Trucler
A Common Lisp environment protocol
and its implementation.

Robert Strandh

2019

Contents

Chapter 1

Introduction

In section 8.5 of the second edition of the book “Common Lisp, the Language” (also known as CLtL2) by Guy Steele [Ste90], a protocol for accessing compile-time environments is defined. That protocol has two main problems:

1. It is incomplete. It does not provide for a way to query the environment for description about blocks or tags.
2. It is not extensible. In order for an implementation to make one of the query functions return more information, additional return values would have to be defined. However, such a change is not considered backward compatible, so this kind of extension is not recommended.

Trucler introduces a protocol that solves these problems as follows:

1. It contains additional query and augmentation functions for blocks and tags.
2. Instead of returning multiple values, the query functions return standard objects. Accessors specialized to the classes of those objects provide the information that the protocol in CLtL2 provides as multiple values.

In addition to providing a mechanism that solves the problems of protocol presented in CLtL2, we also add several new features that a language processor must use to process source code.

Chapter 2

Querying the environment

In this chapter, we describe classes and functions that are used by the language processor to query the environment concerning information about program elements that the language processor needs in order to determine how to process those program elements.

When the language processor calls a generic query function, it passes two or three arguments, depending on the function it calls. The first argument is called the `client`. Trucler does not specialize on this argument, but client code should define a standard class and pass an instance of that class for this argument. Client code can then define auxiliary methods that specialize to this class on the query generic functions. The second argument is the environment concerned by the query. Client code must supply methods on these functions, specialized to its particular representation of its global environments. If the client does not have an explicit representation of its global environment (as is usually the case), it must nevertheless define a dummy standard class to specialize on. Contrary to global environments, Trucler provides its own representation of *lexical* environments, and it provides methods on the query functions, specialized to the classes defined to represent those lexical environments. Client code that wants to use a different representation of lexical environments than the one provided by Trucler must also provide methods specialized to its lexical environment classes.

The primary methods on the query functions should return instances of the classes described in this chapter. Any such instance contains all available information about some program element in that particular environment. This information must typically be assembled from different parts of the environment. For that reason, client code typically creates a new instance whenever a query function is called, rather than attempting to store such instances in the environment. If any of these client-supplied methods fails to accomplish its task, it should return `nil`.

Client code is free to define subclasses of the classes described here, for instance in order to represent implementation-specific information about the program elements. Client code would then typically also provide auxiliary methods or overriding primary methods on the compilation functions that handle these classes.

2.1 Query functions

2.1.1 Variable information

⇒ `describe-variable` *client environment name* [*Generic Function*]

This function is called by the language processor whenever a symbol in a *variable* position is to be compiled. If successful, it returns an instance of one of the classes described in Section 2.4.1.

⇒ `no-variable-description` [*Condition*]

This condition is signaled by Trucler when a client-supplied method on the generic function `describe-variable` returns `nil`.

⇒ `name` *(condition* `no-variable-description`*)* [*Method*]

This method returns the name of the variable for which no description was available.

2.1.2 Function information

⇒ `describe-function` *client environment name* [*Generic Function*]

This function is called by the language processor whenever a symbol in a *func-*

tion position is to be compiled or whenever a function name is found in a context where it is known to refer to a function. If successful, it returns an instance of one of the classes described in Section 2.4.2.

⇒ `no-function-description` [*Condition*]

This condition is signaled by Trucler when a client-supplied method on the generic function `describe-function` returns `nil`.

⇒ `name` *(condition* `no-function-description`*)* [*Method*]

This method returns the name of the function for which no description was available.

2.1.3 Block information

⇒ `describe-block` *client environment name* [*Generic Function*]

This function is called by the language processor whenever a symbol referring to a *block* is found, typically in a `return-from` form. If successful, it returns an instance of the class described in Section 2.4.3.

⇒ `no-block-description` [*Condition*]

This condition is signaled by Trucler when a client-supplied method on the generic function `describe-block` returns `nil`.

⇒ `name` *(condition* `no-block-description`*)* [*Method*]

This method returns the name of the block for which no description was available.

2.1.4 Tag information

⇒ `describe-tag` *client environment tag* [*Generic Function*]

This function is called by the language processor whenever a symbol or an integer referring to a *tag* is found, typically in a `go` form. If successful, it returns an instance of the class described in Section 2.4.4.

⇒ `no-tag-description` [*Condition*]

This condition is signaled by Trucler when a client-supplied method on the generic function `describe-tag` returns `nil`.

⇒ `name` *(condition* `no-tag-description`*)* [*Method*]

This method returns the name of the tag for which no description was available.

2.1.5 Class information

⇒ `describe-class` *client environment class-name* [*Generic Function*]

This function is called by the language processor whenever a symbol referring to a *class* is found, for example as a specializer in a `defmethod` form. If successful, it returns an instance of the class described in Section 2.4.5.

⇒ `no-class-description` [*Condition*]

This condition is signaled by Trucler when a client-supplied method on the generic function `describe-class` returns `nil`.

⇒ `name` *(condition* `no-class-description`*)* [*Method*]

This method returns the name of the class for which no description was available.

2.1.6 Optimize information

⇒ `describe-optimize` *client environment* [*Generic Function*]

Client-supplied methods on this function must always return a valid instance of the class `optimize-description`. It returns an instance of the class described in Section 2.4.6.

2.2 Mixin classes

For maximum flexibility, each query class is the subclass of one or more mixin classes, each one providing one single feature. That feature is represented as a slot with an initarg, a reader, an initform, and a type.

2.2.1 `name-mixin`

⇒ `name-mixin` [*Class*]

This class is a superclass of query classes that require a name to identify the information supplied by the class instances.

⇒ `:name` [*Initarg*]

⇒ `name` *(description* `name-mixin`*)* [*Method*]

Given an instance of the class `name-mixin`, this method returns the name information, as supplied by the initarg `:name`.

2.2.2 `identity-mixin`

⇒ `identity-mixin` [*Class*]

This class is a superclass of query classes that require some kind of identity to distinguish instances of the query class that have the same name.

⇒ `:identity` [*Initarg*]

⇒ `identity` *(description* `identity-mixin`*)* [*Method*]

Given an instance of the class `identity-mixin`, this method returns the identity information, as supplied by the initarg `:idenity`.

2.2.3 `type-mixin`

⇒ `type-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have a type.

⇒ `:type` [*Initarg*]

If this initarg is not supplied, it defaults to `t`.

⇒ `type` *(description* `type-mixin`*)* [*Method*]

Given an instance of the class `type-mixin`, this method returns the type information, as supplied by the initarg `:type`.

2.2.4 `ignore-mixin`

⇒ `ignore-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can be declared `ignore` or `ignorable`.

⇒ `:ignore` [*Initarg*]

The value of this initarg must be one of the symbols `ignore` and `ignorable` from the `common-lisp` package.

⇒ `ignore` *(description* `ignore-mixin`*)* [*Method*]

Given an instance of the class `ignore-mixin`, this method returns the ignore information, as supplied by the initarg `:ignore`.

2.2.5 `dynamic-extent-mixin`

⇒ `dynamic-extent-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can be declared `dynamic-extent`.

⇒ `:dynamic-extent` [*Initarg*]

⇒ `dynamic-extent` *(description* `dynamic-extent-mixin`*)* [*Method*]

Given an instance of the class `dynamic-extent-mixin`, this method returns the dynamic-extent information, as supplied by the initarg `:dynamic-extent`.

2.2.6 `expansion-mixin`

⇒ `expansion-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have an expansion. In particular, it is a superclass of the abstract class `symbol-macro-description`.

⇒ `:expansion` [*Initarg*]

⇒ `expansion` *(description* `expansion-mixin`*)* [*Method*]

Given an instance of the class `expansion-mixin`, this method returns the expansion information, as supplied by the initarg `:expansion`.

2.2.7 `expander-mixin`

⇒ `expander-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have an expander function. In particular, it is a superclass of the abstract class `macro-description`.

⇒ `:expander` [*Initarg*]

⇒ `expander` *(description* `expander-mixin`*)* [*Method*]

Given an instance of the class `expander-mixin`, this method returns the expander information, as supplied by the initarg `:expander`.

2.2.8 `class-name-mixin`

⇒ `class-name-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have a class-name. In particular, it is a superclass of the class `global-function-description`.

⇒ `:class-name` [*Initarg*]

⇒ `class-name` *(description* `class-name-mixin`*)* [*Method*]

Given an instance of the class `class-name-mixin`, this method returns the class-name information, as supplied by the initarg `:class-name`.

2.2.9 `inline-mixin`

⇒ `inline-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have inline information. In particular, it is a superclass of the class `authentic-function-description`.

⇒ `:inline` [*Initarg*]

Possible values for this initarg are `nil`, `inline`, and `notinline`, all symbols in the `common-lisp` package. The value `nil` means that no inline information has been provided, and this is the default value if the initarg is omitted.

⇒ `inline` *(description* `inline-mixin`*)* [*Method*]

Given an instance of the class `inline-mixin`, this method returns the inline information, as supplied by the initarg `:inline`.

2.2.10 `method-class-name-mixin`

⇒ `method-class-name-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have method-class-name information. In particular, it is a superclass of the class `generic-function-description`.

⇒ `:method-class-name` [*Initarg*]

The value of this initarg is a symbol naming a class to be used for methods. If this initarg is not given, it defaults to the symbol `standard-method`.

⇒ `method-class-name` *(description* `method-class-name-mixin`*)* [*Method*]

Given an instance of the class `method-class-name-mixin`, this method returns the method-class-name information, as supplied by the initarg `:method-class-name`.

2.2.11 `speed-mixin`

⇒ `speed-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have speed information. In particular, it is a superclass of the class `optimize-description`.

⇒ `:speed` [*Initarg*]

The value of this initarg must be an integer between 0 and 3 inclusive.

⇒ `speed` *(description* `speed-mixin`*)* [*Method*]

Given an instance of the class `speed-mixin`, this method returns the compilation-speed information, as supplied by the initarg `:speed`.

2.2.12 `compilation-speed-mixin`

⇒ `compilation-speed-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have compilation-speed information. In particular, it is a superclass of the class `optimize-description`.

⇒ `:compilation-speed` [*Initarg*]

The value of this initarg must be an integer between 0 and 3 inclusive.

⇒ `compilation-speed` *(description* `compilation-speed-mixin`*)* [*Method*]

Given an instance of the class `compilation-speed-mixin`, this method returns the compilation-speed information, as supplied by the initarg `:compilation-speed`.

2.2.13 `debug-mixin`

⇒ `debug-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have debug information. In particular, it is a superclass of the class `optimize-description`.

⇒ `:debug` [*Initarg*]

The value of this initarg must be an integer between 0 and 3 inclusive.

⇒ `debug` *(description* `debug-mixin`*)* [*Method*]

Given an instance of the class `debug-mixin`, this method returns the debug information, as supplied by the initarg `:debug`.

2.2.14 `space-mixin`

⇒ `space-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have space information. In particular, it is a superclass of the class `optimize-description`.

⇒ `:space` [*Initarg*]

The value of this initarg must be an integer between 0 and 3 inclusive.

⇒ `space` *(description* `space-mixin`*)* [*Method*]

Given an instance of the class `space-mixin`, this method returns the space information, as supplied by the initarg `:space`.

2.2.15 `safety-mixin`

⇒ `safety-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have safety information. In particular, it is a superclass of the class `optimize-description`.

⇒ `:safety` [*Initarg*]

The value of this initarg must be an integer between 0 and 3 inclusive.

⇒ `safety` *(description* `safety-mixin`*)* [*Method*]

Given an instance of the class `safety-mixin`, this method returns the safety information, as supplied by the initarg `:safety`.

2.2.16 `superclass-names-mixin`

⇒ `superclass-names-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have superclass-names information. In particular, it is a superclass of the class `class-description`.

⇒ `:superclass-names` [*Initarg*]

The value of this initarg is a list of symbols naming a classes. If this initarg is not given, it defaults the empty list. Only explicitly mentioned superclass names should be provided.

⇒ `superclass-names` *(description* `superclass-names-mixin`*)* [*Method*]

Given an instance of the class `superclass-names-mixin`, this method returns the superclass-names information, as supplied by the initarg `:superclass-names`.

2.2.17 `metaclass-name-mixin`

⇒ `metaclass-name-mixin` [*Class*]

This class is a superclass of query classes that provide information about entities that can have metaclass-name information. In particular, it is a superclass of the class `class-description`.

⇒ `:metaclass-name` [*Initarg*]

The value of this initarg is a symbol naming a class to be used as a metaclasss. If this initarg is not given, it defaults to the symbol `standard-class`.

⇒ `metaclass-name` *(description* `metaclass-name-mixin`*)* [*Method*]

Given an instance of the class `metaclass-name-mixin`, this method returns the metaclass-name information, as supplied by the initarg `:metaclass-name`.

2.3 Abstract query classes

⇒ `variable-description` [*Class*]

This abstract class is the superclass of every query class returned by a call to the generic function `describe-variable`. It is a subclass of the class `name-mixin`.

⇒ `authentic-variable-description` [*Class*]

This abstract class is a subclass of the classes `variable-description` and `type-mixin`.

It is a superclass of the two instantiable classes `lexical-variable-description` and `special-variable-description`.

⇒ `symbol-macro-description` [*Class*]

This abstract class is a subclass of the classes `variable-description`, `type-mixin`, and `expansion-mixin`.

It is a superclass of the two instantiable classes `local-symbol-macro-description` and `global-symbol-macro-description`.

⇒ `function-description` [*Class*]

This abstract class is the superclass of every query class returned by a call to the generic function `function-description`. It is a subclass of the class `name-mixin`.

⇒ `authentic-function-description` [*Class*]

This abstract class is a subclass of the classes `function-description` and `type-mixin`.

It is a superclass of the two instantiable classes `local-function-description` and `global-function-description`.

⇒ `macro-description` [*Class*]

This abstract class is a subclass of the classes `function-description` and `expander-mixin`.

It is a superclass of the two instantiable classes `local-macro-description` and `global-macro-description`.

2.4 Instantiable classes

2.4.1 Variable description

⇒ `lexical-variable-description` [*Class*]

This class represents information about lexical variables. An instance of this class is returned by a call to `variable-description` when it turns out that the symbol passed as an argument refers to a lexical variable.

This class is a subclass of the classes `authentic-variable-description` `identity-mixin`, `ignore-mixin`, and `dynamic-extent-mixin`.

⇒ `special-variable-description` [*Class*]

This class represents information about special variables. An instance of this class is returned by a call to `variable-description` when it turns out that the symbol passed as an argument refers to a special variable.

This class is a subclass of the classes `authentic-variable-description` and `global-p-mixin`.

⇒ `constant-variable-description` [*Class*]

This class represents information about constant variables. An instance of this class is returned by a call to `variable-description` when it turns out that the symbol passed as an argument refers to a constant variable.

This class is a subclass of the classes `variable-description` and `value-mixin`.

⇒ `global-symbol-macro-description` [*Class*]

This class is a subclass of `symbol-macro-description`. It is returned by a call to `variable-descriptionrmation` when the name is defined as a global symbol macro, as defined by `define-symbol-macro`.

⇒ `local-symbol-macro-description` [*Class*]

This class is a subclass of `symbol-macro-description` and `ignore-mixin`. It is returned by a call to `variable-descriptionrmation` when the name is defined as a local symbol macro, as defined by `symbol-macrolet`.

2.4.2 Function description

⇒ `local-function-description` [*Class*]

This class represents information about local functions introduced by `flet` or `labels`. An instance of this class is returned by a call to `function-description` when it turns out that the function name passed as an argument refers to a local function.

This class is a subclass of `authentic-function-description`, `identity-mixin`, `ignore-mixin`, and `dynamic-extent-mixin`.

⇒ `global-function-description` [*Class*]

This class represents information about global functions. An instance of this class is returned by a call to `function-description` when it turns out that the function name passed as an argument refers to a global function.

This class is a subclass of `authentic-function-description`, `compiler-macro-mixin`, and `class-name-mixin`.

⇒ `generic-function-description` [*Class*]

This class is a subclass of `global-function-description` and `method-class-name-mixin`.

⇒ `local-macro-description` [*Class*]

This class represents information about local macros introduced by `macrolet`. An instance of this class is returned by a call to `function-description` when it turns out that the function name passed as an argument refers to a local macro.

This class is a subclass of `macro-description` and `ignore-mixin`.

⇒ `global-macro-description` [*Class*]

This class represents information about global macros introduced by `macrolet`. An instance of this class is returned by a call to `function-description` when it turns out that the function name passed as an argument refers to a global macro.

This class is a subclass of `macro-description` and `compiler-macro-mixin`.

⇒ `special-operator-description` [*Class*]

This class represents information about special operators. An instance of this class is returned by a call to `function-description` when it turns out that the function name passed as an argument refers to a specialoperator.

This class is a subclass of the class `function-description`.

2.4.3 Block description

⇒ `block-description` [*Class*]

This class represents information about blocks introduced by `block`. An instance of this class is returned by a call to `block-description` when the symbol passed as an argument refers to a block.

This class is a subclass of the classes `name-mixin` and `identity-mixin`.

2.4.4 Tag description

⇒ `tag-description` [*Class*]

This class represents information about tags introduced by `tagbody`. An instance of this class is returned by a call to `tag-description` when the name (which must be a symbol or an integer) passed as an argument refers to a tag.

This class is a subclass of the classes `name-mixin` and `identity-mixin`.

2.4.5 Class description

⇒ `class-description` [*Class*]

This class represents information about a class introduced by `defclass`. An instance of this class is returned by a call to `class-description` when the name (which must be a symbol) passed as an argument refers to a class.

This class is a subclass of the classes `name-mixin`, `superclass-names-mixin`, and `metaclass-name-mixin`.

2.4.6 Optimize description

⇒ `optimize-description` [*Class*]

This class is a subclass of `speed-mixin`, `compilation-speed-mixin`, `debug-mixin`, `space-mixin`, and `safety-mixin`.

Chapter 3

Augmenting the environment

3.1 Creating new description

In order to create a new description, `make-instance` must be called, providing values for all the initialization arguments corresponding to features that do not have any initialization forms.

3.2 Low-level augmentation functions

In this section, we describe basic functions for augmenting an environment, given an old environment and a description.

⇒ `augment-with-variable-description` *client environment description* [*Generic Function*]

This function is used to create a new environment object from an existing environment object and an instance of the class `variable-description`.

⇒ `augment-with-variable-description`
client
(environment `environment`*)*
(description `variable-description`*)* [*Method*]

This default method returns a new environment object which is like the one passed as an argument, except that *description* will shadow any variable description with the same name.

⇒ `augment-with-function-description` *client environment function-description* [*Generic Function*]

This function is used to create a new environment object from an existing environment object and an instance of the class `function-description`.

⇒ `augment-with-function-description`
client
(environment `environment`*)*
(function-description `function-description`*)* [*Method*]

This default method returns a new environment object which is like the one passed as an argument, except that *function-description* will shadow any function description with the same name.

⇒ `augment-with-block-description` *client environment block-description* [*Generic Function*]

This function is used to create a new environment object from an existing environment object and an instance of the class `block-description`.

⇒ `augment-with-block-description`
client
(environment `environment`*)*
(block-description `block-description`*)* [*Method*]

This default method returns a new environment object which is like the one passed as an argument, except that *block-description* will shadow any block description with the same name.

⇒ `augment-with-tag-description` *client environment tag-description* [*Generic Function*]

This function is used to create a new environment object from an existing environment object and an instance of the class `tag-description`.

⇒ `augment-with-tag-description`
client
(environment `environment`*)*
(tag-description `tag-description`*)* [*Method*]

This default method returns a new environment object which is like the one passed as an argument, except that *tag-description* will shadow any tag description with the same name.

⇒ `augment-with-optimize-description`
client environment optimize-description [*Generic Function*]

This function is used to create a new environment object from an existing environment object and an instance of the class `optimize-description`.

⇒ `augment-with-optimize-description`
client
(environment `environment`*)*
(optimize-description `optimize-description`*)* [*Method*]

This default method returns a new environment object which is like the one passed as an argument, except that *optimize-description* will shadow any previous optimize description.

3.3 Merging descriptions

We use the term *merging* to mean the creation of a new description from an existing description plus some additional information such as type or dynamic extent.

In this section, we describe generic functions that are provided for this purpose.

⇒ `merge-type` *client description type* [*Generic Function*]

Given an instance of the class `description` and a type descriptor, return a new instance that is just like *description* (including the class and the values of all the slots), except that its type description has been updated according to that of *type*.

⇒ `invalid-description-for-merging-type-information` [*Condition*]

This condition is signaled by `merge-type` when the *description* argument is not an instance of a class that contains information about type.

⇒ `merge-type` *client description type* [*Method*]

This is the default method provided on `merge-type`. It signals the condition `invalid-description-for-merging-type-information`.

⇒ `merge-type`
client
(description `type-mixin`*)*

type [*Method*]

This is the main method provided on `merge-type` and it is specialized to `type-mixin`.

⇒ `merge-ignore` *client description ignore* [*Generic Function*]

Given an instance of the class `description` and one of the symbols `cl:ignore` and `cl:ignorable`, return a new instance that is just like *description* (including the class and the values of all the slots), except that its ignore information has been updated according to that of *ignore*.

⇒ `invalid-description-for-merging-ignore-information` [*Condition*]

This condition is signaled by `merge-ignore` when the *description* argument is not an instance of a class that contains information about ignore.

⇒ `merge-ignore` *client description ignore* [*Method*]

This is the default method provided on `merge-ignore`. It signals the condition `invalid-description-for-merging-ignore-information`.

⇒ `merge-ignore`
client
(description `ignore-mixin`*)*
ignore [*Method*]

This is the main method provided on `merge-ignore` and it is specialized to `ignore-mixin`.

⇒ `merge-dynamic-extent` *client description* [*Generic Function*]

Given an instance of the class `description`, return a new instance that is just like *description* (including the class and the values of all the slots), except that its dynamic-extent information has been updated so that it is *true*.

⇒ `invalid-description-for-merging-dynamic-extent-information` [*Condition*]

This condition is signaled by `merge-dynamic-extent` when the *description* argument is not an instance of a class that contains information about dynamic-extent.

⇒ `merge-dynamic-extent` *client description* [*Method*]

This is the default method provided on `merge-dynamic-extent`. It signals the condition `invalid-description-for-merging-dynamic-extent-information`.

⇒ `merge-dynamic-extent` *client (description* `dynamic-extent-mixin`*)* [*Method*]

This is the main method provided on `merge-dynamic-extent` and it is specialized to `dynamic-extent-mixin`.

⇒ `merge-inline` *client description inline* [*Generic Function*]

Given an instance of the class `description` and one of the symbols `cl:inline` and `cl:notinline`, return a new instance that is just like *description* (including the class and the values of all the slots), except that its inline information has been updated according to that of *inline*.

⇒ `invalid-description-for-merging-inline-information` [*Condition*]

This condition is signaled by `merge-inline` when the *description* argument is not an instance of a class that contains information about inline.

⇒ `merge-inline` *client description inline* [*Method*]

This is the default method provided on `merge-inline`. It signals the condition `invalid-description-for-merging-inline-information`.

⇒ `merge-inline`
client
(description `inline-mixin`*)*
inline [*Method*]

This is the main method provided on `merge-inline` and it is specialized to `inline-mixin`.

⇒ `merge-speed` *client description value* [*Generic Function*]

Given an instance of the class `description` and an integer between 0 and 3, return a new instance that is just like *description* (including the class and the values of all the slots), except that its speed information has been updated according to that of *value*.

⇒ `invalid-description-for-merging-speed-information` [*Condition*]

This condition is signaled by `merge-speed` when the *description* argument is not an instance of a class that contains information about speed.

⇒ `merge-speed` *client description speed* [*Method*]

This is the default method provided on `merge-speed`. It signals the condition `invalid-description-for-merging-speed-information`.

⇒ `merge-speed`
client
(description `speed-mixin`*)*
value [*Method*]

This is the main method provided on `merge-speed` and it is specialized to `speed-mixin`.

⇒ `merge-compilation-speed` *client description value* [*Generic Function*]

Given an instance of the class `description` and an integer between 0 and 3, return a new instance that is just like *description* (including the class and the values of all the slots), except that its compilation-speed information has been updated according to that of *value*.

⇒ `invalid-description-for-merging-compilation-speed-information` [*Condition*]

This condition is signaled by `merge-compilation-speed` when the *description* argument is not an instance of a class that contains information about compilation-speed.

⇒ `merge-compilation-speed` *client description compilation-speed* [*Method*]

This is the default method provided on `merge-compilation-speed`. It signals the `invalid-description-for-merging-compilation-speed-information` condition.

⇒ `merge-compilation-speed`
client
(description `compilation-speed-mixin`*)*
value [*Method*]

This is the main method provided on `merge-compilation-speed` and it is specialized to `compilation-speed-mixin`.

⇒ `merge-debug` *client description value* [*Generic Function*]

Given an instance of the class `description` and an integer between 0 and 3, return a new instance that is just like *description* (including the class and the values of all the slots), except that its debug information has been updated according to that of *value*.

⇒ `invalid-description-for-merging-debug-information` [*Condition*]

This condition is signaled by `merge-debug` when the *description* argument is not an instance of a class that contains information about debug.

⇒ `merge-debug` *client description debug* [*Method*]

This is the default method provided on `merge-debug`. It signals the condition `invalid-description-for-merging-debug-information`.

⇒ `merge-debug`
client
(description `debug-mixin`*)*
value [*Method*]

This is the main method provided on `merge-debug` and it is specialized to `debug-mixin`.

⇒ `merge-space` *client description value* [*Generic Function*]

Given an instance of the class `description` and an integer between 0 and 3, return a new instance that is just like *description* (including the class and the values of all the slots), except that its space information has been updated according to that of *value*.

⇒ `invalid-description-for-merging-space-information` [*Condition*]

This condition is signaled by `merge-space` when the *description* argument is not an instance of a class that contains information about space.

⇒ `merge-space` *client description space* [*Method*]

This is the default method provided on `merge-space`. It signals the condition `invalid-description-for-merging-space-information`.

⇒ `merge-space`
client
(description `space-mixin`*)*
value [*Method*]

This is the main method provided on `merge-space` and it is specialized to `space-mixin`.

⇒ `merge-safety` *client description value* [*Generic Function*]

Given an instance of the class `description` and an integer between 0 and 3, return a new instance that is just like *description* (including the class and the values of all the slots), except that its safety information has been updated according to that of *value*.

⇒ `invalid-description-for-merging-safety-information` [*Condition*]

This condition is signaled by `merge-safety` when the *description* argument is not an instance of a class that contains information about safety.

⇒ `merge-safety` *client description safety* [*Method*]

This is the default method provided on `merge-safety`. It signals the condition `invalid-description-for-merging-safety-information`.

⇒ `merge-safety`
client
(description `safety-mixin`*)*
value [*Method*]

This is the main method provided on `merge-safety` and it is specialized to `safety-mixin`.

3.4 High-level annotation functions

3.4.1 Adding and annotating variables

Adding a lexical variable

⇒ `add-lexical-variable` *client environment name* `&optional` *identity* [*Generic Function*]

This function returns a new environment that is like *environment* except that it has been augumented with a lexical variable named *name*. The optional argument *identity* can be supplied by client code to distinguish different lexical variables with the same name.

⇒ `add-lexical-variable` *client (environment environment) name* `&optional` *identity* [*Method*]

This is the main method on `add-lexical-variable`. It instantiates the class `lexical-variable-description` and then creates a new environment by calling the function `augment-with-variable-description`.

Adding a special variable

⇒ `add-special-variable` *client environment name* [*Generic Function*]

This function returns a new environment that is like *environment* except that it has been augumented with a special variable named *name*.

⇒ `add-special-variable` *client (environment environment) name* [*Method*]

This is the main method on `add-special-variable`. It instantiates the class `special-variable-description` and then creates a new environment by calling the function `augment-with-variable-description`.

Adding a local symbol macro

⇒ `add-local-symbol-macro` *client environment name expansion* [*Generic Function*]

This function returns a new environment that is like *environment* except that it has been augmented with a local symbol macro named `name`, with the expansion *expansion*

⇒ `add-local-symbol-macro` *client (environment environment) name expansion* [*Method*]

This is the main method on `add-local-symbol-macro`. It instantiates the class `local-symbol-macro-description` and then creates a new environment by calling the function `augment-with-variable-description`.

Annotating a variable with a type

⇒ `add-variable-type` *client environment name type* [*Generic Function*]

This function returns a new environment that is like *environment* except that the variable named *name* has been annotated with the type specifier *type*.

The type of the variable returned when the new environment is queried for the variable named *name* will have a new type that is the conjunction of *type* and the type it had in *environment*.

This function can be used when *name* names a lexical variable, a special variable, or a symbol macro, but *not* when *name* names a constant variable.

⇒ `add-variable-type` *client (environment environment) name type* [*Method*]

This is the main method on `add-variable-type`. It calls `describe-variable` to obtain an existing variable description. It then calls `merge-type` to create a new variable description. Finally, it calls `augment-with-variable-description` in order to create and return a new environment.

Annotating a variable with an `ignore` declaration

⇒ `add-variable-ignore` *client environment name ignore* [*Generic Function*]

This function returns a new environment that is like *environment* except that the variable named *name* has been annotated with an `ignore` declaration.

The argument *ignore* must be the symbol `ignore` or the symbol `ignorable`.

This function can be used when *name* names a lexical variable or a local symbol macro.

⇒ `add-variable-ignore` *client (environment environment) name ignore* [*Method*]

This is the main method on `add-variable-ignore`. It calls `describe-variable` to obtain an existing variable description. It then calls `merge-ignore` to create a new variable description. Finally, it calls `augment-with-variable-description` in order to create and return a new environment.

Annotating a variable with a `dynamic-extent` declaration

⇒ `add-variable-dynamic-extent` *client environment name* [*Generic Function*]

This function returns a new environment that is like *environment* except that the variable named *name* has been annotated with an `dynamic-extent` declaration.

This function can be used only when *name* names a lexical variable.

⇒ `add-variable-dynamic-extent` *client (environment environment) name* [*Method*]

This is the main method on `add-variable-dynamic-extent`. It calls `describe-variable` to obtain an existing variable description. It then calls `merge-dynamic-extent` to create a new variable description. Finally, it calls `augment-with-variable-description` in order to create and return a new environment.

3.4.2 Adding and annotating functions

Adding a local function

⇒ `add-local-function` *client environment name* `&optional` *identity* [*Generic Function*]

This function returns a new environment that is like *environment* except that it has been augumented with a local function named *name*. The optional argument *identity* can be supplied by client code to distinguish different functions with the same name.

⇒ `add-local-function` *client (environment environment) name* `&optional` *identity* [*Method*]

This is the main method on `add-local-function`. It instantiates the class `local-function-description` and then creates a new environment by calling the function `augment-with-function-description`.

Adding a local macro

⇒ `add-local-macro` *client environment name expander* [*Generic Function*]

This function returns a new environment that is like *environment* except that it has been augmented with a local macro named `name`. The argument *expander* is a macro-expansion function that takes two arguments, a form and an environment.

⇒ `add-local-macro` *client (environment environment) name expander* [*Method*]

This is the main method on `add-local-macro`. It instantiates the class named `local-macro-description` and then creates a new environment by calling the function `augment-with-function-description`.

Annotating a function with a type

⇒ `add-function-type` *client environment name type* [*Generic Function*]

This function returns a new environment that is like *environment* except that the function named *name* has been annotated with the type specifier *type*.

The type of the function returned when the new environment is queried for the function named *name* will have a new type that is the conjunction of *type* and the type it had in *environment*.

This function can be used when *name* names a local function or a global function.

⇒ `add-function-type` *client (environment environment) name type* [*Method*]

This is the main method on `add-function-type`. It calls `describe-function` to obtain an existing function description. It then calls `merge-type` to create a new function description. Finally, it calls `augment-with-function-description` in order to create and return a new environment.

Annotating a function with an `ignore` declaration

⇒ `add-function-ignore` *client environment name ignore* [*Generic Function*]

This function returns a new environment that is like *environment* except that the function named *name* has been annotated with an `ignore` declaration.

The argument *ignore* must be the symbol `ignore` or the symbol `ignorable`.

This function can be used when *name* names a local function or a local macro.

⇒ `add-function-ignore` *client (environment environment) name ignore* [*Method*]

This is the main method on `add-function-ignore`. It calls `describe-function` to obtain an existing function description. It then calls `merge-ignore` to create a new function description. Finally, it calls `augment-with-function-description` in order to create and return a new environment.

Annotating a function with a `dynamic-extent` declaration

⇒ `add-function-dynamic-extent` *client environment name* [*Generic Function*]

This function returns a new environment that is like *environment* except that the function named *name* has been annotated with an `dynamic-extent` declaration.

This function can be used only when *name* names a local function.

⇒ `add-function-dynamic-extent` *client (environment environment) name* [*Method*]

This is the main method on `add-function-dynamic-extent`. It calls `describe-function` to obtain an existing variable description. It then calls `merge-dynamic-extent` to create a new variable description. Finally, it calls `augment-with-function-description` in order to create and return a new environment.

Annotating a function with an `inline` declaration

⇒ `add-inline` *client environment name inline* [*Generic Function*]

This function returns a new environment that is like *environment* except that the function named *name* has been annotated with an `inline` declaration.

The argument *inline* must be the symbol `inline` or the symbol `notinline`.

This function can be used when *name* names a local function or a local macro.

⇒ `add-inline` *client (environment environment) name inline* [*Method*]

This is the main method on `add-inline`. It calls `describe-function` to obtain an existing function description. It then calls `merge-inline` to create a new function description. Finally, it calls `augment-with-function-description` in order to create and return a new environment.

3.4.3 Adding blocks

⇒ `add-block` *client environment name* `&optional` *identity* [*Generic Function*]

This function returns a new environment that is like *environment* except that it has been augumented with a block named *name*, which must be a symbol. The optional argument *identity* can be supplied by client code to distinguish different blocks with the same name.

⇒ `add-block` *client (environment environment) name* `&optional` *identity* [*Method*]

This is the main method on `add-block`. It instantiates the class `block-description` and then creates a new environment by calling the function `augment-with-block-descriptio`

3.4.4 Adding tags

⇒ `add-tag` *client environment tag* `&optional` *identity* [*Generic Function*]

This function returns a new environment that is like *environment* except that it has been augumented with a tag named *tag*, which must be a *go tag*, i.e. a symbol or an integer. The optional argument *identity* can be supplied by client code to distinguish different tags with the same name.

⇒ `add-tag` *client (environment environment) tag* `&optional` *identity* [*Method*]

This is the main method on `add-tag`. It instantiates the class `tag-description` and then creates a new environment by calling the function `augment-with-tag-description`.

3.4.5 Annotating the `optimize` qualities

Annotating `optimize` with a `speed` value

⇒ `add-speed` *client environment value* [*Generic Function*]

This function returns a new environment that is like *environment* except that the `optimize` information has been updated with a `speed` quality value.

The argument *value* must be an integer between 0 and 3.

⇒ `add-speed` *client (environment environment) value* [*Method*]

This is the main method on `add-speed`. It calls `describe-optimize` to obtain the existing optimize description. It then calls `merge-speed` to create a new optimize description. Finally, it calls `augment-with-optimize-description` in order to create and return a new environment.

Annotating `optimize` with a `compilation-speed` value

⇒ `add-compilation-speed` *client environment value* [*Generic Function*]

This function returns a new environment that is like *environment* except that the `optimize` information has been updated with a `compilation-speed` quality value.

The argument *value* must be an integer between 0 and 3.

⇒ `add-compilation-speed` *client (environment environment) value* [*Method*]

This is the main method on `add-compilation-speed`. It calls `describe-optimize` to obtain the existing optimize description. It then calls `merge-compilation-speed` to create a new optimize description. Finally, it calls `augment-with-optimize-description` in order to create and return a new environment.

Annotating `optimize` with a `debug` value

⇒ `add-debug` *client environment value* [*Generic Function*]

This function returns a new environment that is like *environment* except that the `optimize` information has been updated with a `debug` quality value.

The argument *value* must be an integer between 0 and 3.

⇒ `add-debug` *client (environment environment) value* [*Method*]

This is the main method on `add-debug`. It calls `describe-optimize` to obtain the existing optimize description. It then calls `merge-debug` to create a new optimize description. Finally, it calls `augment-with-optimize-description` in order to create and return a new environment.

Annotating `optimize` with a `safety` value

⇒ `add-safety` *client environment value* [*Generic Function*]

This function returns a new environment that is like *environment* except that the `optimize` information has been updated with a `safety` quality value.

The argument *value* must be an integer between 0 and 3.

⇒ `add-safety` *client (environment environment) value* [*Method*]

This is the main method on `add-safety`. It calls `describe-optimize` to obtain the existing optimize description. It then calls `merge-safety` to create a new optimize description. Finally, it calls `augment-with-optimize-description` in order to create and return a new environment.

Annotating `optimize` with a `space` value

⇒ `add-space` *client environment value* [*Generic Function*]

This function returns a new environment that is like *environment* except that the `optimize` information has been updated with a `space` quality value.

The argument *value* must be an integer between 0 and 3.

⇒ `add-space` *client (environment environment) value* [*Method*]

This is the main method on `add-space`. It calls `describe-optimize` to obtain the existing optimize description. It then calls `merge-space` to create a new optimize description. Finally, it calls `augment-with-optimize-description` in order to create and return a new environment.

Bibliography

[Ste90] Guy L. Steele, Jr. *Common LISP: The Language (2Nd Ed.).* Digital Press, Newton, MA, USA, 1990.

Index