedure: delete item lst
     Return a newly-created copy of LST with elements `equal?' to ITEM
     removed.  This procedure mirrors `member': `delete' compares
     elements of LST against ITEM with `equal?'.
```

67 (guix pki)

67.1 Overview

```
Public key infrastructure for the authentication and authorization of
archive imports.  This is essentially a subset of SPKI for our own
purposes (see <http://theworld.com/~cme/spki.txt> and
<http://www.ietf.org/rfc/rfc2693.txt>.)
```

67.2 Usage

`%public-key-file` [Variable]

`%private-key-file` [Variable]

`%acl-file` [Variable]

`current-acl` [Function]
Return the current ACL.

`public-keys->acl` *keys* [Function]
Return an ACL that lists all of KEYS with a '(guix import)' tag—meaning that all of KEYS are authorized for archive imports. Each element in KEYS must be a canonical sexp with type 'public-key'.

`acl->public-keys` *acl* [Function]
Return the public keys (as canonical sexps) listed in ACL with the '(guix import)' tag.

`authorized-key?` *key* [*acl*] [Function]
Return #t if KEY (a canonical sexp) is an authorized public key for archive imports according to ACL.

`write-acl` *acl port* [Function]
Write ACL to PORT in canonical-sexp format.

`signature-sexp` *data secret-key public-key* [Function]
Return a SPKI-style sexp for the signature of DATA with SECRET-KEY that includes DATA, the actual signature value (with a 'sig-val' tag), and PUBLIC-KEY (see <http://theworld.com/~cme/spki.txt> for examples.)

`signature-subject` *sig* [Function]
Return the signer's public key for SIG.

`signature-signed-data` *sig* [Function]
```
Return the signed data from SIG, typically an sexp such as
  (hash "sha256" #...#).
```

`valid-signature?` *sig* [Function]
Return #t if SIG is valid.

`signature-case` (*signature hash acl*) (*valid-signature valid-exp ...*) (*else else-exp ...*) [Special Form]

`signature-case` (*signature hash acl*) (*valid-signature valid-exp ...*) (*invalid-signature invalid-exp ...*) (*hash-mismatch mismatch-exp ...*) (*unauthorized-key unauthorized-exp ...*) (*corrupt-signature corrupt-exp ...*) [Special Form]

```
Match the cases of the verification of SIGNATURE against HASH and ACL:

  - the 'valid-signature' case if SIGNATURE is indeed a signature of HASH with
    a key present in ACL;
  - 'invalid-signature' if SIGNATURE is incorrect;
  - 'hash-mismatch' if the hash in SIGNATURE does not match HASH;
  - 'unauthorized-key' if the public key in SIGNATURE is not listed in ACL;
  - 'corrupt-signature' if SIGNATURE is not a valid signature sexp.

This macro guarantees at compile-time that all these cases are handled.

SIGNATURE, and ACL must be canonical sexps; HASH must be a bytevector.
```

68 (guix platform)

68.1 Overview

68.2 Usage

`platform` [Special Form]

`platform?` [Special Form]

`platform-target` [Special Form]

`platform-system` [Special Form]

`platform-linux-architecture` [Special Form]

`platform-glibc-dynamic-linker` [Special Form]

`platform-modules` [Function]
Return the list of platform modules.

`platforms` . *args* [Function]

`lookup-platform-by-system` *system* [Function]
Return the platform corresponding to the given SYSTEM.

`lookup-platform-by-target` *target* [Function]
Return the platform corresponding to the given TARGET.

`lookup-platform-by-target-or-system` *target-or-system* [Function]
Return the platform corresponding to the given TARGET or SYSTEM.

`platform-system->target` *system* [Function]
Return the target matching the given SYSTEM if it exists or false otherwise.

`platform-target->system` *target* [Function]
Return the system matching the given TARGET if it exists or false otherwise.

`systems` [Function]
Return the list of supported systems.

`targets` [Function]
Return the list of supported targets.

69 (guix profiles)

69.1 Overview

Tools to create and manipulate profiles---i.e., the representation of a set of installed packages.

69.2 Usage

`&profile-error` [Variable]

`profile-error?` *obj* [Function]

`profile-error-profile` *obj* [Function]

`&profile-not-found-error` [Variable]

`profile-not-found-error?` *obj* [Function]

`&profile-collision-error` [Variable]

`profile-collision-error?` *obj* [Function]

`profile-collision-error-entry` *obj* [Function]

`profile-collision-error-conflict` *obj* [Function]

`&missing-generation-error` [Variable]

`missing-generation-error?` *obj* [Function]

`missing-generation-error-generation` *obj* [Function]

`&unmatched-pattern-error` [Variable]

`unmatched-pattern-error?` *obj* [Function]

`unmatched-pattern-error-pattern` *obj* [Function]

`unmatched-pattern-error-manifest` *obj* [Function]

`manifest` [Special Form]

`make-manifest` *entries* [Function]

`manifest?` [Special Form]

`manifest-entries` [Special Form]

`manifest-transitive-entries` *manifest* [Function]
Return the entries of MANIFEST along with their propagated inputs, recursively.

`<manifest-entry>` [Special Form]
This macro lets us query record type info at macro-expansion time.

`manifest-entry` [Special Form]

`manifest-entry?` [Special Form]

`manifest-entry-name` [Special Form]

`manifest-entry-version` [Special Form]

`manifest-entry-output` [Special Form]

`manifest-entry-item` [Special Form]

`manifest-entry-dependencies` [Special Form]

`manifest-entry-search-paths` [Special Form]

`manifest-entry-parent` [Special Form]

`manifest-entry-properties` [Special Form]

`lower-manifest-entry` *entry system* [*#:target*] [Function]
Lower ENTRY for SYSTEM and TARGET such that its 'item' field is a store file name.

`manifest-entry=?` *entry1 entry2* [Function]
Return true if ENTRY1 is equivalent to ENTRY2, ignoring their 'properties' field.

`manifest-pattern` [Special Form]

`manifest-pattern?` [Special Form]

`manifest-pattern-name` [Special Form]

`manifest-pattern-version` [Special Form]

`manifest-pattern-output` [Special Form]

`concatenate-manifests` *lst* [Function]
Concatenate the manifests listed in LST and return the resulting manifest.

`map-manifest-entries` *proc manifest* [Function]
Apply PROC to all the entries of MANIFEST and return a new manifest.

`manifest-remove` *manifest patterns* [Function]
Remove entries for each of PATTERNS from MANIFEST. Each item in PATTERNS must be a manifest-pattern.

`manifest-add` *manifest entries* [Function]
Add a list of manifest ENTRIES to MANIFEST and return new manifest. Remove MANIFEST entries that have the same name and output as ENTRIES.

`manifest-lookup` *manifest pattern* [Function]
Return the first item of MANIFEST that matches PATTERN, or #f if there is no match..

`manifest-installed?` *manifest pattern* [Function]
Return #t if MANIFEST has an entry matching PATTERN (a manifest-pattern), #f otherwise.

`manifest-matching-entries` *manifest patterns* [Function]
Return all the entries of MANIFEST that match one of the PATTERNS. Raise an '&unmatched-pattern-error' if none of the entries of MANIFEST matches one of PATTERNS.

`manifest-search-paths` *manifest* [Function]
Return the list of search path specifications that apply to MANIFEST, including the search path specification for $PATH.

`check-for-collisions` *manifest system* [*#:target*] [Function]
Check whether the entries of MANIFEST conflict with one another; raise a '&profile-collision-error' when a conflict is encountered.

`manifest->code` *manifest* [*#:entry-package-version*] [Function]
Return an sexp representing code to build an approximate version of MANIFEST; the code is wrapped in a top-level 'begin' form. Call ENTRY-PACKAGE-VERSION to determine the version number to use in the spec for a given entry; it can be set to 'manifest-entry-version' for fully-specified version numbers, or to some other procedure to disambiguate versions for packages for which several versions are available.

`manifest-transaction` [Special Form]

`manifest-transaction?` [Special Form]

`manifest-transaction-install` [Special Form]

`manifest-transaction-remove` [Special Form]

`manifest-transaction-install-entry` *entry transaction* [Function]
Augment TRANSACTION's set of installed packages with ENTRY, a <manifest-entry>.

`manifest-transaction-remove-pattern` *pattern transaction* [Function]
Add PATTERN to TRANSACTION's list of packages to remove.

`manifest-transaction-null?` *transaction* [Function]
Return true if TRANSACTION has no effect—i.e., it neither installs nor remove software.

`manifest-transaction-removal-candidate?` *entry transaction* [Function]
Return true if ENTRY is a candidate for removal in TRANSACTION.

`manifest-perform-transaction` *manifest transaction* [Function]
Perform TRANSACTION on MANIFEST and return the new manifest.

`manifest-transaction-effects` *manifest transaction* [Function]
Compute the effect of applying TRANSACTION to MANIFEST. Return 4 values: the list of packages that would be removed, installed, upgraded, or downgraded when applying TRANSACTION to MANIFEST. Upgrades are represented as pairs where the head is the entry being upgraded and the tail is the entry that will replace it.

`profile-manifest` *profile* [Function]
Return the PROFILE's manifest.

`package->manifest-entry` *package* [*output*] [*#:parent*] [*#:properties*] [Function]
Return a manifest entry for the OUTPUT of package PACKAGE.

`package->development-manifest` *package* [*system*] [*#:target*] [Function]
Return a manifest for the "development inputs" of PACKAGE for SYSTEM, optionally when cross-compiling to TARGET. Development inputs include both explicit and implicit inputs of PACKAGE.

`packages->manifest` *packages* [Function]
Return a list of manifest entries, one for each item listed in PACKAGES. Elements of PACKAGES can be either package objects or package/string tuples denoting a specific output of a package.

`ca-certificate-bundle` *manifest* [Function]
Return a derivation that builds a single-file bundle containing the CA certificates in the /etc/ssl/certs sub-directories of the packages in MANIFEST. Single-file bundles are required by programs such as Git and Lynx.

`%default-profile-hooks` [Variable]

`%manifest-format-version` [Variable]

`profile-derivation` *manifest* [*#:name*] [*#:hooks*] [*#:locales?*] [Function]
[*#:allow-unsupported-packages?*] [*#:allow-collisions?*]
[*#:relative-symlinks?*] [*#:format-version*] [*#:system*] [*#:target*]
Return a derivation that builds a profile (aka. 'user environment') with the given MANIFEST. The profile includes additional derivations returned by the monadic procedures listed in HOOKS–such as an Info 'dir' file, etc. Unless ALLOW-COLLISIONS? is true, a '&profile-collision-error' is raised if entries in MANIFEST collide (for instance if there are two same-name packages with a different version number.) Unless ALLOW-UNSUPPORTED-PACKAGES? is true or TARGET is set, raise an error if MANIFEST contains a package that does not support SYSTEM.

When LOCALES? is true, the build is performed under a UTF-8 locale; this adds a dependency on the 'glibc-utf8-locales' package.

When RELATIVE-SYMLINKS? is true, use relative file names for symlink targets. This is one of the things to do for the result to be relocatable.

When TARGET is true, it must be a GNU triplet, and the packages in MANIFEST are cross-built for TARGET.

`profile-search-paths` *profile* [*manifest*] [*#:getenv*] [Function]
Read the manifest of PROFILE and evaluate the values of search path environment variables required by PROFILE; return a list of specification/value pairs. If MANIFEST is not #f, it is assumed to be the manifest of PROFILE, which avoids rereading it.

Use GETENV to determine the current settings and report only settings not already effective.

`load-profile` *profile* [*manifest*] [*#:pure?*] [*#:white-list-regexps*] [Function]
[*#:white-list*]
Set the environment variables specified by MANIFEST for PROFILE. When PURE? is #t, unset the variables in the current environment except those that match the

regexps in WHITE-LIST-REGEXPS and those listed in WHITE-LIST. Otherwise, augment existing environment variables with additional search paths.

`profile` [Special Form]

`profile?` [Special Form]

`profile-name` [Special Form]

`profile-content` [Special Form]

`profile-hooks` [Special Form]

`profile-locales?` [Special Form]

`profile-allow-collisions?` [Special Form]

`profile-relative-symlinks?` [Special Form]

`generation-number` *profile* [*base-profile*] [Function]
Return PROFILE's number or 0. An absolute file name must be used.

Optionally, if BASE-PROFILE is provided, use it instead of PROFILE to construct the regexp matching generations. This is useful in special cases like: (generation-number "/run/current-system" %system-profile).

`generation-profile` *file* [Function]
If FILE is a profile generation GC root such as "guix-profile-42-link", return its corresponding profile—e.g., "guix-profile". Otherwise return #f.

`generation-numbers` *profile* [Function]
Return the sorted list of generation numbers of PROFILE, or '(0) if no former profiles were found.

`profile-generations` *profile* [Function]
Return a list of PROFILE's generations.

`relative-generation-spec->number` *profile spec* [Function]
Return PROFILE's generation specified by SPEC, which is a string. The SPEC may be a N, -N, or +N, where N is a number. If the spec is N, then the number returned is N. If it is -N, then the number returned is the profile's current generation number minus N. If it is +N, then the number returned is the profile's current generation number plus N. Return #f if there is no such generation.

`relative-generation` *profile shift* [*current*] [Function]
Return PROFILE's generation shifted from the CURRENT generation by SHIFT. SHIFT is a positive or negative number. Return #f if there is no such generation.

`previous-generation-number` *profile* [*number*] [Function]
Return the number of the generation before generation NUMBER of PROFILE, or 0 if none exists. It could be NUMBER - 1, but it's not the case when generations have been deleted (there are "holes").

`generation-time` *profile number* [Function]
Return the creation time of a generation in the UTC format.

`generation-file-name` *profile generation* [Function]
Return the file name for PROFILE's GENERATION.

`switch-to-generation` *profile number* [Function]
Atomically switch PROFILE to the generation NUMBER. Return the number of the generation that was current before switching.

`roll-back` *store profile* [Function]
Roll back to the previous generation of PROFILE. Return the number of the generation that was current before switching and the new generation number.

`delete-generation` *store profile number* [Function]
Delete generation with NUMBER from PROFILE. Return the file name of the generation that has been deleted, or #f if nothing was done (for instance because the NUMBER is zero.)

`%user-profile-directory` [Variable]

`%profile-directory` [Variable]

`%current-profile` [Variable]

`ensure-profile-directory` [Function]
Attempt to create /.../profiles/per-user/$USER if needed. Nowadays this is taken care of by the daemon.

`canonicalize-profile` *profile* [Function]
If PROFILE points to a profile in %PROFILE-DIRECTORY, return that. Otherwise return PROFILE unchanged. The goal is to treat '-p ~/.guix-profile' as if '-p' was omitted.

`user-friendly-profile` *profile* [Function]
Return either ~/.guix-profile or ~/.config/guix/current if that's what PROFILE refers to, directly or indirectly, or PROFILE.

`linux-module-database` *manifest* [Function]
Return a derivation that unites all the kernel modules of the manifest and creates the dependency graph of all these kernel modules.

This is meant to be used as a profile hook.

70 (guix profiling)

70.1 Overview

Basic support for Guix-specific profiling.

70.2 Usage

`profiled?` *component* [Function]
: Return true if COMPONENT profiling is active.

`register-profiling-hook!` *component thunk* [Function]
: Register THUNK as a profiling hook for COMPONENT, a string such as "rpc".

71 (guix progress)

71.1 Overview

`Helper to write progress report code for downloads, etc.`

71.2 Usage

`<progress-reporter>` [Special Form]
This macro lets us query record type info at macro-expansion time.

`progress-reporter` [Special Form]

`make-progress-reporter` [Special Form]

`progress-reporter?` [Special Form]

`call-with-progress-reporter` *reporter proc* [Function]
Start REPORTER for progress reporting, and call (*proc* `report`) with the resulting report procedure. When *proc* returns, the REPORTER is stopped.

`start-progress-reporter!` *reporter* [Function]
Low-level procedure to start REPORTER.

`stop-progress-reporter!` *reporter* [Function]
Low-level procedure to stop REPORTER.

`progress-reporter-report!` *reporter . args* [Function]
Low-level procedure to lead REPORTER to emit a report.

`progress-reporter/silent` [Variable]

`progress-reporter/file` *file size* [*log-port*] [*#:abbreviation*] [Function]
Return a <progress-reporter> object to show the progress of FILE's download, which is SIZE bytes long. The progress report is written to LOG-PORT, with ABBREVIATION used to shorten FILE for display.

`progress-reporter/bar` *total* [*prefix*] [*port*] [Function]
Return a reporter that shows a progress bar every time one of the TOTAL tasks is performed. Write PREFIX at the beginning of the line.

`progress-reporter/trace` *file url size* [*log-port*] [Function]
Like 'progress-reporter/file', but instead of returning human-readable progress reports, write "build trace" lines to be processed elsewhere.

`progress-report-port` *reporter port* [*#:close?*] [*#:download-size*] [Function]
Return a port that continuously reports the bytes read from PORT using REPORTER, which should be a <progress-reporter> object. When CLOSE? is true, PORT is closed when the returned port is closed.

When DOWNLOAD-SIZE is passed, do not read more than DOWNLOAD-SIZE bytes from PORT. This is important to avoid blocking when the remote side won't close the underlying connection.

`display-download-progress` *file size* [*#:tty?*] [*#:start-time*] [*#:transferred*] [*#:log-port*] [Function]

Write the progress report to LOG-PORT. Use START-TIME (a SRFI-19 time object) and TRANSFERRED (a total number of bytes) to determine the throughput. When TTY? is false, assume LOG-PORT is not a tty and do not emit ANSI escape codes.

`erase-current-line` *port* [Function]

Write an ANSI erase-current-line sequence to PORT to erase the whole line and move the cursor to the beginning of the line.

`progress-bar` *%* [*bar-width*] [Function]

Return % as a string representing an ASCII-art progress bar. The total width of the bar is BAR-WIDTH.

`byte-count->string` *size* [Function]

Given SIZE in bytes, return a string representing it in a human-readable way.

`current-terminal-columns` [Variable]

`dump-port*` *in out* [*#:buffer-size*] [*#:reporter*] [Function]

Read as much data as possible from IN and write it to OUT, using chunks of BUFFER-SIZE bytes. After each successful transfer of BUFFER-SIZE bytes or less, report the total number of bytes transferred to the REPORTER, which should be a `<progress-reporter>` object.

72 (guix quirks)

72.1 Overview

Time traveling is a challenge! Sometimes, going back to the past requires adjusting the old source code so it can be evaluated with our modern day Guile and against our modern Guix APIs. This file describes quirks found in old Guix revisions, along with ways to address them or patch them.

72.2 Usage

`%quirks` [Variable]

`patch?` [Special Form]

`applicable-patch?` *patch source commit* [Function]
Return true if PATCH is applicable to SOURCE, a directory, which corresponds to the given Guix COMMIT, a SHA1 hexadecimal string.

`apply-patch` *patch source* [Function]
Apply PATCH onto SOURCE, directly modifying files beneath it.

`%patches` [Variable]

73 (guix read-print)

73.1 Overview

This module provides a comment-preserving reader and a comment-preserving pretty-printer smarter than (ice-9 pretty-print).

73.2 Usage

`pretty-print-with-comments` *port obj* [*#:format-comment*] [*#:format-vertical-space*] [*#:indent*] [*#:max-width*] [*#:long-list*] [Function]

Pretty-print OBJ to PORT, attempting to at most MAX-WIDTH character columns and assuming the current column is INDENT. Comments present in OBJ are included in the output.

Lists longer than LONG-LIST are written as one element per line. Comments are passed through FORMAT-COMMENT before being emitted; a useful value for FORMAT-COMMENT is 'canonicalize-comment'. Vertical space is passed through FORMAT-VERTICAL-SPACE; a useful value of 'canonicalize-vertical-space'.

`pretty-print-with-comments/splice` *port lst . rest* [Function]

Write to PORT the expressions and blanks listed in LST.

`read-with-comments` *port* [*#:blank-line?*] [Function]

Like 'read', but include <blank> objects when they're encountered. When BLANK-LINE? is true, assume PORT is at the beginning of a new line.

`read-with-comments/sequence` *port* [Function]

Read from PORT until the end-of-file is reached and return the list of expressions and blanks that were read.

`object->string*` *obj indent . args* [Function]

Pretty-print OBJ with INDENT columns as the initial indent. ARGS are passed as-is to 'pretty-print-with-comments'.

`blank?` *obj* [Function]

`vertical-space` *a* [Function]

`vertical-space?` *obj* [Function]

`vertical-space-height` *obj* [Function]

`canonicalize-vertical-space` *space* [Function]

Return a vertical space corresponding to a single blank line.

`page-break` [Function]

`page-break?` *obj* [Function]

`comment` *str* [*margin?*] [Function]

Return a new comment made from STR. When MARGIN? is true, return a margin comment; otherwise return a line comment. STR must start with a semicolon and end with newline, otherwise an error is raised.

`comment?` *obj* [Function]

`comment->string` *obj* [Function]

`comment-margin?` *obj* [Function]

`canonicalize-comment` *comment indent* [Function]
Canonicalize COMMENT, which is to be printed at INDENT, ensuring it has the "right" number of leading semicolons.

74 (guix records)

74.1 Overview

Utilities for dealing with Scheme records.

74.2 Usage

define-record-type* [Special Form]

Define the given record type such that an additional "syntactic constructor" is defined, which allows instances to be constructed with named field initializers, à la SRFI-35, as well as default values. An example use may look like this:

```
(define-record-type* <thing> thing make-thing
  thing?
  this-thing
  (name  thing-name (default "chbouib"))
  (port  thing-port
         (default (current-output-port)) (thunked))
  (loc   thing-location (innate) (default (current-source-location))))
```

This example defines a macro 'thing' that can be used to instantiate records of this type:

```
(thing
  (name "foo")
  (port (current-error-port)))
```

The value of 'name' or 'port' could as well be omitted, in which case the default value specified in the 'define-record-type*' form is used:

```
(thing)
```

The 'port' field is "thunked", meaning that calls like '(thing-port x)' will actually compute the field's value in the current dynamic extent, which is useful when referring to fluids in a field's value. Furthermore, that thunk can access the record it belongs to via the 'this-thing' identifier.

A field can also be marked as "delayed" instead of "thunked", in which case its value is effectively wrapped in a (delay ...) form.

A field can also have an associated "sanitizer", which is a procedure that takes a user-supplied field value and returns a "sanitized" value for the field:

```
(define-record-type* <thing> thing make-thing
  thing?
  this-thing
  (name  thing-name
         (sanitize (lambda (value)
                     (cond ((string? value) value)
                           ((symbol? value) (symbol->string value))
                           (else (throw 'bad! value)))))))

It is possible to copy an object 'x' created with 'thing' like this:

  (thing (inherit x) (name "bar"))

This expression returns a new object equal to 'x' except for its 'name'
field and its 'loc' field---the latter is marked as "innate", so it is not
inherited.
```

`this-record` [Special Form]
Return the record being defined. This macro may only be used in the context of the definition of a thunked field.

`alist->record` *alist make keys* [*multiple-value-keys*] [Function]
Apply MAKE to the values associated with KEYS in ALIST. Items in KEYS that are also in MULTIPLE-VALUE-KEYS are considered to occur possibly multiple times in ALIST, and thus their value is a list.

`object->fields` *object fields port* [Function]
Write OBJECT (typically a record) as a series of recutils-style fields to PORT, according to FIELDS. FIELDS must be a list of field name/getter pairs.

`recutils->alist` *port* [Function]
Read a recutils-style record from PORT and return it as a list of key/value pairs. Stop upon an empty line (after consuming it) or EOF.

`match-record` *record type* (*fields ...*) *body ...* [Special Form]
Bind each FIELD of a RECORD of the given TYPE to it's FIELD name. The order in which fields appear does not matter. A syntax error is raised if an unknown field is queried.

The current implementation does not support thunked and delayed fields.

75 (guix remote)

75.1 Overview

```
Note: This API is experimental and subject to change!

Evaluate a gexp on a remote machine, over SSH, ensuring that all the
elements the gexp refers to are deployed beforehand.  This is useful for
expressions that have side effects; for pure expressions, you would rather
build a derivation remotely or offload it.
```

75.2 Usage

`remote-eval` *exp session* [*#:build-locally?*] [*#:system*] [*#:module-path*] [*#:socket-name*] [*#:become-command*] [Function]

Evaluate EXP, a gexp, on the host at SESSION, an SSH session. Ensure that all the elements EXP refers to are built and deployed to SESSION beforehand. When BUILD-LOCALLY? is true, said dependencies are built locally and sent to the remote store afterwards; otherwise, dependencies are built directly on the remote store.

76 (guix repl)

76.1 Overview

```
This module implements the "machine-readable" REPL provided by
'guix repl -t machine'.  It's a lightweight module meant to be
embedded in any Guile process providing REPL functionality.
```

76.2 Usage

`send-repl-response` *exp output* [*#:version*] [Function]

Write the response corresponding to the evaluation of EXP to PORT, an output port. VERSION is the client's protocol version we are targeting.

`machine-repl` [*input*] [*output*] [Function]

Run a machine-usable REPL over ports INPUT and OUTPUT.

The protocol of this REPL is meant to be machine-readable and provides proper support to represent multiple-value returns, exceptions, objects that lack a read syntax, and so on. As such it is more convenient and robust than parsing Guile's REPL prompt.

77 (guix)

77.1 Overview

77.2 Usage

`define-public` [Special Form]

Like 'define-public' but set 'current-definition-location' for the lexical scope of its body.

78 (guix scripts)

78.1 Overview

```
General code for Guix scripts.
```

78.2 Usage

`synopsis` [Variable]

`category` [Variable]

`define-command` (*name . args*) (*synopsis doc*) *body ...* [Special Form]
`define-command` (*name . args*) (*category cat*) (*synopsis doc*) *body ...* [Special Form]

Define the given command as a procedure along with its synopsis and, optionally, its category. The synopsis becomes the docstring of the procedure, but both the category and synopsis are meant to be read (parsed) by 'guix help'.

`%command-categories` [Variable]

`args-fold*` *args options unrecognized-option-proc operand-proc . seeds* [Function]

A wrapper on top of 'args-fold' that does proper user-facing error reporting.

`parse-command-line` *args options seeds* [*#:build-options?*] [*#:argument-handler*] [Function]

Parse the command-line arguments ARGS according to OPTIONS (a list of SRFI-37 options) and return the result, seeded by SEEDS. When BUILD-OPTIONS? is true, also pass arguments passed via the 'GUIX_BUILD_OPTIONS' environment variable. Command-line options take precedence those passed via 'GUIX_BUILD_OPTIONS'.

ARGUMENT-HANDLER is called for non-option arguments, like the 'operand-proc' parameter of 'args-fold'.

`maybe-build` *drvs* [*#:dry-run?*] [*#:use-substitutes?*] [Function]

Show what will/would be built, and actually build DRVS, unless DRY-RUN? is true.

`build-package` *package* [*#:dry-run?*] [*#:use-substitutes?*] *. build-options* [Function]

Build PACKAGE using BUILD-OPTIONS acceptable by 'set-build-options'. Show what and how will/would be built.

`build-package-source` *package* [*#:dry-run?*] [*#:use-substitutes?*] *. build-options* [Function]

Build PACKAGE source using BUILD-OPTIONS.

`%distro-age-warning` [Variable]

`warn-about-old-distro` [*old*] [*#:suggested-command*] [Function]

Emit a warning if Guix is older than OLD seconds.

`%disk-space-warning` [Variable]

`warn-about-disk-space` [*profile*] [*#:thresholds*] [Function]
Display a hint about 'guix gc' if less than THRESHOLD of /gnu/store is available. THRESHOLDS is a pair (ABSOLUTE-THRESHOLD . RELATIVE-THRESHOLD).

`development` [Variable]

`extension` [Variable]

`internal` [Variable]

`main` [Variable]

`packaging` [Variable]

`plumbing` [Variable]

79 (guix search-paths)

79.1 Overview

```
This module defines "search path specifications", which allow packages to
declare environment variables that they use to define search paths.  For
instance, GCC has the 'CPATH' variable, Guile has the 'GUILE_LOAD_PATH'
variable, etc.
```

79.2 Usage

`<search-path-specification>` [Special Form]
This macro lets us query record type info at macro-expansion time.

`search-path-specification` [Special Form]

`search-path-specification?` [Special Form]

`search-path-specification-variable` [Special Form]

`search-path-specification-files` [Special Form]

`search-path-specification-separator` [Special Form]

`search-path-specification-file-type` [Special Form]

`search-path-specification-file-pattern` [Special Form]

`$PATH` [Variable]

`$GUIX_EXTENSIONS_PATH` [Variable]

`$SSL_CERT_DIR` [Variable]

`$SSL_CERT_FILE` [Variable]

`search-path-specification->sexp` *spec* [Function]
Return an sexp representing SPEC, a <search-path-specification>. The sexp corresponds to the arguments expected by 'set-path-environment-variable'.

`sexp->search-path-specification` *sexp* [Function]
Convert SEXP, which is as returned by 'search-path-specification->sexp', to a <search-path-specification> object.

`string-tokenize*` *string separator* [Function]
Return the list of substrings of STRING separated by SEPARATOR. This is like 'string-tokenize', but SEPARATOR is a string.

`evaluate-search-paths` *search-paths directories* [*getenv*] [Function]
Evaluate SEARCH-PATHS, a list of search-path specifications, for DIRECTORIES, a list of directory names, and return a list of specification/value pairs. Use GETENV to determine the current settings and report only settings not already effective.

`environment-variable-definition` *variable value* [*#:kind*] [*#:separator*] [Function]

Return a the definition of VARIABLE to VALUE in Bash syntax.

KIND can be either 'exact (return the definition of VARIABLE=VALUE), 'prefix (return the definition where VALUE is added as a prefix to VARIABLE's current value), or 'suffix (return the definition where VALUE is added as a suffix to VARIABLE's current value.) In the case of 'prefix and 'suffix, SEPARATOR is used as the separator between VARIABLE's current value and its prefix/suffix.

`search-path-definition` *search-path value* [*#:kind*] [Function]

Similar to 'environment-variable-definition', but applied to a <search-path-specification>.

`set-search-paths` *search-paths directories* [*#:setenv*] [Function]

Set the search path environment variables specified by SEARCH-PATHS for the given directories.

80 (guix self)

80.1 Overview

80.2 Usage

`make-config.scm` [*#:gzip*] [*#:xz*] [*#:bzip2*] [*#:package-name*] [*#:package-version*] [*#:channel-metadata*] [*#:config-variables*] [*#:bug-report-address*] [*#:home-page-url*] [Function]

`whole-package` *name modules dependencies* [*#:guile-version*] [*#:info*] [*#:daemon*] [*#:miscellany*] [*#:guile*] [*#:command*] [Function]

Return the whole Guix package NAME that uses MODULES, a derivation of all the modules (under share/guile/site and lib/guile), and DEPENDENCIES, a list of packages depended on. COMMAND is the 'guix' program to use; INFO is the Info manual.

`compiled-guix` *source* [*#:version*] [*#:channel-metadata*] [*#:pull-version*] [*#:name*] [*#:guile-version*] [*#:guile-for-build*] [*#:gzip*] [*#:bzip2*] [*#:xz*] [*#:guix*] [Function]

Return a file-like object that contains a compiled Guix.

`guix-derivation` *source version* [*guile-version*] [*#:pull-version*] [*#:channel-metadata*] [Function]

Return, as a monadic value, the derivation to build the Guix from SOURCE for GUILE-VERSION. Use VERSION as the version string. Use CHANNEL-METADATA as the channel metadata sexp to include in (guix config).

PULL-VERSION specifies the version of the 'guix pull' protocol. Return #f if this PULL-VERSION value is not supported.

81 (guix serialization)

81.1 Overview

81.2 Usage

`write-int` *n p* [Function]

`read-int` *p* [Function]

`write-long-long` *n p* [Function]

`read-long-long` *p* [Function]

`write-padding` *n p* [Function]

`write-bytevector` *s p* [*l*] [Function]

`write-string` *s p* [Function]

`read-string` *p* [Function]

`read-latin1-string` *p* [Function]
Read an ISO-8859-1 string from P.

`read-maybe-utf8-string` *p* [Function]
Read a serialized string from port P. Attempt to decode it as UTF-8 and substitute invalid byte sequences with question marks. This is a "permissive" UTF-8 decoder.

`write-string-list` *l p* [Function]

`read-string-list` *p* [Function]

`write-string-pairs` *l p* [Function]

`read-string-pairs` *p* [Function]

`write-store-path` *f p* [Function]

`read-store-path` *p* [Function]

`write-store-path-list` *l p* [Function]

`read-store-path-list` *p* [Function]

`&nar-error` [Variable]

`nar-error?` *obj* [Function]

`nar-error-port` *obj* [Function]

`nar-error-file` *obj* [Function]

`&nar-read-error` [Variable]

`nar-read-error?` *obj* [Function]

`nar-read-error-token` *obj* [Function]

`write-file` *file port* [*#:select?*] [Function]
Write the contents of FILE to PORT in Nar format, recursing into sub-directories of FILE as needed. For each directory entry, call (SELECT? FILE STAT), where FILE is the entry's absolute file name and STAT is the result of 'lstat'; exclude entries for which SELECT? does not return true.

`write-file-tree` *file port* [*#:file-type+size*] [*#:file-port*] [Function]
[*#:symlink-target*] [*#:directory-entries*] [*#:postprocess-entries*]
Write the contents of FILE to PORT in Nar format, recursing into sub-directories of FILE as needed.

This procedure does not make any file-system I/O calls. Instead, it calls the user-provided FILE-TYPE+SIZE, FILE-PORT, SYMLINK-TARGET, and DIRECTORY-ENTRIES procedures, which roughly correspond to 'lstat', 'readlink', and 'scandir'. POSTPROCESS-ENTRIES ensures that directory entries are valid; leave it as-is unless you know that DIRECTORY-ENTRIES provide filtered and sorted entries, in which case you can use 'identity'.

`fold-archive` *proc seed port file* [Function]
```
Read a file (possibly a directory structure) in Nar format from PORT.  Call
PROC on each file or directory read from PORT using:

  (PROC FILE TYPE CONTENTS RESULT)

using SEED as the first RESULT.  TYPE is a symbol like 'regular, and CONTENTS
depends on TYPE.
```

`restore-file` *port file* [*#:dump-file*] [Function]
```
Read a file (possibly a directory structure) in Nar format from PORT.
Restore it as FILE with canonical permissions and timestamps.  To write a
regular or executable file, call:

  (DUMP-FILE FILE INPUT SIZE TYPE)

The default is to dump SIZE bytes from INPUT to FILE, but callers can provide
a custom procedure, for instance to deduplicate FILE on the fly.
```

`dump-file` *file input size type* [Function]
Dump SIZE bytes from INPUT to FILE.

This procedure is suitable for use as the #:dump-file argument to 'restore-file'.

`dump-port*` *in out size* [Function]
Copy SIZE bytes from IN to OUT.

82 (guix sets)

82.1 Overview

```
A simple (simplistic?) implementation of unordered persistent sets based
on vhashes that seems to be good enough so far.

Another option would be to use "bounded balance trees" (Adams 1992) as
implemented by Ian Price in 'pfds', which has faster union etc. but needs
an order on the objects of the set.
```

82.2 Usage

`set` . *args* [Function]
Return a set containing the ARGS, compared as per 'equal?'.

`setq` . *args* [Function]
Return a set containing the ARGS, compared as per 'eq?'.

`set?` [Special Form]

`set-insert` *value set* [Function]
Insert VALUE into SET.

`set-union` *set1 set2* [Function]
Return the union of SET1 and SET2. Warning: this is linear in the number of elements of the smallest.

`set-contains?` [Special Form]

`set->list` *set* [Function]
Return the list of elements of SET.

`list->set` *lst* [Function]
Return a set with the elements taken from LST. Elements of the set will be compared with 'equal?'.

`list->setq` *lst* [Function]
Return a set with the elements taken from LST. Elements of the set will be compared with 'eq?'.

83 (guix ssh)

83.1 Overview

This module provides tools to support communication with remote stores over SSH, using Guile-SSH.

83.2 Usage

`open-ssh-session` *host* [*#:user*] [*#:port*] [*#:identity*] [*#:host-key*] [*#:compression*] [*#:timeout*] [*#:connection-timeout*] [Function]

Open an SSH session for HOST and return it. IDENTITY specifies the file name of a private key to use for authenticating with the host. When USER, PORT, or IDENTITY are #f, use default values or whatever '~/.ssh/config' specifies; otherwise use them.

When HOST-KEY is true, it must be a string like "ssh-ed25519 AAAAC3Nz... root@@example.org"; the server is authenticated and an error is raised if its host key is different from HOST-KEY.

Error out if connection establishment takes more than CONNECTION-TIMEOUT seconds. Install TIMEOUT as the maximum time in seconds after which a read or write operation on a channel of the returned session is considered as failing.

Throw an error on failure.

`authenticate-server*` *session key* [Function]

Make sure the server for SESSION has the given KEY, where KEY is a string such as "ssh-ed25519 AAAAC3Nz... root@@example.org". Raise an exception if the actual key does not match.

`remote-inferior` *session* [*become-command*] [Function]

Return a remote inferior for the given SESSION. If BECOME-COMMAND is given, use that to invoke the remote Guile REPL.

`remote-daemon-channel` *session* [*socket-name*] [Function]

Return an input/output port (an SSH channel) to the daemon at SESSION.

`connect-to-remote-daemon` *session* [*socket-name*] [Function]

Connect to the remote build daemon listening on SOCKET-NAME over SESSION, an SSH session. Return a <store-connection> object.

`remote-system` *session* [Function]

Return the system type as expected by Nix, usually ARCHITECTURE-KERNEL, of the machine on the other end of SESSION.

`remote-authorize-signing-key` *key session* [*become-command*] [Function]

Send KEY, a canonical sexp containing a public key, over SESSION and add it to the system ACL file if it has not yet been authorized.

`send-files` *local files remote* [*#:recursive?*] [*#:log-port*] [Function]
Send the subset of FILES from LOCAL (a local store) that's missing to REMOTE, a remote store. When RECURSIVE? is true, send the closure of FILES. Return the list of store items actually sent.

`retrieve-files` *local files remote* [*#:recursive?*] [*#:log-port*] [Function]
Retrieve FILES from REMOTE and import them using the 'import-paths' RPC on LOCAL. When RECURSIVE? is true, retrieve the closure of FILES.

`retrieve-files*` *files remote* [*#:recursive?*] [*#:log-port*] [*#:import*] [Function]
Pass IMPORT an input port from which to read the sequence of FILES coming from REMOTE. When RECURSIVE? is true, retrieve the closure of FILES.

`remote-store-host` *remote* [Function]
Return the name of the host REMOTE is connected to, where REMOTE is a remote store as returned by 'connect-to-remote-daemon'.

`report-guile-error` *host* [Function]

84 (guix status)

84.1 Overview

This module provides facilities to track the status of ongoing builds and downloads in a given session, as well as tools to report about the current status to user interfaces. It does so by analyzing the output of 'current-build-output-port'. The build status is maintained in a <build-status> record.

84.2 Usage

`build-event-output-port` *proc* [*seed*] [Function]

Return an output port for use as 'current-build-output-port' that calls PROC with its current state value, initialized with SEED, on every build event. Build events passed to PROC are tuples corresponding to the "build traces" produced by the daemon:

```
(build-started "/gnu/store/...-foo.drv" ...)
(substituter-started "/gnu/store/...-foo" ...)
```

and so on.

The second return value is a thunk to retrieve the current state.

`compute-status` *event status* [*#:current-time*] [Function]
[*#:derivation-path->output-path*]

Given EVENT, a tuple like (build-started "/gnu/store/...-foo.drv" ...), compute a new status based on STATUS.

`build-status` [Special Form]

`build-status?` [Special Form]

`build-status-building` [Special Form]

`build-status-downloading` [Special Form]

`build-status-builds-completed` [Special Form]

`build-status-downloads-completed` [Special Form]

`build?` [Special Form]

`build` *derivation system* [*#:id*] [*#:log-file*] [*#:phase*] [*#:completion*] [Function]

Return a new build.

`build-derivation` [Special Form]

`build-system` [Special Form]

`build-log-file` [Special Form]

`build-phase` [Special Form]

`build-completion` [Special Form]

`download?` [Special Form]

`download` *item uri* [*#:size*] [*#:start*] [*#:end*] [*#:transferred*] [Function]
Return a new download.

`download-item` [Special Form]

`download-uri` [Special Form]

`download-size` [Special Form]

`download-start` [Special Form]

`download-end` [Special Form]

`download-transferred` [Special Form]

`build-status-updater` [*on-change*] [Function]
```
Return a procedure that can be passed to 'build-event-output-port'.  That
procedure computes the new build status upon each event and calls ON-CHANGE:

  (ON-CHANGE event status new-status)

ON-CHANGE can display the build status, build events, etc.
```

`print-build-event` *event old-status status* [*port*] [*#:colorize?*] [*#:print-urls?*] [*#:print-log?*] [Function]
Print information about EVENT and STATUS to PORT. When COLORIZE? is true, produce colorful output. When PRINT-LOG? is true, display the build log in addition to build events. When PRINT-URLS? is true, display the URL of substitutes being downloaded.

`print-build-event/quiet` *event old-status status* [*port*] [*#:colorize?*] [Function]

`print-build-status` [Variable]
[unbound!]

`with-status-report` *on-event exp ...* [Special Form]
Set up build status reporting to the user using the ON-EVENT procedure; evaluate EXP... in that context.

`with-status-verbosity` *level exp ...* [Special Form]
Set up build status reporting to the user at the given LEVEL: 0 means silent, 1 means quiet, 2 means verbose. Evaluate EXP... in that context.

85 (guix store)

85.1 Overview

85.2 Usage

`%daemon-socket-uri` [Variable]

`%gc-roots-directory` [Variable]

`%default-substitute-urls` [Variable]

`store-connection?` [Special Form]

`store-connection-version` *store* [Function]
Return the protocol version of STORE as an integer.

`store-connection-major-version` [Special Form]

`store-connection-minor-version` [Special Form]

`store-connection-socket` [Special Form]

`nix-server?` [Special Form]

`nix-server-version` [Special Form]

`nix-server-major-version` [Special Form]

`nix-server-minor-version` [Special Form]

`nix-server-socket` [Special Form]

`current-store-protocol-version` [Variable]

`cache-lookup-recorder` *component title* [Function]
Return a procedure of two arguments to record cache lookups, hits, and misses for COMPONENT. The procedure must be passed a Boolean indicating whether the cache lookup was a hit, and the actual cache (a vhash).

`mcached` *eq?* (`=>` *cache*) *mvalue object keys ...* [Special Form]
`mcached` *equal?* (`=>` *cache*) *mvalue object keys ...* [Special Form]
`mcached` *eq? mvalue object keys ...* [Special Form]
`mcached` *equal? mvalue object keys ...* [Special Form]
`mcached` *mvalue object keys ...* [Special Form]
Run MVALUE, which corresponds to OBJECT/KEYS, and cache it; or return the value associated with OBJECT/KEYS in the store's object cache if there is one.

`&store-error` [Variable]

`store-error?` *obj* [Function]

`&store-connection-error` [Variable]

`store-connection-error?` *obj* [Function]

`store-connection-error-file` *obj* [Function]

`store-connection-error-code` *obj* [Function]

`&store-protocol-error` [Variable]

`store-protocol-error?` *obj* [Function]

`store-protocol-error-message` *obj* [Function]

`store-protocol-error-status` *obj* [Function]

`&nix-error` [Special Form]

`nix-error?` [Special Form]

`&nix-connection-error` [Special Form]

`nix-connection-error?` [Special Form]

`nix-connection-error-file` [Special Form]

`nix-connection-error-code` [Special Form]

`&nix-protocol-error` [Special Form]

`nix-protocol-error?` [Special Form]

`nix-protocol-error-message` [Special Form]

`nix-protocol-error-status` [Special Form]

`allocate-store-connection-cache` *name* [Function]
Allocate a new cache for store connections and return its identifier. Said identifier can be passed as an argument to

`store-connection-cache` *store cache* [Function]
Return the cache of STORE identified by CACHE, an identifier as returned by 'allocate-store-connection-cache'.

`set-store-connection-cache` *store cache value* [Function]
Return a copy of STORE where CACHE has the given VALUE. CACHE must be a value returned by 'allocate-store-connection-cache'.

`set-store-connection-cache!` *store cache value* [Function]
Set STORE's CACHE to VALUE.

This is a mutating version that should be avoided. Prefer the functional 'set-store-connection-cache' instead, together with using %STORE-MONAD.

`hash-algo` *md5* [Special Form]
`hash-algo` *sha1* [Special Form]
`hash-algo` *sha256* [Special Form]

`build-mode` *normal* [Special Form]
`build-mode` *repair* [Special Form]

`build-mode` *check* [Special Form]

`connect-to-daemon` *uri* [Function]
Connect to the daemon at URI, a string that may be an actual URI or a file name, and return an input/output port.

This is a low-level procedure that does not perform the initial handshake with the daemon. Use 'open-connection' for that.

`open-connection` [*uri*] [*#:port*] [*#:reserve-space?*] [*#:cpu-affinity*] [Function]
Connect to the daemon at URI (a string), or, if PORT is not #f, use it as the I/O port over which to communicate to a build daemon.

When RESERVE-SPACE? is true, instruct it to reserve a little bit of extra space on the file system so that the garbage collector can still operate, should the disk become full. When CPU-AFFINITY is true, it must be an integer corresponding to an OS-level CPU number to which the daemon's worker process for this connection will be pinned. Return a server object.

`port->connection` *port* [*#:version*] [Function]
Assimilate PORT, an input/output port, and return a connection to the daemon, assuming the given protocol VERSION.

Warning: this procedure assumes that the initial handshake with the daemon has already taken place on PORT and that we're just continuing on this established connection. Use with care.

`close-connection` *server* [Function]
Close the connection to SERVER.

`with-store` *store exp ...* [Special Form]
Bind STORE to an open connection to the store and evaluate EXPs; automatically close the store when the dynamic extent of EXP is left.

`set-build-options` *server* [*#:keep-failed?*] [*#:keep-going?*] [Function]
[*#:fallback?*] [*#:verbosity*] [*#:rounds*] [*#:max-build-jobs*] [*#:timeout*]
[*#:max-silent-time*] [*#:offload?*] [*#:use-build-hook?*] [*#:build-verbosity*]
[*#:log-type*] [*#:print-build-trace*] [*#:user-name*]
[*#:print-extended-build-trace?*] [*#:multiplexed-build-output?*]
[*#:build-cores*] [*#:use-substitutes?*] [*#:substitute-urls*]
[*#:terminal-columns*] [*#:locale*]

`set-build-options*` . *args* [Function]

`valid-path?` *server path* [Function]
Return #t when PATH designates a valid store item and #f otherwise (an invalid item may exist on disk but still be invalid, for instance because it is the result of an aborted or failed build.)

A '&store-protocol-error' condition is raised if PATH is not prefixed by the store directory (/gnu/store).

`query-path-hash` *server path* [Function]
Return the SHA256 hash of the nar serialization of PATH as a bytevector.

`hash-part->path` *server hash-part* [Function]
Return the store path whose hash part is HASH-PART (a nix-base32 string). Return the empty string if no such path exists.

`query-path-info` *server path* [Function]
Return the info (hash, references, etc.) for PATH.

`add-data-to-store` *server name bytes* [*references*] [Function]
Add BYTES under file NAME in the store, and return its store path. REFERENCES is the list of store paths referred to by the resulting store path.

`add-text-to-store` *store name text* [*references*] [Function]
Add TEXT under file NAME in the store, and return its store path. REFERENCES is the list of store paths referred to by the resulting store path.

`add-to-store` *server basename recursive? hash-algo file-name* [Function]
[*#:select?*]
Add the contents of FILE-NAME under BASENAME to the store. When RECURSIVE? is false, FILE-NAME must designate a regular file–not a directory nor a symlink. When RECURSIVE? is true and FILE-NAME designates a directory, the contents of FILE-NAME are added recursively; if FILE-NAME designates a flat file and RECURSIVE? is true, its contents are added, and its permission bits are kept. HASH-ALGO must be a string such as "sha256".

When RECURSIVE? is true, call (SELECT? FILE STAT) for each directory entry, where FILE is the entry's absolute file name and STAT is the result of 'lstat'; exclude entries for which SELECT? does not return true.

`add-file-tree-to-store` *server tree* [*#:hash-algo*] [*#:recursive?*] [Function]
Add the given TREE to the store on SERVER. TREE must be an entry such as:

```
("my-tree" directory
  ("a" regular (data "hello"))
  ("b" symlink "a")
  ("c" directory
    ("d" executable (file "/bin/sh"))))
```

This is a generalized version of 'add-to-store'. It allows you to reproduce an arbitrary directory layout in the store without creating a derivation.

`file-mapping->tree` *mapping* [Function]
Convert MAPPING, an alist like:

```
(("guix/build/utils.scm" . ".../utils.scm"))
```

to a tree suitable for 'add-file-tree-to-store' and 'interned-file-tree'.

`binary-file` *name data* [*references*] [Function]
Return as a monadic value the absolute file name in the store of the file containing DATA, a bytevector. REFERENCES is a list of store items that the resulting text file refers to; it defaults to the empty list.

`with-build-handler` *handler exp ...* [Special Form]

```
Register HANDLER as a "build handler" and invoke THUNK.  When
'build-things' is called within the dynamic extent of the call to THUNK,
HANDLER is invoked like so:

  (HANDLER CONTINUE STORE THINGS MODE)

where CONTINUE is the continuation, and the remaining arguments are those that
were passed to 'build-things'.

Build handlers are useful to announce a build plan with 'show-what-to-build'
and to implement dry runs (by not invoking CONTINUE) in a way that gracefully
deals with "dynamic dependencies" such as grafts---derivations that depend
on the build output of a previous derivation.
```

`map/accumulate-builds` *store proc lst* [*#:cutoff*] [Function]

Apply PROC over each element of LST, accumulating 'build-things' calls and coalescing them into a single call.

CUTOFF is the threshold above which we stop accumulating unresolved nodes.

`mapm/accumulate-builds` *mproc lst* [Function]

Like 'mapm' in %STORE-MONAD, but accumulate 'build-things' calls and coalesce them into a single call.

`build-things` *store things* [*mode*] [Function]

Build THINGS, a list of store items which may be either '.drv' files or outputs, and return when the worker is done building them. Elements of THINGS that are not derivations can only be substituted and not built locally. Alternately, an element of THING can be a derivation/output name pair, in which case the daemon will attempt to substitute just the requested output of the derivation. Return #t on success.

When a handler is installed with 'with-build-handler', it is called any time 'build-things' is called.

`build` . *args* [Function]

Build THINGS, a list of store items which may be either '.drv' files or outputs, and return when the worker is done building them. Elements of THINGS that are not derivations can only be substituted and not built locally. Alternately, an element of THING can be a derivation/output name pair, in which case the daemon will attempt to substitute just the requested output of the derivation. Return #t on success.

When a handler is installed with 'with-build-handler', it is called any time 'build-things' is called.

`query-failed-paths` *server* [Function]

Return the list of store items for which a build failure is cached.

The result is always the empty list unless the daemon was started with '–cache-failures'.

`clear-failed-paths` *server items* [Function]
Remove ITEMS from the list of cached build failures.

This makes sense only when the daemon was started with '–cache-failures'.

`ensure-path` *server path* [Function]
Ensure that a path is valid. If it is not valid, it may be made valid by running a substitute. As a GC root is not created by the daemon, you may want to call ADD-TEMP-ROOT on that store path.

`find-roots` *server* [Function]
Return a list of root/target pairs: for each pair, the first element is the GC root file name and the second element is its target in the store.

When talking to a local daemon, this operation is equivalent to the 'gc-roots' procedure in (guix store roots), except that the 'find-roots' excludes potential roots that do not point to store items.

`add-temp-root` *server path* [Function]
Make PATH a temporary root for the duration of the current session. Return #t.

`add-indirect-root` *server file-name* [Function]
Make the symlink FILE-NAME an indirect root for the garbage collector: whatever store item FILE-NAME points to will not be collected. Return #t on success.

FILE-NAME can be anywhere on the file system, but it must be an absolute file name–it is the caller's responsibility to ensure that it is an absolute file name.

`add-permanent-root` *target* [Function]
Add a garbage collector root pointing to TARGET, an element of the store, preventing TARGET from even being collected. This can also be used if TARGET does not exist yet.

Raise an error if the caller does not have write access to the GC root directory.

`remove-permanent-root` *target* [Function]
Remove the permanent garbage collector root pointing to TARGET. Raise an error if there is no such root.

`substitutable?` [Special Form]

`substitutable-path` [Special Form]

`substitutable-deriver` [Special Form]

`substitutable-references` [Special Form]

`substitutable-download-size` [Special Form]

`substitutable-nar-size` [Special Form]

`has-substitutes?` *server path* [Function]
Return #t if binary substitutes are available for PATH, and #f otherwise.

`substitutable-paths` *server paths* [Function]
Return the subset of PATHS that is substitutable.

`substitutable-path-info` *server paths* [Function]
Return information about the subset of PATHS that is substitutable. For each substitutable path, a 'substitutable?' object is returned; thus, the resulting list can be shorter than PATHS. Furthermore, that there is no guarantee that the order of the resulting list matches the order of PATHS.

`path-info?` [Special Form]

`path-info-deriver` [Special Form]

`path-info-hash` [Special Form]

`path-info-references` [Special Form]

`path-info-registration-time` [Special Form]

`path-info-nar-size` [Special Form]

`built-in-builders` *store* [Function]
Return the names of the supported built-in derivation builders supported by STORE.

`references` *server path* [Function]
Return the list of references of PATH.

`references/cached` *store item* [Function]
Like 'references', but cache results.

`references*` . *args* [Function]
Return the list of references of PATH.

`query-path-info*` *item* [Function]
Monadic version of 'query-path-info' that returns #f when ITEM is not in the store.

`requisites` *store paths* [Function]
Return the requisites of PATHS, including PATHS—i.e., their closures (all its references, recursively).

`referrers` *server path* [Function]
Return the list of path that refer to PATH.

`optimize-store` *server* [Function]
Optimize the store by hard-linking identical files (`"deduplication"`.) Return #t on success.

`verify-store` *store* [*#:check-contents?*] [*#:repair?*] [Function]
Verify the integrity of the store and return false if errors remain, and true otherwise. When REPAIR? is true, repair any missing or altered store items by substituting them (this typically requires root privileges because it is not an atomic operation.) When CHECK-CONTENTS? is true, check the contents of store items; this can take a lot of time.

`topologically-sorted` *store paths* [Function]
Return a list containing PATHS and all their references sorted in topological order.

`valid-derivers` *server path* [Function]
Return the list of valid "derivers" of PATH—i.e., all the .drv present in the store that have PATH among their outputs.

`query-derivation-outputs` *server path* [Function]
Return the list of outputs of PATH, a .drv file.

`live-paths` *server* [Function]
Return the list of live store paths—i.e., store paths still referenced, and thus not subject to being garbage-collected.

`dead-paths` *server* [Function]
Return the list of dead store paths—i.e., store paths no longer referenced, and thus subject to being garbage-collected.

`collect-garbage` *server* [*min-freed*] [Function]
Collect garbage from the store at SERVER. If MIN-FREED is non-zero, then collect at least MIN-FREED bytes. Return the paths that were collected, and the number of bytes freed.

`delete-paths` *server paths* [*min-freed*] [Function]
Delete PATHS from the store at SERVER, if they are no longer referenced. If MIN-FREED is non-zero, then stop after at least MIN-FREED bytes have been collected. Return the paths that were collected, and the number of bytes freed.

`import-paths` *server port* [Function]
Import the set of store paths read from PORT into SERVER's store. An error is raised if the set of paths read from PORT is not signed (as per 'export-path #:sign? #t'.) Return the list of store paths imported.

`export-paths` *server paths port* [*#:sign?*] [*#:recursive?*] [*#:start*] [*#:progress*] [*#:finish*] [Function]
Export the store paths listed in PATHS to PORT, in topological order, signing them if SIGN? is true. When RECURSIVE? is true, export the closure of PATHS—i.e., PATHS and all their dependencies.

START, PROGRESS, and FINISH are used to track progress of the data transfer. START is a one-argument that is passed the list of store items that will be transferred; it returns values that are then used as the initial state threaded through PROGRESS calls. PROGRESS is passed the store item about to be sent, along with the values previously return by START or by PROGRESS itself. FINISH is called when the last store item has been called.

`current-build-output-port` [Variable]

`%store-monad` [Special Form]

`store-bind` [Special Form]

`store-return` [Special Form]

`store-lift` *proc* [Function]
Lift PROC, a procedure whose first argument is a connection to the store, in the store monad.

`store-lower` *proc* [Function]
Lower PROC, a monadic procedure in %STORE-MONAD, to a "normal" procedure taking the store as its first argument.

`run-with-store` *store mval* [*#:guile-for-build*] [*#:system*] [*#:target*] [Function]
Run MVAL, a monadic value in the store monad, in STORE, an open store connection, and return the result.

`%guile-for-build` [Variable]

`current-system` [Special Form]

`set-current-system` [Special Form]

`current-target-system` [Special Form]

`set-current-target` [Special Form]

`text-file` *name text* [*references*] [Function]
Return as a monadic value the absolute file name in the store of the file containing TEXT, a string. REFERENCES is a list of store items that the resulting text file refers to; it defaults to the empty list.

`interned-file` *file* [*name*] [*#:recursive?*] [*#:select?*] [Function]
Return the name of FILE once interned in the store. Use NAME as its store name, or the basename of FILE if NAME is omitted.

When RECURSIVE? is true, the contents of FILE are added recursively; if FILE designates a flat file and RECURSIVE? is true, its contents are added, and its permission bits are kept.

When RECURSIVE? is true, call (SELECT? FILE STAT) for each directory entry, where FILE is the entry's absolute file name and STAT is the result of 'lstat'; exclude entries for which SELECT? does not return true.

`interned-file-tree` . *args* [Function]
`Add the given TREE to the store on SERVER.  TREE must be an entry such as:`

```
("my-tree" directory
  ("a" regular (data "hello"))
  ("b" symlink "a")
  ("c" directory
    ("d" executable (file "/bin/sh"))))
```

`This is a generalized version of 'add-to-store'.  It allows you to reproduce an arbitrary directory layout in the store without creating a derivation.`

`%graft?` [Variable]

`without-grafting` *mexp ...* [Special Form]
Bind monadic expressions MEXP in a dynamic extent where '%graft?' is false.

`set-grafting` [Special Form]

`grafting?` [Special Form]

`%store-prefix` [Variable]

`store-path` *type hash name* [Function]
Return the store path for NAME/HASH/TYPE.

`output-path` *output hash name* [Function]
Return an output path for OUTPUT (the name of the output as a string) of the derivation called NAME with hash HASH.

`fixed-output-path` *name hash* [*#:output*] [*#:hash-algo*] [Function]
[*#:recursive?*]
Return an output path for the fixed output OUTPUT defined by HASH of type HASH-ALGO, of the derivation NAME. RECURSIVE? has the same meaning as for 'add-to-store'.

`store-path?` *path* [Function]
Return #t if PATH is a store path.

`direct-store-path?` *path* [Function]
Return #t if PATH is a store path, and not a sub-directory of a store path. This predicate is sometimes needed because files *under* a store path are not valid inputs.

`derivation-path?` *path* [Function]
Return #t if PATH is a derivation path.

`store-path-base` *path* [Function]
Return the base path of a path in the store.

`store-path-package-name` *path* [Function]
Return the package name part of PATH, a file name in the store.

`store-path-hash-part` *path* [Function]
Return the hash part of PATH as a base32 string, or #f if PATH is not a syntactically valid store path.

`direct-store-path` *path* [Function]
Return the direct store path part of PATH, stripping components after '/gnu/store/xxxx-foo'.

`derivation-log-file` *drv* [Function]
Return the build log file for DRV, a derivation file name, or #f if it could not be found.

`log-file` *store file* [Function]
Return the build log file for FILE, or #f if none could be found. FILE must be an absolute store file name, or a derivation file name.

86 (guix substitutes)

86.1 Overview

86.2 Usage

`%narinfo-cache-directory` [Variable]

`call-with-connection-error-handling` *uri proc* [Function]
Call PROC, and catch if a connection fails, print a warning and return #f.

`lookup-narinfos` *cache paths* [*#:open-connection*] [Function]
[*#:make-progress-reporter*]
Return the narinfos for PATHS, invoking the server at CACHE when no information is available locally.

`lookup-narinfos/diverse` *caches paths authorized?* [Function]
[*#:open-connection*] [*#:make-progress-reporter*]
Look up narinfos for PATHS on all of CACHES, a list of URLS, in that order. That is, when a cache lacks an AUTHORIZED? narinfo, look it up in the next cache, and so on.

Return a list of narinfos for PATHS or a subset thereof. The returned narinfos are either AUTHORIZED?, or they claim a hash that matches an AUTHORIZED? narinfo.

87 (guix svn-download)

87.1 Overview

An <origin> method that fetches a specific revision from a Subversion repository. The repository URL and REVISION are specified with a <svn-reference> object. REVISION should be specified as a number.

87.2 Usage

`svn-reference` [Special Form]

`svn-reference?` [Special Form]

`svn-reference-url` [Special Form]

`svn-reference-revision` [Special Form]

`svn-reference-recursive?` [Special Form]

`svn-reference-user-name` [Special Form]

`svn-reference-password` [Special Form]

`svn-fetch` *ref hash-algo hash* [*name*] [*#:system*] [*#:guile*] [*#:svn*] [Function]
Return a fixed-output derivation that fetches REF, a <svn-reference> object. The output is expected to have recursive hash HASH of type HASH-ALGO (a symbol). Use NAME as the file name, or a generic name if #f.

`download-svn-to-store` *store ref* [*name*] [*#:log*] [Function]
Download from REF, a <svn-reference> object to STORE. Write progress reports to LOG.

`svn-multi-reference` [Special Form]

`svn-multi-reference?` [Special Form]

`svn-multi-reference-url` [Special Form]

`svn-multi-reference-revision` [Special Form]

`svn-multi-reference-locations` [Special Form]

`svn-multi-reference-recursive?` [Special Form]

`svn-multi-reference-user-name` [Special Form]

`svn-multi-reference-password` [Special Form]

`svn-multi-fetch` *ref hash-algo hash* [*name*] [*#:system*] [*#:guile*] [*#:svn*] [Function]
Return a fixed-output derivation that fetches REF, a <svn-multi-reference> object. The output is expected to have recursive hash HASH of type HASH-ALGO (a symbol). Use NAME as the file name, or a generic name if #f.

`download-multi-svn-to-store` *store ref* [*name*] [*#:log*] [Function]
Download from REF, a <svn-multi-reference> object to STORE. Write progress reports to LOG.

88 (guix swh)

88.1 Overview

```
This module provides bindings to the HTTP interface of Software Heritage.
It allows you to browse the archive, look up revisions (such as SHA1
commit IDs), "origins" (code hosting URLs), content (files), etc.  See
<https://archive.softwareheritage.org/api/> for more information.

The high-level 'swh-download' procedure allows you to download a Git
revision from Software Heritage, provided it is available.
```

88.2 Usage

`%swh-base-url` [Variable]

`%verify-swh-certificate?` [Variable]

`%allow-request?` [Variable]

`request-rate-limit-reached?` *url method* [Function]
Return true if the rate limit has been reached for URI.

`origin?` [Special Form]

`origin-type` [Special Form]

`origin-url` [Special Form]

`origin-visits` *origin* [Function]
Return the list of visits of ORIGIN, a record as returned by 'lookup-origin'.

`lookup-origin` *url* [Function]
Return an origin for URL.

`visit?` [Special Form]

`visit-date` [Special Form]

`visit-origin` [Special Form]

`visit-url` [Special Form]

`visit-snapshot-url` [Special Form]

`visit-status` [Special Form]

`visit-number` [Special Form]

`visit-snapshot` *visit* [Function]
Return the snapshot corresponding to VISIT or #f if no snapshot is available.

`snapshot?` [Special Form]

`snapshot-id` [Special Form]

`snapshot-branches` [Special Form]

`lookup-snapshot-branch` *snapshot name* [Function]
Look up branch NAME on SNAPSHOT. Return the branch, or return #f if it could not be found.

`branch?` [Special Form]

`branch-name` [Special Form]

`branch-target` *branch* [Function]
Return the target of BRANCH, either a <revision> or a <release>.

`release?` [Special Form]

`release-id` [Special Form]

`release-name` [Special Form]

`release-message` [Special Form]

`release-target` *release* [Function]
Return the revision that is the target of RELEASE.

`revision?` [Special Form]

`revision-id` [Special Form]

`revision-date` [Special Form]

`revision-directory` [Special Form]

`lookup-revision` *id* [Function]
Return the revision with the given ID, typically a Git commit SHA1.

`lookup-origin-revision` *url tag* [Function]
Return a <revision> corresponding to the given TAG for the repository coming from URL. Example:

```
(lookup-origin-revision "https://github.com/guix-mirror/guix/" "v0.8")
=> #<<revision> id: "44941..." ...>
```

The information is based on the latest visit of URL available. Return #f if URL could not be found.

`content?` [Special Form]

`content-checksums` [Special Form]

`content-data-url` [Special Form]

`content-length` [Special Form]

`lookup-content` *hash type* [Function]
Return a content for HASH, of the given TYPE–e.g., "sha256".

`directory-entry?` [Special Form]

`directory-entry-name` [Special Form]

`directory-entry-type` [Special Form]

`directory-entry-checksums` [Special Form]

`directory-entry-length` [Special Form]

`directory-entry-permissions` [Special Form]

`lookup-directory` *id* [Function]
Return the directory with the given ID.

`directory-entry-target` *entry* [Function]
If ENTRY, a directory entry, has type 'directory, return its list of directory entries; if it has type 'file, return its <content> object.

`save-reply?` [Special Form]

`save-reply-origin-url` [Special Form]

`save-reply-origin-type` [Special Form]

`save-reply-request-date` [Special Form]

`save-reply-request-status` [Special Form]

`save-reply-task-status` [Special Form]

`save-origin` *url* [*type*] [Function]
Request URL to be saved.

`save-origin-status` *url type* [Function]
Return the status of a /save request for URL and TYPE (e.g., "git").

`vault-reply?` [Special Form]

`vault-reply-id` [Special Form]

`vault-reply-fetch-url` [Special Form]

`vault-reply-progress-message` [Special Form]

`vault-reply-status` [Special Form]

`vault-reply-swhid` [Special Form]

`query-vault` *id* [*kind*] [*#:archive-type*] [Function]
Ask the availability of object ID (an SWHID) to the vault. Return #f if it could not be found, or a <vault-reply> on success. ARCHIVE-TYPE can be 'flat for a tarball containing a directory, or 'git-bare for a tarball containing a bare Git repository corresponding to a revision.

Passing KIND (one of 'directory or 'revision) together with a raw revision or directory identifier is deprecated.

`request-cooking` *id* [*kind*] [*#:archive-type*] [Function]
Request the cooking of object ID, an SWHID. Return a <vault-reply>. ARCHIVE-TYPE can be 'flat for a tarball containing a directory, or 'git-bare for a tarball containing a bare Git repository corresponding to a revision.

Passing KIND (one of 'directory or 'revision) together with a raw revision or directory identifier is deprecated.

`vault-fetch` *id* [*kind*] [*#:archive-type*] [*#:log-port*] [Function]
Return an input port from which a bundle of the object with the given ID, an SWHID, or #f if the object could not be found.

ARCHIVE-TYPE can be 'flat for a tarball containing a directory, or 'git-bare for a tarball containing a bare Git repository corresponding to a revision.

`commit-id?` *reference* [Function]
Return true if REFERENCE is likely a commit ID, false otherwise—e.g., if it is a tag name. This is based on a simple heuristic so use with care!

`swh-download-directory` *id output* [*#:log-port*] [Function]
Download from Software Heritage the directory with the given ID, and unpack it to OUTPUT. Return #t on success and #f on failure.

`swh-download` *url reference output* [*#:archive-type*] [*#:log-port*] [Function]
Download from Software Heritage a checkout (if ARCHIVE-TYPE is 'flat) or a full Git repository (if ARCHIVE-TYPE is 'git-bare) of the Git tag or commit REFERENCE originating from URL, and unpack it in OUTPUT. Return #t on success and #f on failure.

This procedure uses the "vault", which contains "cooked" directories in the form of tarballs. If the requested directory is not cooked yet, it will wait until it becomes available, which could take several minutes.

89 (guix transformations)

89.1 Overview

This module implements "package transformation options"---tools for package graph rewriting. It contains the graph rewriting logic, but also the tip of its user interface: command-line option handling.

89.2 Usage

`options->transformation` *opts* [Function]

Return a procedure that, when passed an object to build (package, derivation, etc.), applies the transformations specified by OPTS and returns the resulting objects. OPTS must be a list of symbol/string pairs such as:

```
((with-branch . "guile-gcrypt=master")
 (without-tests . "libgcrypt"))
```

Each symbol names a transformation and the corresponding string is an argument to that transformation.

`manifest-entry-with-transformations` *entry* [Function]

Return ENTRY with an additional 'transformations' property if it's not already there.

`tunable-package?` *package* [Function]

Return true if package PACKAGE is "tunable"–i.e., if tuning it for the host CPU is worthwhile.

`tuned-package` *p micro-architecture* [Function]

Return package P tuned for MICRO-ARCHITECTURE.

`show-transformation-options-help` [Function]

Show basic help for package transformation options.

`transformation-option-key?` *key* [Function]

Return true if KEY is an option key (as returned while parsing options with %TRANSFORMATION-OPTIONS) corresponding to a package transformation option. For example, (transformation-option-key? 'with-input) => #t.

`%transformation-options` [Variable]

90 (guix ui)

90.1 Overview

```
User interface facilities for command-line tools.
```

90.2 Usage

`display-hint` *message* [*port*] [Function]
Display MESSAGE, a l10n message possibly containing Texinfo markup, to PORT.

`make-user-module` *modules* [Function]
Return a new user module with the additional MODULES loaded.

`load*` *file user-module* [*#:on-error*] [Function]
Load the user provided Scheme source code FILE.

`warn-about-load-error` *file module args* [Function]
Report the failure to load FILE, a user-provided Scheme file, without exiting. ARGS is the list of arguments received by the 'throw' handler.

`show-version-and-exit` [*command*] [Function]
Display version information for COMMAND and '(exit 0)'.

`show-bug-report-information` [Function]

`make-regexp*` *regexp . flags* [Function]
Like 'make-regexp' but error out if REGEXP is invalid, reporting the error nicely.

`string->number*` *str* [Function]
Like 'string->number', but error out with an error message on failure.

`size->number` *str* [Function]
Convert STR, a storage measurement representation such as `"1024"` or `"1MiB"`, to a number of bytes. Raise an error if STR could not be interpreted.

`show-derivation-outputs` *derivation* [Function]
Show the output file names of DERIVATION, which can be a derivation or a derivation input.

`build-notifier` [*#:dry-run?*] [*#:use-substitutes?*] [*#:verbosity*] [Function]
Return a procedure suitable for 'with-build-handler' that, when 'build-things' is called, invokes 'show-what-to-build' to display the build plan. When DRY-RUN? is true, the 'with-build-handler' form returns without any build happening.

`show-what-to-build` *store drv* [*#:dry-run?*] [*#:use-substitutes?*] [Function]
[*#:verbosity*] [*#:mode*]
Show what will or would (depending on DRY-RUN?) be built in realizing the derivations listed in DRV using MODE, a 'build-mode' value. The elements of DRV can be either derivations or derivation inputs.

Return two values: a Boolean indicating whether there's something to build, and a Boolean indicating whether there's something to download.

When USE-SUBSTITUTES?, check and report what is prerequisites are available for download. VERBOSITY is an integer indicating the level of details to be shown: level 2 and higher provide all the details, level 1 shows a high-level summary, and level 0 shows nothing.

`show-what-to-build*` . *args* [Function]

Show what will or would (depending on DRY-RUN?) be built in realizing the derivations listed in DRV using MODE, a 'build-mode' value. The elements of DRV can be either derivations or derivation inputs.

Return two values: a Boolean indicating whether there's something to build, and a Boolean indicating whether there's something to download.

When USE-SUBSTITUTES?, check and report what is prerequisites are available for download. VERBOSITY is an integer indicating the level of details to be shown: level 2 and higher provide all the details, level 1 shows a high-level summary, and level 0 shows nothing.

`show-manifest-transaction` *store manifest transaction* [*#:dry-run?*] [Function]

Display what will/would be installed/removed from MANIFEST by TRANSACTION.

`guard*` (*var clauses ...*) *exp ...* [Special Form]

This variant of SRFI-34 'guard' does not unwind the stack before evaluating the tests and bodies of CLAUSES.

`call-with-error-handling` *thunk* [Function]

Call THUNK within a user-friendly error handler.

`with-error-handling` *body ...* [Special Form]

Run BODY within a user-friendly error condition handler.

`with-unbound-variable-handling` *exp ...* [Special Form]

Capture 'unbound-variable' exceptions in the dynamic extent of EXP... and report them in a user-friendly way.

`leave-on-EPIPE` *exp ...* [Special Form]

Run EXP... in a context where EPIPE errors are caught and lead to 'exit' with successful exit code. This is useful when writing to the standard output may lead to EPIPE, because the standard output is piped through 'head' or similar.

`read/eval` *str* [Function]

Read and evaluate STR, raising an error if something goes wrong.

`read/eval-package-expression` *str* [Function]

Read and evaluate STR and return the package it refers to, or exit an error.

`check-available-space` *need* [*directory*] [Function]

Make sure at least NEED bytes are available in DIRECTORY. Otherwise emit a warning.

`indented-string` *str indent* [*#:initial-indent?*] [Function]
Return STR with each newline preceded by INDENT spaces. When INITIAL-INDENT? is true, the first line is also indented.

`fill-paragraph` *str width* [*column*] [Function]
Fill STR such that each line contains at most WIDTH characters, assuming that the first character is at COLUMN.

When STR contains a single line break surrounded by other characters, it is converted to a space; sequences of more than one line break are preserved.

`%text-width` [Variable]

`texi->plain-text` *str* [Function]
Return a plain-text representation of texinfo fragment STR.

`package-description-string` *package* [Function]
Return a plain-text representation of PACKAGE description field.

`package-synopsis-string` *package* [Function]
Return a plain-text representation of PACKAGE synopsis field.

`string->recutils` *str* [Function]
Return a version of STR where newlines have been replaced by newlines followed by `"+ "`, which makes for a valid multi-line field value in the 'recutils' syntax.

`package->recutils` *p port* [*width*] [*#:hyperlinks?*] [*#:extra-fields*] [*#:highlighting*] [Function]
Write to PORT a 'recutils' record of package P, arranging to fit within WIDTH columns. EXTRA-FIELDS is a list of symbol/value pairs to emit. When HYPERLINKS? is true, emit hyperlink escape sequences when appropriate. Pass the synopsis and description through HIGHLIGHTING, a one-argument procedure that may return a colorized version of its argument.

`package-specification->name+version+output` *spec* [*output*] [Function]
`Parse package specification SPEC and return three value: the specified package name, version number (or #f), and output name (or OUTPUT). SPEC may optionally contain a version number and an output name, as in these examples:`

```
guile
guile@@2.0.9
guile:debug
guile@@2.0.9:debug
```

`pager-wrapped-port` [*port*] [Function]
If PORT is a pipe to a pager created by 'with-paginated-output-port', return the underlying port. Otherwise return #f.

`with-paginated-output-port` *port exp ... #:less-options opts* [Special Form]
`with-paginated-output-port` *port exp ...* [Special Form]
Evaluate EXP... with PORT bound to a port that talks to the pager if standard output is a tty, or with PORT set to the current output port.

`relevance` *obj regexps metrics* [Function]
Compute a "relevance score" for OBJ as a function of its number of matches of REGEXPS and accordingly to METRICS. METRICS is list of field/weight pairs, where FIELD is a procedure that returns a string or list of strings describing OBJ, and WEIGHT is a positive integer denoting the weight of this field in the final score.

A score of zero means that OBJ does not match any of REGEXPS. The higher the score, the more relevant OBJ is to REGEXPS.

`package-relevance` *package regexps* [Function]
Return a score denoting the relevance of PACKAGE for REGEXPS. A score of zero means that PACKAGE does not match any of REGEXPS.

`display-search-results` *matches port* [*#:regexps*] [*#:command*] [*#:print*] [Function]
Display MATCHES, a list of object/score pairs, by calling PRINT on each of them. If PORT is a terminal, print at most a full screen of results. REGEXPS is a list of regexps to highlight in search results.

`with-profile-lock` *profile exp ...* [Special Form]
Grab PROFILE's lock and evaluate EXP... Call 'leave' if the lock is already taken.

`string->generations` *str* [Function]
Return the list of generations matching a pattern in STR. This function accepts the following patterns: "1", "1,2,3", "1..9", "1..", "..9".

`string->duration` *str* [Function]
Return the duration matching a pattern in STR. This function accepts the following patterns: "1d", "1w", "1m".

`matching-generations` *str profile* [*#:duration-relation*] [Function]
Return the list of available generations matching a pattern in STR. See 'string->generations' and 'string->duration' for the list of valid patterns. When STR is a duration pattern, return all the generations whose ctime has DURATION-RELATION with the current time.

`display-generation` *profile number* [Function]
Display a one-line summary of generation NUMBER of PROFILE.

`display-profile-content` *profile number* [Function]
Display the packages in PROFILE, generation NUMBER, in a human-readable way.

`display-profile-content-diff` *profile gen1 gen2* [Function]
Display the changed packages in PROFILE GEN2 compared to generation GEN1.

`roll-back*` *store profile* [Function]
Like 'roll-back', but display what is happening.

`switch-to-generation*` *profile number* [Function]
Like 'switch-to-generation', but display what is happening.

`delete-generation*` *store profile generation* [Function]
Like 'delete-generation', but display what is going on.

`%default-message-language` [Variable]

`current-message-language` [Function]
Return the language used for messages according to the current locale. Return %DEFAULT-MESSAGE-LANGUAGE if that information could not be obtained. The result is an ISO-639-2 language code such as `"ar"`, without the territory part.

`run-guix-command` *command . args* [Function]
Run COMMAND with the given ARGS. Report an error when COMMAND is not found.

`run-guix` *. args* [Function]
Run the 'guix' command defined by command line ARGS. Unlike 'guix-main', this procedure assumes that locale, i18n support, and signal handling have already been set up.

`guix-main` *arg0 . args* [Function]

`guix-warning-port` [Variable]

`program-name` [Variable]

`info` [Special Form]

`leave` *args ...* [Special Form]
Emit an error message and exit.

`report-error` [Special Form]

`warning` [Special Form]

`G_` *t-13c8accbfea35c4d-1d* [Function]

`N_` *t-13c8accbfea35c4d-24 t-13c8accbfea35c4d-25 t-13c8accbfea35c4d-26* [Function]

`P_` *msgid* [Function]
Return the translation of the package description or synopsis MSGID.

`location->string` *loc* [Function]
Return a human-friendly, GNU-standard representation of LOC.

91 (guix upstream)

91.1 Overview

`This module provides tools to represent and manipulate a upstream source code, and to auto-update package recipes.`

91.2 Usage

`upstream-source` [Special Form]

`upstream-source?` [Special Form]

`upstream-source-package` [Special Form]

`upstream-source-version` [Special Form]

`upstream-source-urls` [Special Form]

`upstream-source-signature-urls` [Special Form]

`upstream-source-archive-types` *release* [Function]
Return the available types of archives for RELEASE—a list of strings such as `"gz"` or `"xz"`.

`upstream-source-input-changes` [Special Form]

`url-predicate` *matching-url?* [Function]
Return a predicate that returns true when passed a package whose source is an <origin> with the URL-FETCH method, and one of its URLs passes MATCHING-URL?.

`url-prefix-predicate` *prefix* [Function]
Return a predicate that returns true when passed a package where one of its source URLs starts with PREFIX.

`coalesce-sources` *sources* [Function]
Coalesce the elements of SOURCES, a list of <upstream-source>, that correspond to the same version.

`upstream-updater` [Special Form]

`upstream-updater?` [Special Form]

`upstream-updater-name` [Special Form]

`upstream-updater-description` [Special Form]

`upstream-updater-predicate` [Special Form]

`upstream-updater-latest` [Special Form]

`upstream-input-change?` [Special Form]

`upstream-input-change-name` [Special Form]

`upstream-input-change-type` [Special Form]

`upstream-input-change-action` [Special Form]

`changed-inputs` *package package-sexp* [Function]
Return a list of input changes for PACKAGE based on the newly imported S-expression PACKAGE-SEXP.

`%updaters` [Variable]

`lookup-updater` *package* [*updaters*] [Function]
Return an updater among UPDATERS that matches PACKAGE, or #f if none of them matches.

`download-tarball` *store url signature-url* [*#:key-download*] [Function]
Download the tarball at URL to the store; check its OpenPGP signature at SIGNATURE-URL, unless SIGNATURE-URL is false. On success, return the tarball file name; return #f on failure (network failure or authentication failure). KEY-DOWNLOAD specifies a download policy for missing OpenPGP keys; allowed values: 'interactive' (default), 'always', and 'never'.

`package-archive-type` *package* [Function]
If PACKAGE's source is a tarball or zip archive, return its archive type–a string such as `"xz"`. Otherwise return #f.

`package-latest-release` *package* [*updaters*] [Function]
Return an upstream source to update PACKAGE, a `<package>` object, or #f if none of UPDATERS matches PACKAGE. When several updaters match PACKAGE, try them until one of them returns an upstream source. It is the caller's responsibility to ensure that the returned source is newer than the current one.

`package-latest-release*` *package* [*updaters*] [Function]
Like 'package-latest-release', but ensure that the return source is newer than that of PACKAGE.

`package-update` *store package* [*updaters*] [*#:key-download*] [Function]
Return the new version, the file name of the new version tarball, and input changes for PACKAGE; return #f (three values) when PACKAGE is up-to-date; raise an error when the updater could not determine available releases. KEY-DOWNLOAD specifies a download policy for missing OpenPGP keys; allowed values: 'always', 'never', and 'interactive' (default).

`update-package-source` *package source hash* [Function]
Modify the source file that defines PACKAGE to refer to SOURCE, an `<upstream-source>` whose tarball has SHA256 HASH (a bytevector). Return the new version string if an update was made, and #f otherwise.

92 (guix utils)

92.1 Overview

92.2 Usage

`strip-keyword-arguments` *keywords args* [Function]
Remove all of the keyword arguments listed in KEYWORDS from ARGS.

`default-keyword-arguments` *args defaults* [Function]
Return ARGS augmented with any keyword/value from DEFAULTS for keywords not already present in ARGS.

`substitute-keyword-arguments` *original-args* ((*kw var dflt ...*) *exp*) ... [Special Form]
Return a new list of arguments where the value for keyword arg KW is replaced by EXP. EXP is evaluated in a context where VAR is bound to the previous value of the keyword argument, or DFLT if given.

`ensure-keyword-arguments` *args kw/values* [Function]
`Force the keywords arguments KW/VALUES in the keyword argument list ARGS.`
`For instance:`

```
(ensure-keyword-arguments '(#:foo 2) '(#:foo 2))
=> (#:foo 2)

(ensure-keyword-arguments '(#:foo 2) '(#:bar 3))
=> (#:foo 2 #:bar 3)

(ensure-keyword-arguments '(#:foo 2) '(#:bar 3 #:foo 42))
=> (#:foo 42 #:bar 3)
```

`%guix-source-root-directory` [Function]
Return the source root directory of the Guix found in %load-path.

`current-source-directory` [Special Form]
Return the absolute name of the current directory, or #f if it could not be determined.

`nix-system->gnu-triplet` [*system*] [*vendor*] [Function]
Return a guess of the GNU triplet corresponding to Nix system identifier SYSTEM.

`gnu-triplet->nix-system` *triplet* [Function]
Return the Nix system type corresponding to TRIPLET, a GNU triplet as returned by 'config.guess'.

`%current-system` [Variable]

`%current-target-system` [Variable]

`package-name->name+version` *spec* [*delimiter*] [Function]

Given SPEC, a package name like "foo@@0.9.1b", return two values: "foo" and "0.9.1b". When the version part is unavailable, SPEC and #f are returned. Both parts must not contain any '@@'. Optionally, DELIMITER can be a character other than '@@'.

`target-linux?` [*target*] [Function]
Does the operating system of TARGET use the Linux kernel?

`target-hurd?` [*target*] [Function]
Does TARGET represent the GNU(/Hurd) system?

`target-mingw?` [*target*] [Function]
Is the operating system of TARGET Windows?

`target-x86-32?` [*target*] [Function]
Is the architecture of TARGET a variant of Intel's 32-bit architecture (IA32)?

`target-x86-64?` [*target*] [Function]
Is the architecture of TARGET a variant of Intel/AMD's 64-bit architecture (x86_64)?

`target-x86?` [*target*] [Function]

`target-arm32?` [*target*] [Function]

`target-aarch64?` [*target*] [Function]

`target-arm?` [*target*] [Function]

`target-ppc32?` [*target*] [Function]

`target-ppc64le?` [*target*] [Function]

`target-powerpc?` [*target*] [Function]

`target-riscv64?` [*target*] [Function]
Is the architecture of TARGET a 'riscv64' machine?

`target-mips64el?` [*target*] [Function]

`target-64bit?` [*system*] [Function]

`ar-for-target` [*target*] [Function]

`as-for-target` [*target*] [Function]

`cc-for-target` [*target*] [Function]

`cxx-for-target` [*target*] [Function]

`ld-for-target` [*target*] [Function]

`pkg-config-for-target` [*target*] [Function]

`version-compare` *a b* [Function]
Return '> when A denotes a newer version than B, '< when A denotes a older version than B, or '= when they denote equal versions.

`version>?` *a b* [Function]
Return #t when A denotes a version strictly newer than B.

`version>=?` *a b* [Function]
Return #t when A denotes a version newer or equal to B.

`version-prefix` *version-string num-parts* [Function]
Truncate version-string to the first num-parts components of the version. For example, (version-prefix "2.1.47.4.23" 3) returns "2.1.47"

`version-major+minor+point` *version-string* [Function]
Return "major>.<minor>.<point>", where major, minor and point are the major, minor and point version numbers from the version-string. For example, (version-major+minor+point "6.4.5.2") returns "6.4.5" or (version-major+minor+point "1.19.2-2581-324ca14c3003") returns "1.19.2".

`version-major+minor` *version-string* [Function]
Return "<major>.<minor>", where major and minor are the major and minor version numbers from version-string.

`version-major` *version-string* [Function]
Return the major version number as string from the version-string.

`version-unique-prefix` *version versions* [Function]
Return the shortest version prefix to unambiguously identify VERSION among VERSIONS. For example:

```
(version-unique-prefix "2.0" '("3.0" "2.0"))
=> "2"

(version-unique-prefix "2.2" '("3.0.5" "2.0.9" "2.2.7"))
=> "2.2"

(version-unique-prefix "27.1" '("27.1"))
=> ""
```

`guile-version>?` *str* [Function]
Return #t if the running Guile version is greater than STR.

`version-prefix?` *v1 v2* [Function]
Return true if V1 is a version prefix of V2:

```
(version-prefix? "4.1" "4.16.2") => #f
(version-prefix? "4.1" "4.1.2") => #t
```

`string-replace-substring` *str substr replacement* [*start*] [*end*] [Function]
Replace all occurrences of SUBSTR in the START–END range of STR by REPLACEMENT.

`file-extension` *file* [Function]
Return the extension of FILE or #f if there is none.

`file-sans-extension` *file* [Function]
Return the substring of FILE without its extension, if any.

`tarball-sans-extension` *tarball* [Function]
Return TARBALL without its .tar.* or .zip extension.

`compressed-file?` *file* [Function]
Return true if FILE denotes a compressed file.

`switch-symlinks` *link target* [Function]
Atomically switch LINK, a symbolic link, to point to TARGET. Works both when LINK already exists and when it does not.

`call-with-temporary-directory` *proc* [Function]
Call PROC with a name of a temporary directory; close the directory and delete it when leaving the dynamic extent of this call.

`with-atomic-file-output` *file proc* [Function]
Call PROC with an output port for the file that is going to replace FILE. Upon success, FILE is atomically replaced by what has been written to the output port, and PROC's result is returned.

`with-environment-variables` *variables exp ...* [Special Form]
Evaluate EXP with the given environment VARIABLES set.

`arguments-from-environment-variable` *variable* [Function]
Retrieve value of environment variable denoted by string VARIABLE in the form of a list of strings ('char-set:graphic' tokens) suitable for consumption by 'args-fold', if VARIABLE is defined, otherwise return an empty list.

`config-directory` . *t-1bee376549df88c2-2bb7* [Function]

`cache-directory` . *t-1bee376549df88c2-2bbf* [Function]

`readlink*` *file* [Function]
Call 'readlink' until the result is not a symlink.

`go-to-location` *port line column* [Function]
Jump to LINE and COLUMN (both one-indexed) in PORT. Maintain a source location map such that this can boil down to seek(2) and a few read(2) calls, which can drastically speed up repetitive operations on large files.

`edit-expression` *source-properties proc* [*#:encoding*] [Function]
[*#:include-trailing-newline?*]
Edit the expression specified by SOURCE-PROPERTIES using PROC, which should be a procedure that takes the original expression in string and returns a new one. ENCODING will be used to interpret all port I/O, it defaults to UTF-8. This procedure returns #t on success. When INCLUDE-TRAILING-NEWLINE? is true, the trailing line is included in the edited expression.

`delete-expression` *source-properties* [Function]
Delete the expression specified by SOURCE-PROPERTIES.

`filtered-port` *command input* [Function]
Return an input port where data drained from INPUT is filtered through COMMAND (a list). In addition, return a list of PIDs that the caller must wait. When INPUT is a file port, it must be unbuffered; otherwise, any buffered data is lost.

`decompressed-port` *compression input* [Function]
Return an input port where INPUT is decompressed according to COMPRESSION, a symbol such as 'xz.

`call-with-decompressed-port` *compression port proc* [Function]
Call PROC with a wrapper around PORT, a file port, that decompresses data read from PORT according to COMPRESSION, a symbol such as 'xz.

`compressed-output-port` *compression output* [*#:options*] [Function]
Return an output port whose input is compressed according to COMPRESSION, a symbol such as 'xz, and then written to OUTPUT. In addition return a list of PIDs to wait for. OPTIONS is a list of strings passed to the compression program–e.g., '("–fast").

`call-with-compressed-output-port` *compression port proc* [Function]
[*#:options*]
Call PROC with a wrapper around PORT, a file port, that compresses data that goes to PORT according to COMPRESSION, a symbol such as 'xz. OPTIONS is a list of command-line arguments passed to the compression program.

`canonical-newline-port` *port* [Function]
```
Return an input port that wraps PORT such that all newlines consist
  of a single linefeed.
```

`string-distance` *s1 s2* [Function]
Compute the Levenshtein distance between two strings.

`string-closest` *trial tests* [*#:threshold*] [Function]
Return the string from TESTS that is the closest from the TRIAL, according to 'string-distance'. If the TESTS are too far from TRIAL, according to THRESHOLD, then #f is returned.

`pretty-print-table` *rows* [*#:max-column-width*] [*#:left-pad*] [Function]
Print ROWS in neat columns. All rows should be lists of strings and each row should have the same length. The columns are separated by a tab character, and aligned using spaces. The maximum width of each column is bound by MAX-COLUMN-WIDTH. Each row is prefixed with LEFT-PAD spaces.

`&error-location` [Variable]

`&fix-hint` [Variable]

`<location>` [Variable]

`location-column` [Special Form]

`location-file` [Special Form]

`location-line` [Special Form]

`location?` [Special Form]

`call-with-temporary-output-file` *proc* [Function]
Call PROC with a name of a temporary file and open output port to that file; close the file and delete it when leaving the dynamic extent of this call.

`condition-fix-hint` *obj* [Function]

`error-location` *obj* [Function]

`error-location?` *obj* [Function]

`fix-hint?` *obj* [Function]

`location` *file line column* [Function]
Return the <location> object for the given FILE, LINE, and COLUMN.

`location->source-properties` *loc* [Function]
Return the source property association list based on the info in LOC, a location object.

`source-properties->location` *loc* [Function]
Return a location object based on the info in LOC, an alist as returned by Guile's 'source-properties', 'frame-source', 'current-source-location', etc.

93 (guix workers)

93.1 Overview

```
This module implements "worker pools".  Worker pools are the low-level
mechanism that's behind futures: there's a fixed set of threads
("workers") that one can submit work to, and one of them will eventually
pick the submitted tasks.

Unlike futures, these worker pools are meant to be used for tasks that
have a side-effect.  Thus, we never "touch" a task that was submitted like
we "touch" a future.  Instead, we simply assume that the task will
eventually complete.
```

93.2 Usage

`pool?` [Special Form]

`make-pool` [*count*] [*#:thread-name*] [Function]
Return a pool of COUNT workers. Use THREAD-NAME as the name of these threads as reported by the operating system.

`pool-enqueue!` *pool thunk* [Function]
Enqueue THUNK for future execution by POOL.

`pool-idle?` *pool* [Function]
Return true if POOL doesn't have any task in its queue and all the workers are currently idle (i.e., waiting for a task).

`eventually` *pool exp ...* [Special Form]
Run EXP eventually on one of the workers of POOL.