Magit-Section Developer Manual

for version 4.1.1

Jonas Bernoulli

Table of Contents

1 Introduction

This package implements the main user interface of Magit — the collapsible sections that make up its buffers. This package used to be distributed as part of Magit but how it can also be used by other packages that have nothing to do with Magit or Git.

To learn more about the section abstraction and available commands and user options see Section “Sections” in `magit`. This manual documents how you can use sections in your own packages.

When the documentation leaves something unaddressed, then please consider that Magit uses this library extensively and search its source for suitable examples before asking me for help. Thanks!

2 Creating Sections

`magit-insert-section` [*name*] (*type* **&optional** *value hide*) **&rest** *body* [Macro]

Create a section object of type CLASS, storing VALUE in its `value` slot, and insert the section at point. CLASS is a subclass of 'magit-section' or has the form `(eval FORM)`, in which case FORM is evaluated at runtime and should return a subclass. In other places a sections class is often referred to as its "type".

Many commands behave differently depending on the class of the current section and sections of a certain class can have their own keymap, which is specified using the 'keymap' class slot. The value of that slot should be a variable whose value is a keymap.

For historic reasons Magit and Forge in most cases use symbols as CLASS that don't actually identify a class and that lack the appropriate package prefix. This works due to some undocumented kludges, which are not available to other packages.

When optional HIDE is non-nil collapse the section body by default, i.e., when first creating the section, but not when refreshing the buffer. Else expand it by default. This can be overwritten using `magit-section-set-visibility-hook`. When a section is recreated during a refresh, then the visibility of predecessor is inherited and HIDE is ignored (but the hook is still honored).

BODY is any number of forms that actually insert the section's heading and body. Optional NAME, if specified, has to be a symbol, which is then bound to the object of the section being inserted.

Before BODY is evaluated the `start` of the section object is set to the value of 'point' and after BODY was evaluated its `end` is set to the new value of `point`; BODY is responsible for moving `point` forward.

If it turns out inside BODY that the section is empty, then `magit-cancel-section` can be used to abort and remove all traces of the partially inserted section. This can happen when creating a section by washing Git's output and Git didn't actually output anything this time around.

`magit-insert-heading` [*child-count*] **&rest** *args* [Function]

Insert the heading for the section currently being inserted.

This function should only be used inside `magit-insert-section`.

When called without any arguments, then just set the `content` slot of the object representing the section being inserted to a marker at `point`. The section should only contain a single line when this function is used like this.

When called with arguments ARGS, which have to be strings, or nil, then insert those strings at point. The section should not contain any text before this happens and afterwards it should again only contain a single line. If the `face` property is set anywhere inside any of these strings, then insert all of them unchanged. Otherwise use the 'magit-section-heading' face for all inserted text.

The `content` property of the section object is the end of the heading (which lasts from `start` to `content`) and the beginning of the the body (which lasts from `content` to `end`). If the value of `content` is nil, then the section has no heading and its

body cannot be collapsed. If a section does have a heading, then its height must be exactly one line, including a trailing newline character. This isn't enforced, you are responsible for getting it right. The only exception is that this function does insert a newline character if necessary.

If provided, optional CHILD-COUNT must evaluate to an integer or boolean. If t, then the count is determined once the children have been inserted, using `magit-insert-child-count` (which see). For historic reasons, if the heading ends with `":"`, the count is substituted for that, at this time as well. If `magit-section-show-child-count` is nil, no counts are inserted

`magit-insert-section-body` **&rest** *body* [Macro]

Use BODY to insert the section body, once the section is expanded. If the section is expanded when it is created, then this is like `progn`. Otherwise BODY isn't evaluated until the section is explicitly expanded.

`magit-cancel-section` [Function]

Cancel inserting the section that is currently being inserted. Remove all traces of that section.

`magit-wash-sequence` *function* [Function]

Repeatedly call FUNCTION until it returns `nil` or the end of the buffer is reached. FUNCTION has to move point forward or return `nil`.

3 Core Functions

`magit-current-section` [Function]
Return the section at point or where the context menu was invoked. When using the context menu, return the section that the user clicked on, provided the current buffer is the buffer in which the click occurred. Otherwise return the section at point.

Function magit-section-at &optional position
Return the section at POSITION, defaulting to point. Default to point even when the context menu is used.

`magit-section-ident` *section* [Function]
Return an unique identifier for SECTION. The return value has the form `((TYPE . VALUE)...)`.

`magit-section-ident-value` *value* [Function]
Return a constant representation of VALUE.

VALUE is the value of a `magit-section` object. If that is an object itself, then that is not suitable to be used to identify the section because two objects may represent the same thing but not be equal. If possible a method should be added for such objects, which returns a value that is equal. Otherwise the catch-all method is used, which just returns the argument itself.

`magit-get-section` *ident* **&optional** *root* [Function]
Return the section identified by IDENT. IDENT has to be a list as returned by `magit-section-ident`. If optional ROOT is non-nil, then search in that section tree instead of in the one whose root `magit-root-section` is.

`magit-section-lineage` *section* **&optional** *raw* [Function]
Return the lineage of SECTION. If optional RAW is non-nil, return a list of section objects, beginning with SECTION, otherwise return a list of section types.

`magit-section-content-p` *section* [Function]
Return non-nil if SECTION has content or an unused washer function.

The next two functions are replacements for the Emacs functions that have the same name except for the `magit-` prefix. Like `magit-current-section` they do not act on point, the cursors position, but on the position where the user clicked to invoke the context menu.

If your package provides a context menu and some of its commands act on the "thing at point", even if just as a default, then use the prefixed functions to teach them to instead use the click location when appropriate.

Function magit-point
Return point or the position where the context menu was invoked. When using the context menu, return the position the user clicked on, provided the current buffer is the buffer in which the click occurred. Otherwise return the same value as `point`.

Function magit-thing-at-point thing &optional no-properties

Return the THING at point or where the context menu was invoked. When using the context menu, return the thing the user clicked on, provided the current buffer is the buffer in which the click occurred. Otherwise return the same value as `thing-at-point`. For the meaning of THING and NO-PROPERTIES see that function.

4 Matching Functions

`magit-section-match` *condition* **&optional** (*section* **(magit-current-section)**) [Function]

Return t if SECTION matches CONDITION.

SECTION defaults to the section at point. If SECTION is not specified and there also is no section at point, then return nil.

CONDITION can take the following forms:

- `(CONDITION...)` matches if any of the CONDITIONs matches.
- `[CLASS...]` matches if the section's class is the same as the first CLASS or a subclass of that; the section's parent class matches the second CLASS; and so on.
- `[* CLASS...]` matches sections that match [CLASS...] and also recursively all their child sections.
- `CLASS` matches if the section's class is the same as CLASS or a subclass of that; regardless of the classes of the parent sections.

Each CLASS should be a class symbol, identifying a class that derives from `magit-section`. For backward compatibility CLASS can also be a "type symbol". A section matches such a symbol if the value of its `type` slot is `eq`. If a type symbol has an entry in `magit--section-type-alist`, then a section also matches that type if its class is a subclass of the class that corresponds to the type as per that alist.

Note that it is not necessary to specify the complete section lineage as printed by `magit-describe-section-briefly`, unless of course you want to be that precise.

`magit-section-value-if` *condition* **&optional** *section* [Function]

If the section at point matches CONDITION, then return its value.

If optional SECTION is non-nil then test whether that matches instead. If there is no section at point and SECTION is nil, then return nil. If the section does not match, then return nil.

See `magit-section-match` for the forms CONDITION can take.

`magit-section-case` **&rest** *clauses* [Macro]

Choose among clauses on the type of the section at point.

Each clause looks like `(CONDITION BODY...)`. The type of the section is compared against each CONDITION; the BODY forms of the first match are evaluated sequentially and the value of the last form is returned. Inside BODY the symbol `it` is bound to the section at point. If no clause succeeds or if there is no section at point, return nil.

See `magit-section-match` for the forms CONDITION can take. Additionally a CONDITION of t is allowed in the final clause, and matches if no other CONDITION match, even if there is no section at point.