chromedp package

Version: v0.0.4 Latest | Published: Jun 30, 2024 | License: MIT, MIT | Imports: 44 | Imported by: 0

Details Valid go.mod file Redistributable license Tagged version Stable version Learn more about best practices

Repository github.com/cdvelop/vanify

README

README ¶

About chromedp

Package `chromedp` is a faster, simpler way to drive browsers supporting the Chrome DevTools Protocol in Go without external dependencies.

Installing

Install in the usual Go way:

Expand ▾

Documentation

linux/amd64

Overview

Package chromedp is a high level Chrome DevTools Protocol client that simplifies driving browsers for scraping, unit testing, or profiling web pages using the CDP.

chromedp requires no third-party dependencies, implementing the async Chrome DevTools Protocol entirely in Go.

This package includes a number of simple examples. Additionally, chromedp/examples contains more complex examples.

▶ Example (DocumentDump)

▶ Example (Dump)

▶ Example (RetrieveHTML)

Index

func EmulateLandscape(p1 *emulation.SetDeviceMetricsOverrideParams, ...)
func EmulateMobile(p1 *emulation.SetDeviceMetricsOverrideParams, ...)
func EmulatePortrait(p1 *emulation.SetDeviceMetricsOverrideParams, ...)
func EmulateTouch(p1 *emulation.SetDeviceMetricsOverrideParams, ...)
func EvalAsValue(p *runtime.EvaluateParams) *runtime.EvaluateParams
func EvalIgnoreExceptions(p *runtime.EvaluateParams) *runtime.EvaluateParams
func EvalWithCommandLineAPI(p *runtime.EvaluateParams) *runtime.EvaluateParams
func Headless(a *ExecAllocator)
func IgnoreCertErrors(a *ExecAllocator)
func ListenBrowser(ctx context.Context, fn func(ev interface{}))
func ListenTarget(ctx context.Context, fn func(ev interface{}))
func NewContext(parent context.Context, opts ...ContextOption) (context.Context, context.CancelFunc)
func NewExecAllocator(parent context.Context, opts ...ExecAllocatorOption) (context.Context, context.CancelFunc)
func NewRemoteAllocator(parent context.Context, url string, opts ...RemoteAllocatorOption) (context.Context, context.CancelFunc)
func NoDefaultBrowserCheck(a *ExecAllocator)
func NoFirstRun(a *ExecAllocator)
func NoModifyURL(a *RemoteAllocator)
func NoSandbox(a *ExecAllocator)
func NodeEnabled(s *Selector)
func NodeNotPresent(s *Selector)
func NodeNotVisible(s *Selector)
func NodeReady(s *Selector)
func NodeSelected(s *Selector)
func NodeVisible(s *Selector)
func Run(ctx context.Context, actions ...Action) error
func RunResponse(ctx context.Context, actions ...Action) (*network.Response, error)
func Targets(ctx context.Context) ([]*target.Info, error)
func WaitNewTarget(ctx context.Context, fn func(*target.Info) bool) <-chan target.ID
type Action
	func CaptureScreenshot(res *[]byte) Action
	func Location(urlstr *string) Action

func Stop() Action
func Title(title *string) Action
type ActionFunc
func (f ActionFunc) Do(ctx context.Context) error
type Allocator
type Browser
func NewBrowser(ctx context.Context, urlstr string, opts ...BrowserOption) (*Browser, error)
func (b *Browser) Execute(ctx context.Context, method string, params easyjson.Marshaler, ...) error
func (b *Browser) Process() *os.Process
type BrowserOption
func WithBrowserDebugf(f func(string, ...interface{})) BrowserOption
func WithBrowserErrorf(f func(string, ...interface{})) BrowserOption
func WithBrowserLogf(f func(string, ...interface{})) BrowserOption
func WithConsolef(f func(string, ...interface{})) BrowserOption
func WithDialTimeout(d time.Duration) BrowserOption
type CallAction
func CallFunctionOn(functionDeclaration string, res interface{}, opt CallOption, ...) CallAction
type CallOption
type Conn
func DialContext(ctx context.Context, urlstr string, opts ...DialOption) (*Conn, error)
func (c *Conn) Close() error
func (c *Conn) Read(_ context.Context, msg *cdproto.Message) error
func (c *Conn) Write(_ context.Context, msg *cdproto.Message) error
type Context
func FromContext(ctx context.Context) *Context
type ContextOption
func WithBrowserOption(opts ...BrowserOption) ContextOption
func WithDebugf(f func(string, ...interface{})) ContextOption
func WithErrorf(f func(string, ...interface{})) ContextOption
func WithExistingBrowserContext(id cdp.BrowserContextID) ContextOption
func WithLogf(f func(string, ...interface{})) ContextOption

type Device
type DialOption
func WithConnDebugf(f func(string, ...interface{})) DialOption
type EmulateAction
func Emulate(device Device) EmulateAction
func EmulateReset() EmulateAction
func EmulateViewport(width, height int64, opts ...EmulateViewportOption) EmulateAction
func FullScreenshot(res *[]byte, quality int) EmulateAction
func ResetViewport() EmulateAction
type EmulateViewportOption
func EmulateOrientation(orientation emulation.OrientationType, angle int64) EmulateViewportOption
func EmulateScale(scale float64) EmulateViewportOption
type Error
func (err Error) Error() string
type EvaluateAction
func Evaluate(expression string, res interface{}, opts ...EvaluateOption) EvaluateAction
func EvaluateAsDevTools(expression string, res interface{}, opts ...EvaluateOption) EvaluateAction
type EvaluateOption
func EvalObjectGroup(objectGroup string) EvaluateOption
type ExecAllocator
func (a *ExecAllocator) Allocate(ctx context.Context, opts ...BrowserOption) (*Browser, error)
func (a *ExecAllocator) Wait()
type ExecAllocatorOption
func CombinedOutput(w io.Writer) ExecAllocatorOption
func Env(vars ...string) ExecAllocatorOption
func ExecPath(path string) ExecAllocatorOption
func Flag(name string, value interface{}) ExecAllocatorOption
func ModifyCmdFunc(f func(cmd *exec.Cmd)) ExecAllocatorOption
func ProxyServer(proxy string) ExecAllocatorOption
func UserAgent(userAgent string) ExecAllocatorOption
func UserDataDir(dir string) ExecAllocatorOption

func KeyEvent(keys string, opts ...KeyOption) KeyAction
func KeyEventNode(n *cdp.Node, keys string, opts ...KeyOption) KeyAction
type KeyOption
func KeyModifiers(modifiers ...input.Modifier) KeyOption
type MouseAction
func MouseClickNode(n *cdp.Node, opts ...MouseOption) MouseAction
func MouseClickXY(x, y float64, opts ...MouseOption) MouseAction
func MouseEvent(typ input.MouseType, x, y float64, opts ...MouseOption) MouseAction
type MouseOption
func Button(btn string) MouseOption
func ButtonModifiers(modifiers ...input.Modifier) MouseOption
func ButtonType(button input.MouseButton) MouseOption
func ClickCount(n int) MouseOption
type NavigateAction
func Navigate(urlstr string) NavigateAction
func NavigateBack() NavigateAction
func NavigateForward() NavigateAction
func NavigateToHistoryEntry(entryID int64) NavigateAction
func Reload() NavigateAction
type PollAction
func Poll(expression string, res interface{}, opts ...PollOption) PollAction
func PollFunction(pageFunction string, res interface{}, opts ...PollOption) PollAction
type PollOption
func WithPollingArgs(args ...interface{}) PollOption
func WithPollingInFrame(frame *cdp.Node) PollOption
func WithPollingInterval(interval time.Duration) PollOption
func WithPollingMutation() PollOption
func WithPollingTimeout(timeout time.Duration) PollOption
type PopulateOption
func PopulateWait(wait time.Duration) PopulateOption
type QueryAction

func Blur(sel interface{}, opts ...QueryOption) QueryAction
func Clear(sel interface{}, opts ...QueryOption) QueryAction
func Click(sel interface{}, opts ...QueryOption) QueryAction
func ComputedStyle(sel interface{}, style *[]*css.ComputedStyleProperty, opts ...QueryOption) QueryAction
func Dimensions(sel interface{}, model **dom.BoxModel, opts ...QueryOption) QueryAction
func DoubleClick(sel interface{}, opts ...QueryOption) QueryAction
func Dump(sel interface{}, w io.Writer, opts ...QueryOption) QueryAction
func DumpTo(sel interface{}, w io.Writer, prefix, indent string, nodeIDs bool, depth int64, ...) QueryAction
func Focus(sel interface{}, opts ...QueryOption) QueryAction
func InnerHTML(sel interface{}, html *string, opts ...QueryOption) QueryAction
func JavascriptAttribute(sel interface{}, name string, res interface{}, opts ...QueryOption) QueryAction
func MatchedStyle(sel interface{}, style **css.GetMatchedStylesForNodeReturns, ...) QueryAction
func NodeIDs(sel interface{}, ids *[]cdp.NodeID, opts ...QueryOption) QueryAction
func Nodes(sel interface{}, nodes *[]*cdp.Node, opts ...QueryOption) QueryAction
func OuterHTML(sel interface{}, html *string, opts ...QueryOption) QueryAction
func Query(sel interface{}, opts ...QueryOption) QueryAction
func QueryAfter(sel interface{}, ...) QueryAction
func RemoveAttribute(sel interface{}, name string, opts ...QueryOption) QueryAction
func Reset(sel interface{}, opts ...QueryOption) QueryAction
func Screenshot(sel interface{}, picbuf *[]byte, opts ...QueryOption) QueryAction
func ScreenshotScale(sel interface{}, scale float64, picbuf *[]byte, opts ...QueryOption) QueryAction
func ScrollIntoView(sel interface{}, opts ...QueryOption) QueryAction
func SendKeys(sel interface{}, v string, opts ...QueryOption) QueryAction
func SetAttributeValue(sel interface{}, name, value string, opts ...QueryOption) QueryAction
func SetAttributes(sel interface{}, attributes map[string]string, opts ...QueryOption) QueryAction
func SetJavascriptAttribute(sel interface{}, name, value string, opts ...QueryOption) QueryAction
func SetUploadFiles(sel interface{}, files []string, opts ...QueryOption) QueryAction
func SetValue(sel interface{}, value string, opts ...QueryOption) QueryAction
func Submit(sel interface{}, opts ...QueryOption) QueryAction
func Text(sel interface{}, text *string, opts ...QueryOption) QueryAction
func TextContent(sel interface{}, text *string, opts ...QueryOption) QueryAction

func WaitNotVisible(sel interface{}, opts ...QueryOption) QueryAction
func WaitReady(sel interface{}, opts ...QueryOption) QueryAction
func WaitSelected(sel interface{}, opts ...QueryOption) QueryAction
func WaitVisible(sel interface{}, opts ...QueryOption) QueryAction
type QueryOption
func After(f func(context.Context, runtime.ExecutionContextID, ...*cdp.Node) error) QueryOption
func AtLeast(n int) QueryOption
func ByFunc(f func(context.Context, *cdp.Node) ([]cdp.NodeID, error)) QueryOption
func FromNode(node *cdp.Node) QueryOption
func Populate(depth int64, pierce bool, opts ...PopulateOption) QueryOption
func RetryInterval(interval time.Duration) QueryOption
func WaitFunc(...) QueryOption
type RemoteAllocator
func (a *RemoteAllocator) Allocate(ctx context.Context, opts ...BrowserOption) (*Browser, error)
func (a *RemoteAllocator) Wait()
type RemoteAllocatorOption
type Selector
func (s *Selector) Do(ctx context.Context) error
type Target
func (t *Target) Execute(ctx context.Context, method string, params easyjson.Marshaler, ...) error
type Tasks
func (t Tasks) Do(ctx context.Context) error
type Transport

Examples

Package (DocumentDump)
Package (Dump)
Package (RetrieveHTML)
ByJSPath
Emulate
Evaluate

FullScreenshot
ListenTarget (AcceptAlert)
ListenTarget (ConsoleLog)
NewContext (ManyTabs)
NewContext (ReuseBrowser)
RunResponse
Title
WaitNewTarget

Constants

This section is empty.

Variables

View Source

```
var DefaultExecAllocatorOptions = [...]ExecAllocatorOption{
    NoFirstRun,
    NoDefaultBrowserCheck,
    Headless,

    Flag("disable-background-networking", true),
    Flag("enable-features", "NetworkService,NetworkServiceInProcess"),
    Flag("disable-background-timer-throttling", true),
    Flag("disable-backgrounding-occluded-windows", true),
    Flag("disable-breakpad", true),
    Flag("disable-client-side-phishing-detection", true),
    Flag("disable-default-apps", true),
    Flag("disable-dev-shm-usage", true),
    Flag("disable-extensions", true),
    Flag("disable-features", "site-per-process,Translate,BlinkGenPropertyTrees"),
    Flag("disable-hang-monitor", true),
    Flag("disable-ipc-flooding-protection", true),
    Flag("disable-popup-blocking", true),
```

```
    Flag("force-color-profile", "srgb"),
    Flag("metrics-recording-only", true),
    Flag("safebrowsing-disable-auto-update", true),
    Flag("enable-automation", true),
    Flag("password-store", "basic"),
    Flag("use-mock-keychain", true),
}
```

DefaultExecAllocatorOptions are the ExecAllocator options used by NewContext if the given parent context doesn't have an allocator set up. Do not modify this global; instead, use NewExecAllocator. See ExampleExecAllocator.

Functions

func ButtonLeft

```
func ButtonLeft(p *input.DispatchMouseEventParams) *input.DispatchMouseEventParams
```

ButtonLeft is a mouse action option to set the button clicked as the left mouse button.

func ButtonMiddle

```
func ButtonMiddle(p *input.DispatchMouseEventParams) *input.DispatchMouseEventParams
```

ButtonMiddle is a mouse action option to set the button clicked as the middle mouse button.

func ButtonNone

```
func ButtonNone(p *input.DispatchMouseEventParams) *input.DispatchMouseEventParams
```

ButtonNone is a mouse action option to set the button clicked as none (used for mouse movements).

func ButtonRight

ButtonRight is a mouse action option to set the button clicked as the right mouse button.

func ByID

```
func ByID(s *Selector)
```

ByID is an element query option to select a single element by its CSS #id.

Similar to calling document.querySelector('#' + ID) in the browser.

func ByJSPath

```
func ByJSPath(s *Selector)
```

ByJSPath is an element query option to select elements by the "JS Path" value (as shown in the Chrome DevTools UI).

Allows for the direct querying of DOM elements that otherwise cannot be retrieved using the other By* funcs, such as ShadowDOM elements.

Note: Do not use with an untrusted selector value, as any defined selector will be passed to runtime.Evaluate.

▶ Example

func ByNodeID

```
func ByNodeID(s *Selector)
```

ByNodeID is an element query option to select elements by their node IDs.

Uses DOM.requestChildNodes to retrieve elements with specific node IDs.

Note: must be used with []cdp.NodeID.

```
func ByQuery(s *Selector)
```

ByQuery is an element query action option to select a single element by the DOM.querySelector command.

Similar to calling document.querySelector() in the browser.

func ByQueryAll

```
func ByQueryAll(s *Selector)
```

ByQueryAll is an element query action option to select elements by the DOM.querySelectorAll command.

Similar to calling document.querySelectorAll() in the browser.

func BySearch

```
func BySearch(s *Selector)
```

BySearch is an element query option to select elements by the DOM.performSearch command. It matches nodes by plain text, CSS selector or XPath query.

func Cancel

```
func Cancel(ctx context.Context) error
```

Cancel cancels a chromedp context, waits for its resources to be cleaned up, and returns any error encountered during that process.

If the context allocated a browser, the browser will be closed gracefully by Cancel. A timeout can be attached to this context to determine how long to wait for the browser to close itself:

```
tctx, tcancel := context.WithTimeout(ctx, 10 * time.Second)
defer tcancel()
chromedp.Cancel(tctx)
```

Usually a "defer cancel()" will be enough for most use cases. However, Cancel is the better option if one wants to gracefully close a browser, or catch underlying errors happening during cancellation.

func DisableGPU

```
func DisableGPU(a *ExecAllocator)
```

DisableGPU is the command line option to disable the GPU process.

The --disable-gpu option is a temporary workaround for a few bugs in headless mode. According to the references below, it's no longer required:

- https://bugs.chromium.org/p/chromium/issues/detail?id=737678
- https://github.com/puppeteer/puppeteer/pull/2908
- https://github.com/puppeteer/puppeteer/pull/4523

But according to this reported issue, it's still required in some cases:

- https://github.com/chromedp/chromedp/issues/904

func EmulateLandscape

```
func EmulateLandscape(p1 *emulation.SetDeviceMetricsOverrideParams, p2 *emulation.SetTouchEmulationEnabledParams)
```

EmulateLandscape is an emulate viewport option to set the device viewport screen orientation in landscape primary mode and an angle of 90.

func EmulateMobile

```
func EmulateMobile(p1 *emulation.SetDeviceMetricsOverrideParams, p2 *emulation.SetTouchEmulationEnabledParams)
```

EmulateMobile is an emulate viewport option to toggle the device viewport to display as a mobile device.

EmulatePortrait is an emulate viewport option to set the device viewport screen orientation in portrait primary mode and an angle of 0.

func EmulateTouch

```
func EmulateTouch(p1 *emulation.SetDeviceMetricsOverrideParams, p2 *emulation.SetTouchEmulationEnabledParams)
```

EmulateTouch is an emulate viewport option to enable touch emulation.

func EvalAsValue

```
func EvalAsValue(p *runtime.EvaluateParams) *runtime.EvaluateParams
```

EvalAsValue is an evaluate option that will cause the evaluated JavaScript expression to encode the result of the expression as a JSON-encoded value.

func EvalIgnoreExceptions

```
func EvalIgnoreExceptions(p *runtime.EvaluateParams) *runtime.EvaluateParams
```

EvalIgnoreExceptions is an evaluate option that will cause JavaScript evaluation to ignore exceptions.

func EvalWithCommandLineAPI

```
func EvalWithCommandLineAPI(p *runtime.EvaluateParams) *runtime.EvaluateParams
```

EvalWithCommandLineAPI is an evaluate option to make the DevTools Command Line API available to the evaluated script.

See Evaluate for more information on how evaluate actions work.

Note: this should not be used with untrusted JavaScript.

func Headless

Headless is the command line option to run in headless mode. On top of setting the headless flag, it also hides scrollbars and mutes audio.

func IgnoreCertErrors

```
func IgnoreCertErrors(a *ExecAllocator)
```

IgnoreCertErrors is the command line option to ignore certificate-related errors. This option is useful when you need to access an HTTPS website through a proxy.

func ListenBrowser

```
func ListenBrowser(ctx context.Context, fn func(ev interface{}))
```

ListenBrowser adds a function which will be called whenever a browser event is received on the chromedp context. Note that this only includes browser events; command responses and target events are not included. Cancelling ctx stops the listener from receiving any more events.

Note that the function is called synchronously when handling events. The function should avoid blocking at all costs. For example, any Actions must be run via a separate goroutine (otherwise, it could result in a deadlock if the action sends CDP messages).

func ListenTarget

```
func ListenTarget(ctx context.Context, fn func(ev interface{}))
```

ListenTarget adds a function which will be called whenever a target event is received on the chromedp context. Cancelling ctx stops the listener from receiving any more events.

Note that the function is called synchronously when handling events. The function should avoid blocking at all costs. For example, any Actions must be run via a separate goroutine (otherwise, it could result in a deadlock if the action sends CDP messages).

▶ Example (AcceptAlert)

func NewContext

```
func NewContext(parent context.Context, opts ...ContextOption) (context.Context, context.CancelFunc)
```

NewContext creates a chromedp context from the parent context. The parent context's Allocator is inherited, defaulting to an ExecAllocator with DefaultExecAllocatorOptions.

If the parent context contains an allocated Browser, the child context inherits it, and its first Run creates a new tab on that browser. Otherwise, its first Run will allocate a new browser.

Cancelling the returned context will close a tab or an entire browser, depending on the logic described above. To cancel a context while checking for errors, see Cancel.

Note that NewContext doesn't allocate nor start a browser; that happens the first time Run is used on the context.

▶ Example (ManyTabs)

▶ Example (ReuseBrowser)

func NewExecAllocator

```
func NewExecAllocator(parent context.Context, opts ...ExecAllocatorOption) (context.Context, context.CancelFunc)
```

NewExecAllocator creates a new context set up with an ExecAllocator, suitable for use with NewContext.

func NewRemoteAllocator

```
func NewRemoteAllocator(parent context.Context, url string, opts ...RemoteAllocatorOption) (context.Context, context.CancelFunc)
```

NewRemoteAllocator creates a new context set up with a RemoteAllocator, suitable for use with NewContext. The url should point to

If the url does not contain "/devtools/browser/", it will try to detect the correct one by sending a request to "http://$HOST:$PORT/json/version".

The url with the following formats are accepted:

- ws://127.0.0.1:9222/
- http://127.0.0.1:9222/

But "ws://127.0.0.1:9222/devtools/browser/" are not accepted. Because the allocator won't try to modify it and it's obviously invalid.

Use chromedp.NoModifyURL to prevent it from modifying the url.

func NoDefaultBrowserCheck

```
func NoDefaultBrowserCheck(a *ExecAllocator)
```

NoDefaultBrowserCheck is the Chrome command line option to disable the default browser check.

func NoFirstRun

```
func NoFirstRun(a *ExecAllocator)
```

NoFirstRun is the Chrome command line option to disable the first run dialog.

func NoModifyURL

```
func NoModifyURL(a *RemoteAllocator)
```

NoModifyURL is a RemoteAllocatorOption that prevents the remote allocator from modifying the websocket debugger URL passed to it.

func NoSandbox

```
func NoSandbox(a *ExecAllocator)
```

func NodeEnabled

```
func NodeEnabled(s *Selector)
```

NodeEnabled is an element query option to wait until all queried element nodes have been sent by the browser and are enabled (i.e., do not have a 'disabled' attribute).

func NodeNotPresent

```
func NodeNotPresent(s *Selector)
```

NodeNotPresent is an element query option to wait until no elements are present that match the query.

Note: forces the expected number of element nodes to be 0.

func NodeNotVisible

```
func NodeNotVisible(s *Selector)
```

NodeNotVisible is an element query option to wait until all queried element nodes have been sent by the browser and are not visible.

func NodeReady

```
func NodeReady(s *Selector)
```

NodeReady is an element query option to wait until all queried element nodes have been sent by the browser.

func NodeSelected

```
func NodeSelected(s *Selector)
```

NodeSelected is an element query option to wait until all queried element nodes have been sent by the browser and are selected (i.e., has 'selected' attribute).

```
func NodeVisible(s *Selector)
```

NodeVisible is an element query option to wait until all queried element nodes have been sent by the browser and are visible.

func Run

```
func Run(ctx context.Context, actions ...Action) error
```

Run runs an action against context. The provided context must be a valid chromedp context, typically created via NewContext.

Note that the first time Run is called on a context, a browser will be allocated via Allocator. Thus, it's generally a bad idea to use a context timeout on the first Run call, as it will stop the entire browser.

Also note that the actions are run with the Target executor. In the case that a Browser executor is required, the action can be written like this:

```
err := chromedp.Run(ctx, chromedp.ActionFunc(func(ctx context.Context) error {
    c := chromedp.FromContext(ctx)
    id, err := target.CreateBrowserContext().Do(cdp.WithExecutor(ctx, c.Browser))
    return err
}))
```

func RunResponse

```
func RunResponse(ctx context.Context, actions ...Action) (*network.Response, error)
```

RunResponse is an alternative to Run which can be used with a list of actions that trigger a page navigation, such as clicking on a link or button.

RunResponse will run the actions and block until a page loads, returning the HTTP response information for its HTML document. This can be useful to wait for the page to be ready, or to catch 404 status codes, for example.

Note that if the actions trigger multiple navigations, only the first is used. And if the actions trigger no navigations at all, RunResponse

▶ Example

func Targets

```
func Targets(ctx context.Context) ([]*target.Info, error)
```

Targets lists all the targets in the browser attached to the given context.

func WaitNewTarget

```
func WaitNewTarget(ctx context.Context, fn func(*target.Info) bool) <-chan target.ID
```

WaitNewTarget can be used to wait for the current target to open a new target. Once fn matches a new unattached target, its target ID is sent via the returned channel.

▶ Example

Types

type Action

```
type Action interface {
    // Do executes the action using the provided context and frame handler.
    Do(context.Context) error
}
```

Action is the common interface for an action that will be executed against a context and frame handler.

func CaptureScreenshot

It's supposed to act the same as the command "Capture screenshot" in Chrome. See the behavior notes of Screenshot for more information.

See the Screenshot action to take a screenshot of a specific element.

See screenshot for an example of taking a screenshot of the entire page.

func Location

```
func Location(urlstr *string) Action
```

Location is an action that retrieves the document location.

func NavigationEntries

```
func NavigationEntries(currentIndex *int64, entries *[]*page.NavigationEntry) Action
```

NavigationEntries is an action that retrieves the page's navigation history entries.

func ScreenshotNodes

```
func ScreenshotNodes(nodes []*cdp.Node, scale float64, picbuf *[]byte) Action
```

ScreenshotNodes is an action that captures/takes a screenshot of the specified nodes, by calculating the extents of the top most left node and bottom most right node.

func Sleep

```
func Sleep(d time.Duration) Action
```

Sleep is an empty action that calls time.Sleep with the specified duration.

Note: this is a temporary action definition for convenience, and will likely be marked for deprecation in the future, after the remaining

```
func Stop() Action
```

Stop is an action that stops all navigation and pending resource retrieval.

func Title

```
func Title(title *string) Action
```

Title is an action that retrieves the document title.

▶ Example

type ActionFunc

```
type ActionFunc func(context.Context) error
```

ActionFunc is an adapter to allow the use of ordinary func's as an Action.

func (ActionFunc) Do

```
func (f ActionFunc) Do(ctx context.Context) error
```

Do executes the func f using the provided context and frame handler.

type Allocator

```
type Allocator interface {
    // Allocate creates a new browser. It can be cancelled via the provided
    // context, at which point all the resources used by the browser (such
    // as temporary directories) will be freed.
    Allocate(context.Context, ...BrowserOption) (*Browser, error)
```

```
    // so normally there's no need to call Wait directly.
    Wait()
}
```

An Allocator is responsible for creating and managing a number of browsers.

This interface abstracts away how the browser process is actually run. For example, an Allocator implementation may reuse browser processes, or connect to already-running browsers on remote machines.

type Browser

```
type Browser struct {

    // LostConnection is closed when the websocket connection to Chrome is
    // dropped. This can be useful to make sure that Browser's context is
    // cancelled (and the handler stopped) once the connection has failed.
    LostConnection chan struct{}
    // contains filtered or unexported fields
}
```

Browser is the high-level Chrome DevTools Protocol browser manager, handling the browser process runner, WebSocket clients, associated targets, and network, page, and DOM events.

func NewBrowser

```
func NewBrowser(ctx context.Context, urlstr string, opts ...BrowserOption) (*Browser, error)
```

NewBrowser creates a new browser. Typically, this function wouldn't be called directly, as the Allocator interface takes care of it.

func (*Browser) Execute

```
func (b *Browser) Execute(ctx context.Context, method string, params easyjson.Marshaler, res easyjson.Unmarshaler) error
```

```
func (b *Browser) Process() *os.Process
```

Process returns the process object of the browser.

It could be nil when the browser is allocated with RemoteAllocator. It could be useful for a monitoring system to collect process metrics of the browser process. (See prometheus.NewProcessCollector for an example).

Example:

```
if process := chromedp.FromContext(ctx).Browser.Process(); process != nil {
    fmt.Printf("Browser PID: %v", process.Pid)
}
```

type BrowserOption

```
type BrowserOption = func(*Browser)
```

BrowserOption is a browser option.

func WithBrowserDebugf

```
func WithBrowserDebugf(f func(string, ...interface{})) BrowserOption
```

WithBrowserDebugf is a browser option to specify a func to log actual websocket messages.

func WithBrowserErrorf

```
func WithBrowserErrorf(f func(string, ...interface{})) BrowserOption
```

WithBrowserErrorf is a browser option to specify a func to receive error logging.

func WithBrowserLogf

WithBrowserLogf is a browser option to specify a func to receive general logging.

func WithConsolef

```
func WithConsolef(f func(string, ...interface{})) BrowserOption
```

WithConsolef is a browser option to specify a func to receive chrome log events.

Note: NOT YET IMPLEMENTED.

func WithDialTimeout

```
func WithDialTimeout(d time.Duration) BrowserOption
```

WithDialTimeout is a browser option to specify the timeout when dialing a browser's websocket address. The default is ten seconds; use a zero duration to not use a timeout.

type CallAction

```
type CallAction Action
```

CallAction are actions that calls a JavaScript function using runtime.CallFunctionOn.

func CallFunctionOn

```
func CallFunctionOn(functionDeclaration string, res interface{}, opt CallOption, args ...interface{}) CallAction
```

CallFunctionOn is an action to call a JavaScript function, unmarshaling the result of the function to res.

The handling of res is the same as that of Evaluate.

Do not call the following methods on runtime.CallFunctionOnParams: - WithReturnByValue: it will be set depending on the type of res; - WithArguments: pass the arguments with args instead.

type CallOption

```
type CallOption = func(params *runtime.CallFunctionOnParams) *runtime.CallFunctionOnParams
```

CallOption is a function to modify the runtime.CallFunctionOnParams to provide more information.

type Conn

```
type Conn struct {
    // contains filtered or unexported fields
}
```

Conn implements Transport with a gobwas/ws websocket connection.

func DialContext

```
func DialContext(ctx context.Context, urlstr string, opts ...DialOption) (*Conn, error)
```

DialContext dials the specified websocket URL using gobwas/ws.

func (*Conn) Close

```
func (c *Conn) Close() error
```

Close satisfies the io.Closer interface.

func (*Conn) Read

```
func (c *Conn) Read(_ context.Context, msg *cdproto.Message) error
```

Read reads the next message.

func (*Conn) Write

Write writes a message.

type Context

```
type Context struct {
    // Allocator is used to create new browsers. It is inherited from the
    // parent context when using NewContext.
    Allocator Allocator

    // Browser is the browser being used in the context. It is inherited
    // from the parent context when using NewContext.
    Browser *Browser

    // Target is the target to run actions (commands) against. It is not
    // inherited from the parent context, and typically each context will
    // have its own unique Target pointing to a separate browser tab (page).
    Target *Target

    // BrowserContextID is set up by WithExistingBrowserContext.
    //
    // Otherwise, BrowserContextID holds a non-empty value in the following cases:
    //
    // 1. if the context is created with the WithNewBrowserContext option, a new
    // BrowserContext is created on its first run, and BrowserContextID holds
    // the id of that new BrowserContext;
    //
    // 2. if the context is not created with the WithTargetID option, and its
    // parent context has a non-empty BrowserContextID, this context's
    // BrowserContextID is copied from the parent context.
    BrowserContextID cdp.BrowserContextID
    // contains filtered or unexported fields
}
```

Context is attached to any context.Context which is valid for use with Run.

```
func FromContext(ctx context.Context) *Context
```

FromContext extracts the Context data stored inside a context.Context.

type ContextOption

```
type ContextOption = func(*Context)
```

ContextOption is a context option.

func WithBrowserOption

```
func WithBrowserOption(opts ...BrowserOption) ContextOption
```

WithBrowserOption allows passing a number of browser options to the allocator when allocating a new browser. As such, this context option can only be used when NewContext is allocating a new browser.

func WithDebugf

```
func WithDebugf(f func(string, ...interface{})) ContextOption
```

WithDebugf is a shortcut for WithBrowserOption(WithBrowserDebugf(f)).

func WithErrorf

```
func WithErrorf(f func(string, ...interface{})) ContextOption
```

WithErrorf is a shortcut for WithBrowserOption(WithBrowserErrorf(f)).

func WithExistingBrowserContext

```
func WithExistingBrowserContext(id cdp.BrowserContextID) ContextOption
```

func WithLogf

```
func WithLogf(f func(string, ...interface{})) ContextOption
```

WithLogf is a shortcut for WithBrowserOption(WithBrowserLogf(f)).

func WithNewBrowserContext

```
func WithNewBrowserContext(options ...CreateBrowserContextOption) ContextOption
```

WithNewBrowserContext sets up a context to create a new BrowserContext, and create a new target in this BrowserContext. A child context will create its target in this BrowserContext too, unless it's set up with other options. The new BrowserContext will be disposed when the context is done.

func WithTargetID

```
func WithTargetID(id target.ID) ContextOption
```

WithTargetID sets up a context to be attached to an existing target, instead of creating a new one.

type CreateBrowserContextOption

```
type CreateBrowserContextOption = func(*target.CreateBrowserContextParams) *target.CreateBrowserContextParams
```

CreateBrowserContextOption is a BrowserContext creation options.

type Device

```
type Device interface {
    // Device returns the device info.
    Device() device.Info
}
```

type DialOption

```
type DialOption = func(*Conn)
```

DialOption is a dial option.

func WithConnDebugf

```
func WithConnDebugf(f func(string, ...interface{})) DialOption
```

WithConnDebugf is a dial option to set a protocol logger.

type EmulateAction

```
type EmulateAction Action
```

EmulateAction are actions that change the emulation settings for the browser.

func Emulate

```
func Emulate(device Device) EmulateAction
```

Emulate is an action to emulate a specific device.

See device for a set of off-the-shelf devices and modes.

▶ Example

func EmulateReset

```
func EmulateReset() EmulateAction
```

Resets the browser's viewport, screen orientation, user-agent, and mobile/touch emulation settings to the original values the browser was started with.

func EmulateViewport

```
func EmulateViewport(width, height int64, opts ...EmulateViewportOption) EmulateAction
```

EmulateViewport is an action to change the browser viewport.

Wraps calls to emulation.SetDeviceMetricsOverride and emulation.SetTouchEmulationEnabled.

Note: this has the effect of setting/forcing the screen orientation to landscape, and will disable mobile and touch emulation by default. If this is not the desired behavior, use the emulate viewport options EmulateOrientation (or EmulateLandscape/EmulatePortrait), EmulateMobile, and EmulateTouch, respectively.

func FullScreenshot

```
func FullScreenshot(res *[]byte, quality int) EmulateAction
```

FullScreenshot takes a full screenshot with the specified image quality of the entire browser viewport.

It's supposed to act the same as the command "Capture full size screenshot" in Chrome. See the behavior notes of Screenshot for more information.

The valid range of the compression quality is [0..100]. When this value is 100, the image format is png; otherwise, the image format is jpeg.

▶ Example

func ResetViewport

```
func ResetViewport() EmulateAction
```

Note: does not modify / change the browser's emulated User-Agent, if any.

type EmulateViewportOption

```
type EmulateViewportOption = func(*emulation.SetDeviceMetricsOverrideParams, *emulation.SetTouchEmulationEnabledParams)
```

EmulateViewportOption is the type for emulate viewport options.

func EmulateOrientation

```
func EmulateOrientation(orientation emulation.OrientationType, angle int64) EmulateViewportOption
```

EmulateOrientation is an emulate viewport option to set the device viewport screen orientation.

func EmulateScale

```
func EmulateScale(scale float64) EmulateViewportOption
```

EmulateScale is an emulate viewport option to set the device viewport scaling factor.

type Error

```
type Error string
```

Error is a chromedp error.

```
const (
    // ErrInvalidWebsocketMessage is the invalid websocket message.
    ErrInvalidWebsocketMessage Error = "invalid websocket message"

    // ErrInvalidDimensions is the invalid dimensions error.
    ErrInvalidDimensions Error = "invalid dimensions"
```

```
    // ErrHasResults is the has results error.
    ErrHasResults Error = "has results"

    // ErrNotVisible is the not visible error.
    ErrNotVisible Error = "not visible"

    // ErrVisible is the visible error.
    ErrVisible Error = "visible"

    // ErrDisabled is the disabled error.
    ErrDisabled Error = "disabled"

    // ErrNotSelected is the not selected error.
    ErrNotSelected Error = "not selected"

    // ErrInvalidBoxModel is the invalid box model error.
    ErrInvalidBoxModel Error = "invalid box model"

    // ErrChannelClosed is the channel closed error.
    ErrChannelClosed Error = "channel closed"

    // ErrInvalidTarget is the invalid target error.
    ErrInvalidTarget Error = "invalid target"

    // ErrInvalidContext is the invalid context error.
    ErrInvalidContext Error = "invalid context"

    // ErrPollingTimeout is the error that the timeout reached before the pageFunction returns a truthy value.
    ErrPollingTimeout Error = "waiting for function failed: timeout"

    // ErrJSUndefined is the error that the type of RemoteObject is "undefined".
    ErrJSUndefined Error = "encountered an undefined value"

    // ErrJSNull is the error that the value of RemoteObject is null.
    ErrJSNull Error = "encountered a null value"
)
```

func (Error) Error

```
func (err Error) Error() string
```

Error satisfies the error interface.

type EvaluateAction

```
type EvaluateAction Action
```

EvaluateAction are actions that evaluate JavaScript expressions using runtime.Evaluate.

func Evaluate

```
func Evaluate(expression string, res interface{}, opts ...EvaluateOption) EvaluateAction
```

Evaluate is an action to evaluate the JavaScript expression, unmarshaling the result of the script evaluation to res.

When res is nil, the script result will be ignored.

When res is a *[]byte, the raw JSON-encoded value of the script result will be placed in res.

When res is a **runtime.RemoteObject, res will be set to the low-level protocol type, and no attempt will be made to convert the result. The original objects could be maintained in memory until the page is navigated or closed. `runtime.ReleaseObject` or `runtime.ReleaseObjectGroup` can be used to ask the browser to release the original objects.

For all other cases, the result of the script will be returned "by value" (i.e., JSON-encoded), and subsequently an attempt will be made to json.Unmarshal the script result to res. When the script result is "undefined" or "null", and the value that res points to can not be nil (only the value of a chan, func, interface, map, pointer, or slice can be nil), it returns ErrJSUndefined or ErrJSNull respectively.

▶ Example

EvaluateAsDevTools is an action that evaluates a JavaScript expression as Chrome DevTools would, evaluating the expression in the "console" context, and making the Command Line API available to the script.

See Evaluate for more information on how script expressions are evaluated.

Note: this should not be used with untrusted JavaScript.

type EvaluateOption

```
type EvaluateOption = func(*runtime.EvaluateParams) *runtime.EvaluateParams
```

EvaluateOption is the type for JavaScript evaluation options.

func EvalObjectGroup

```
func EvalObjectGroup(objectGroup string) EvaluateOption
```

EvalObjectGroup is an evaluate option to set the object group.

type ExecAllocator

```
type ExecAllocator struct {
    // contains filtered or unexported fields
}
```

ExecAllocator is an Allocator which starts new browser processes on the host machine.

▶ Example

func (*ExecAllocator) Allocate

```
func (a *ExecAllocator) Allocate(ctx context.Context, opts ...BrowserOption) (*Browser, error)
```

func (*ExecAllocator) Wait

```
func (a *ExecAllocator) Wait()
```

Wait satisfies the Allocator interface.

type ExecAllocatorOption

```
type ExecAllocatorOption = func(*ExecAllocator)
```

ExecAllocatorOption is an exec allocator option.

func CombinedOutput

```
func CombinedOutput(w io.Writer) ExecAllocatorOption
```

CombinedOutput is used to set an io.Writer where stdout and stderr from the browser will be sent

func Env

```
func Env(vars ...string) ExecAllocatorOption
```

Env is a list of generic environment variables in the form NAME=value to pass into the new Chrome process. These will be appended to the environment of the Go process as retrieved by os.Environ.

func ExecPath

```
func ExecPath(path string) ExecAllocatorOption
```

ExecPath returns an ExecAllocatorOption which uses the given path to execute browser processes. The given path can be an absolute path to a binary, or just the name of the program to find via exec.LookPath.

Flag is a generic command line option to pass a flag to Chrome. If the value is a string, it will be passed as --name=value. If it's a boolean, it will be passed as --name if value is true.

func ModifyCmdFunc

```
func ModifyCmdFunc(f func(cmd *exec.Cmd)) ExecAllocatorOption
```

ModifyCmdFunc allows for running an arbitrary function on the browser exec.Cmd object. This overrides the default version of the command which sends SIGKILL to any open browsers when the Go program exits.

func ProxyServer

```
func ProxyServer(proxy string) ExecAllocatorOption
```

ProxyServer is the command line option to set the outbound proxy server.

func UserAgent

```
func UserAgent(userAgent string) ExecAllocatorOption
```

UserAgent is the command line option to set the default User-Agent header.

func UserDataDir

```
func UserDataDir(dir string) ExecAllocatorOption
```

UserDataDir is the command line option to set the user data dir.

Note: set this option to manually set the profile directory used by Chrome. When this is not set, then a default path will be created in the /tmp directory.

func WSURLReadTimeout

WSURLReadTimeout sets the waiting time for reading the WebSocket URL. The default value is 20 seconds.

func WindowSize

```
func WindowSize(width, height int) ExecAllocatorOption
```

WindowSize is the command line option to set the initial window size.

type KeyAction

```
type KeyAction Action
```

KeyAction are keyboard (key) input event actions.

func KeyEvent

```
func KeyEvent(keys string, opts ...KeyOption) KeyAction
```

KeyEvent is a key action that synthesizes a keyDown, char, and keyUp event for each rune contained in keys along with any supplied key options.

Only well-known, "printable" characters will have char events synthesized.

See the SendKeys action to synthesize key events for a specific element node.

See the kb package for implementation details and list of well-known keys.

func KeyEventNode

```
func KeyEventNode(n *cdp.Node, keys string, opts ...KeyOption) KeyAction
```

KeyEventNode is a key action that dispatches a key event on an element node.

KeyOption is a key action option.

func KeyModifiers

```
func KeyModifiers(modifiers ...input.Modifier) KeyOption
```

KeyModifiers is a key action option to add additional modifiers on the key press.

type MouseAction

```
type MouseAction Action
```

MouseAction are mouse input event actions

func MouseClickNode

```
func MouseClickNode(n *cdp.Node, opts ...MouseOption) MouseAction
```

MouseClickNode is an action that dispatches a mouse left button click event at the center of a specified node.

Note that the window will be scrolled if the node is not within the window's viewport.

func MouseClickXY

```
func MouseClickXY(x, y float64, opts ...MouseOption) MouseAction
```

MouseClickXY is an action that sends a left mouse button click (i.e., mousePressed and mouseReleased event) to the X, Y location.

func MouseEvent

```
func MouseEvent(typ input.MouseType, x, y float64, opts ...MouseOption) MouseAction
```

MouseEvent is a mouse event action to dispatch the specified mouse event type at coordinates x, y

```
type MouseOption = func(*input.DispatchMouseEventParams) *input.DispatchMouseEventParams
```

MouseOption is a mouse action option.

func Button

```
func Button(btn string) MouseOption
```

Button is a mouse action option to set the button to click from a string.

func ButtonModifiers

```
func ButtonModifiers(modifiers ...input.Modifier) MouseOption
```

ButtonModifiers is a mouse action option to add additional input modifiers for a button click.

func ButtonType

```
func ButtonType(button input.MouseButton) MouseOption
```

ButtonType is a mouse action option to set the button to click.

func ClickCount

```
func ClickCount(n int) MouseOption
```

ClickCount is a mouse action option to set the click count.

type NavigateAction

```
type NavigateAction Action
```

func Navigate

```
func Navigate(urlstr string) NavigateAction
```

Navigate is an action that navigates the current frame.

func NavigateBack

```
func NavigateBack() NavigateAction
```

NavigateBack is an action that navigates the current frame backwards in its history.

func NavigateForward

```
func NavigateForward() NavigateAction
```

NavigateForward is an action that navigates the current frame forwards in its history.

func NavigateToHistoryEntry

```
func NavigateToHistoryEntry(entryID int64) NavigateAction
```

NavigateToHistoryEntry is an action to navigate to the specified navigation entry.

func Reload

```
func Reload() NavigateAction
```

Reload is an action that reloads the current page.

type PollAction

```
type PollAction Action
```

See Poll for details on building poll tasks.

func Poll

```
func Poll(expression string, res interface{}, opts ...PollOption) PollAction
```

Polling Options

Poll is a poll action that will wait for a general JavaScript predicate. It builds the predicate from a JavaScript expression.

This is a copy of puppeteer's page.waitForFunction. It's named Poll intentionally to avoid messing up with the Wait* query actions. The behavior is not guaranteed to be compatible. For example, our implementation makes the poll task not survive from a navigation, and an error is raised in this case (see unit test TestPoll/NotSurviveNavigation).

Polling Options

The default polling mode is "raf", to constantly execute pageFunction in requestAnimationFrame callback. This is the tightest polling mode which is suitable to observe styling changes. The WithPollingInterval option makes it to poll the predicate with a specified interval. The WithPollingMutation option makes it to poll the predicate on every DOM mutation.

The WithPollingTimeout option specifies the maximum time to wait for the predicate returns truthy value. It defaults to 30 seconds. Pass 0 to disable timeout.

The WithPollingInFrame option specifies the frame in which to evaluate the predicate. If not specified, it will be evaluated in the root page of the current tab.

The WithPollingArgs option provides extra arguments to pass to the predicate. Only apply this option when the predicate is built from a function. See PollFunction.

func PollFunction

```
func PollFunction(pageFunction string, res interface{}, opts ...PollOption) PollAction
```

PollFunction is a poll action that will wait for a general JavaScript predicate. It builds the predicate from a JavaScript function.

type PollOption

```
type PollOption = func(task *pollTask)
```

PollOption is a poll task option.

func WithPollingArgs

```
func WithPollingArgs(args ...interface{}) PollOption
```

WithPollingArgs provides extra arguments to pass to the predicate.

func WithPollingInFrame

```
func WithPollingInFrame(frame *cdp.Node) PollOption
```

WithPollingInFrame specifies the frame in which to evaluate the predicate. If not specified, it will be evaluated in the root page of the current tab.

func WithPollingInterval

```
func WithPollingInterval(interval time.Duration) PollOption
```

WithPollingInterval makes it to poll the predicate with the specified interval.

func WithPollingMutation

```
func WithPollingMutation() PollOption
```

WithPollingMutation makes it to poll the predicate on every DOM mutation.

func WithPollingTimeout

WithPollingTimeout specifies the maximum time to wait for the predicate returns truthy value. It defaults to 30 seconds. Pass 0 to disable timeout.

type PopulateOption

```
type PopulateOption = func(*time.Duration)
```

PopulateOption is an element populate action option.

func PopulateWait

```
func PopulateWait(wait time.Duration) PopulateOption
```

PopulateWait is populate option to set a wait interval after requesting child nodes.

type QueryAction

```
type QueryAction Action
```

QueryAction are element query actions that select node elements from the browser's DOM for retrieval or manipulation.

See Query for details on building element query selectors.

func AttributeValue

```
func AttributeValue(sel interface{}, name string, value *string, ok *bool, opts ...QueryOption) QueryAction
```

AttributeValue is an element query action that retrieves the element attribute value for the first element node matching the selector.

func Attributes

```
func Attributes(sel interface{}, attributes *map[string]string, opts ...QueryOption) QueryAction
```

func AttributesAll

```
func AttributesAll(sel interface{}, attributes *[]map[string]string, opts ...QueryOption) QueryAction
```

AttributesAll is an element query action that retrieves the element attributes for all element nodes matching the selector.

Note: this should be used with the ByQueryAll query option.

func Blur

```
func Blur(sel interface{}, opts ...QueryOption) QueryAction
```

Blur is an element query action that unfocuses (blurs) the first element node matching the selector.

func Clear

```
func Clear(sel interface{}, opts ...QueryOption) QueryAction
```

Clear is an element query action that clears the values of any input/textarea element nodes matching the selector.

func Click

```
func Click(sel interface{}, opts ...QueryOption) QueryAction
```

Click is an element query action that sends a mouse click event to the first element node matching the selector.

func ComputedStyle

```
func ComputedStyle(sel interface{}, style *[]*css.ComputedStyleProperty, opts ...QueryOption) QueryAction
```

ComputedStyle is an element query action that retrieves the computed style of the first element node matching the selector.

func Dimensions

Dimensions is an element query action that retrieves the box model dimensions for the first element node matching the selector.

func DoubleClick

```
func DoubleClick(sel interface{}, opts ...QueryOption) QueryAction
```

DoubleClick is an element query action that sends a mouse double click event to the first element node matching the selector.

func Dump

```
func Dump(sel interface{}, w io.Writer, opts ...QueryOption) QueryAction
```

Dump is an element query action that writes a readable tree of the first element node matching the selector and its children, up to the specified depth.

See DumpTo for more configurable options, which includes the ability to set the sleep wait timeout.

func DumpTo

```
func DumpTo(sel interface{}, w io.Writer, prefix, indent string, nodeIDs bool, depth int64, pierce bool, wait time.Duration, opt
s ...QueryOption) QueryAction
```

DumpTo is an element query action that writes a readable tree of the first element node matching the selector and its children, up to the specified depth.

See Dump for a simpler interface.

func Focus

```
func Focus(sel interface{}, opts ...QueryOption) QueryAction
```

Focus is an element query action that focuses the first element node matching the selector.

```
func InnerHTML(sel interface{}, html *string, opts ...QueryOption) QueryAction
```

InnerHTML is an element query action that retrieves the inner html of the first element node matching the selector.

func JavascriptAttribute

```
func JavascriptAttribute(sel interface{}, name string, res interface{}, opts ...QueryOption) QueryAction
```

JavascriptAttribute is an element query action that retrieves the JavaScript attribute for the first element node matching the selector.

func MatchedStyle

```
func MatchedStyle(sel interface{}, style **css.GetMatchedStylesForNodeReturns, opts ...QueryOption) QueryAction
```

MatchedStyle is an element query action that retrieves the matched style information for the first element node matching the selector.

func NodeIDs

```
func NodeIDs(sel interface{}, ids *[]cdp.NodeID, opts ...QueryOption) QueryAction
```

NodeIDs is an element query action that retrieves the element node IDs matching the selector.

func Nodes

```
func Nodes(sel interface{}, nodes *[]*cdp.Node, opts ...QueryOption) QueryAction
```

Nodes is an element query action that retrieves the document element nodes matching the selector.

func OuterHTML

```
func OuterHTML(sel interface{}, html *string, opts ...QueryOption) QueryAction
```

func Query

```
func Query(sel interface{}, opts ...QueryOption) QueryAction
```

Query Options
By Options
Node Options

Query is a query action that queries the browser for specific element node(s) matching the criteria.

Query actions that target a browser DOM element node (or nodes) make use of Query, in conjunction with the After option to retrieve data or to modify the element(s) selected by the query.

For example:

```
chromedp.Run(ctx, chromedp.SendKeys(`thing`, chromedp.ByID))
```

The above will perform a SendKeys action on the first element matching a browser CSS query for "#thing".

[Element] selection queries work in conjunction with specific actions and form the primary way of automating Tasks in the browser. They are typically written in the following form:

```
Action(selector[, parameter1, ...parameterN][,result][, queryOptions...])
```

Where:

- Action - the action to perform
- selector - element query selection (typically a string), that any matching node(s) will have the action applied
- parameter[1-N] - parameter(s) needed for the individual action (if any)
- result - pointer to a result (if any)
- queryOptions - changes how queries are executed, or how nodes are waited for

Query Options

Node* options specify node conditions that cause the query to wait until the specified condition is true. When not specified, queries will use the NodeReady wait condition.

The AtLeast option alters the minimum number of nodes that must be returned by the element query. If not specified, the default value is 1.

The After option is used to specify a func that will be executed when element query has returned one or more elements, and after the node condition is true.

By Options

The BySearch (default) option enables querying for elements by plain text, CSS selector or XPath query, wrapping DOM.performSearch.

The ByID option enables querying for a single element with the matching CSS ID, wrapping DOM.querySelector. ByID is similar to calling document.querySelector('#' + ID) from within the browser.

The ByQuery option enables querying for a single element using a CSS selector, wrapping DOM.querySelector. ByQuery is similar to calling document.querySelector() from within the browser.

The ByQueryAll option enables querying for elements using a CSS selector, wrapping DOM.querySelectorAll. ByQueryAll is similar to calling document.querySelectorAll() from within the browser.

The ByJSPath option enables querying for a single element using its "JS Path" value, wrapping Runtime.evaluate. ByJSPath is similar to executing a JavaScript snippet that returns an element from within the browser. ByJSPath should be used only with trusted element queries, as it is passed directly to Runtime.evaluate, and no attempt is made to sanitize the query. Useful for querying DOM elements that cannot be retrieved using other By* funcs, such as ShadowDOM elements.

Node Options

The NodeReady (default) option causes the query to wait until all element nodes matching the selector have been retrieved from the browser.

The NodeVisible option causes the query to wait until all element nodes matching the selector have been retrieved from the browser, and are visible.

The NodeEnabled option causes the query to wait until all element nodes matching the selector have been retrieved from the browser, and are enabled (i.e., do not have a 'disabled' attribute).

The NodeSelected option causes the query to wait until all element nodes matching the selector have been retrieved from the browser, and are selected (i.e., has a 'selected' attribute).

The NodeNotPresent option causes the query to wait until there are no element nodes matching the selector.

func QueryAfter

```
func QueryAfter(sel interface{}, f func(context.Context, runtime.ExecutionContextID, ...*cdp.Node) error, opts ...QueryOption) QueryAction
```

QueryAfter is an element query action that queries the browser for selector sel. Waits until the visibility conditions of the query have been met, after which executes f.

func RemoveAttribute

```
func RemoveAttribute(sel interface{}, name string, opts ...QueryOption) QueryAction
```

RemoveAttribute is an element query action that removes the element attribute with name from the first element node matching the selector.

func Reset

```
func Reset(sel interface{}, opts ...QueryOption) QueryAction
```

Reset is an element query action that resets the parent form of the first element node matching the selector.

func Screenshot

```
func Screenshot(sel interface{}, picbuf *[]byte, opts ...QueryOption) QueryAction
```

Behavior notes: the Protocol Monitor shows that the command sends the following CDP commands too:

- Emulation.clearDeviceMetricsOverride
- Network.setUserAgentOverride with {"userAgent": ""}
- Overlay.setShowViewportSizeOnResize with {"show": false}

These CDP commands are not sent by chromedp. If it does not work as expected, you can try to send those commands yourself.

See CaptureScreenshot for capturing a screenshot of the browser viewport.

See screenshot for an example of taking a screenshot of the entire page.

func ScreenshotScale

```
func ScreenshotScale(sel interface{}, scale float64, picbuf *[]byte, opts ...QueryOption) QueryAction
```

ScreenshotScale is like Screenshot but accepts a scale parameter that specifies the page scale factor.

func ScrollIntoView

```
func ScrollIntoView(sel interface{}, opts ...QueryOption) QueryAction
```

ScrollIntoView is an element query action that scrolls the window to the first element node matching the selector.

func SendKeys

```
func SendKeys(sel interface{}, v string, opts ...QueryOption) QueryAction
```

SendKeys is an element query action that synthesizes the key up, char, and down events as needed for the runes in v, sending them to the first element node matching the selector.

See the keys for a complete example on how to use SendKeys.

Note: when the element query matches an input[type="file"] node, then dom.SetFileInputFiles is used to set the upload path of the

func SetAttributeValue

```
func SetAttributeValue(sel interface{}, name, value string, opts ...QueryOption) QueryAction
```

SetAttributeValue is an element query action that sets the element attribute with name to value for the first element node matching the selector.

func SetAttributes

```
func SetAttributes(sel interface{}, attributes map[string]string, opts ...QueryOption) QueryAction
```

SetAttributes is an element query action that sets the element attributes for the first element node matching the selector.

func SetJavascriptAttribute

```
func SetJavascriptAttribute(sel interface{}, name, value string, opts ...QueryOption) QueryAction
```

SetJavascriptAttribute is an element query action that sets the JavaScript attribute for the first element node matching the selector.

func SetUploadFiles

```
func SetUploadFiles(sel interface{}, files []string, opts ...QueryOption) QueryAction
```

SetUploadFiles is an element query action that sets the files to upload (i.e., for a input[type="file"] node) for the first element node matching the selector.

func SetValue

```
func SetValue(sel interface{}, value string, opts ...QueryOption) QueryAction
```

SetValue is an element query action that sets the JavaScript value of the first element node matching the selector.

Useful for setting an element's JavaScript value, namely form, input, textarea, select, or other element with a 'value' field.

```
func Submit(sel interface{}, opts ...QueryOption) QueryAction
```

Submit is an element query action that submits the parent form of the first element node matching the selector.

func Text

```
func Text(sel interface{}, text *string, opts ...QueryOption) QueryAction
```

Text is an element query action that retrieves the visible text of the first element node matching the selector.

func TextContent

```
func TextContent(sel interface{}, text *string, opts ...QueryOption) QueryAction
```

TextContent is an element query action that retrieves the text content of the first element node matching the selector.

func Value

```
func Value(sel interface{}, value *string, opts ...QueryOption) QueryAction
```

Value is an element query action that retrieves the JavaScript value field of the first element node matching the selector.

Useful for retrieving an element's JavaScript value, namely form, input, textarea, select, or any other element with a '.value' field.

func WaitEnabled

```
func WaitEnabled(sel interface{}, opts ...QueryOption) QueryAction
```

WaitEnabled is an element query action that waits until the element matching the selector is enabled (i.e., does not have attribute 'disabled').

func WaitNotPresent

WaitNotPresent is an element query action that waits until no elements are present matching the selector.

func WaitNotVisible

```
func WaitNotVisible(sel interface{}, opts ...QueryOption) QueryAction
```

WaitNotVisible is an element query action that waits until the element matching the selector is not visible.

func WaitReady

```
func WaitReady(sel interface{}, opts ...QueryOption) QueryAction
```

WaitReady is an element query action that waits until the element matching the selector is ready (i.e., has been "loaded").

func WaitSelected

```
func WaitSelected(sel interface{}, opts ...QueryOption) QueryAction
```

WaitSelected is an element query action that waits until the element matching the selector is selected (i.e., has attribute 'selected').

func WaitVisible

```
func WaitVisible(sel interface{}, opts ...QueryOption) QueryAction
```

WaitVisible is an element query action that waits until the element matching the selector is visible.

type QueryOption

```
type QueryOption = func(*Selector)
```

QueryOption is an element query action option.

After is an element query option that sets a func to execute after the matched nodes have been returned by the browser, and after the node condition is true.

func AtLeast

```
func AtLeast(n int) QueryOption
```

AtLeast is an element query option to set a minimum number of elements that must be returned by the query.

By default, a query will have a value of 1.

func ByFunc

```
func ByFunc(f func(context.Context, *cdp.Node) ([]cdp.NodeID, error)) QueryOption
```

ByFunc is an element query action option to set the func used to select elements.

func FromNode

```
func FromNode(node *cdp.Node) QueryOption
```

FromNode is an element query action option where a query will be run. That is, the query will only look at the node's element sub-tree. By default, or when passed nil, the document's root element will be used.

Note that, at present, BySearch and ByJSPath do not support FromNode; this option is mainly useful for ByQuery selectors.

▶ Example

func Populate

```
func Populate(depth int64, pierce bool, opts ...PopulateOption) QueryOption
```

NOTE: this could be extremely resource intensive. Avoid doing this unless necessary.

func RetryInterval

```
func RetryInterval(interval time.Duration) QueryOption
```

RetryInterval is an element query action option to set the retry interval to specify how often it should retry when it failed to select the target element(s).

The default value is 5ms.

func WaitFunc

```
func WaitFunc(wait func(context.Context, *cdp.Frame, runtime.ExecutionContextID, ...cdp.NodeID) ([]*cdp.Node, error)) QueryOption
```

WaitFunc is an element query option to set a custom node condition wait.

type RemoteAllocator

```
type RemoteAllocator struct {
    // contains filtered or unexported fields
}
```

RemoteAllocator is an Allocator which connects to an already running Chrome process via a websocket URL.

func (*RemoteAllocator) Allocate

```
func (a *RemoteAllocator) Allocate(ctx context.Context, opts ...BrowserOption) (*Browser, error)
```

Allocate satisfies the Allocator interface.

func (*RemoteAllocator) Wait

Wait satisfies the Allocator interface.

type RemoteAllocatorOption

```
type RemoteAllocatorOption = func(*RemoteAllocator)
```

RemoteAllocatorOption is a remote allocator option.

type Selector

```
type Selector struct {
    // contains filtered or unexported fields
}
```

Selector holds information pertaining to an element selection query.

See Query for information on building an element selector and relevant options.

func (*Selector) Do

```
func (s *Selector) Do(ctx context.Context) error
```

Do executes the selector, only finishing if the selector's by, wait, and after funcs succeed, or if the context is cancelled.

type Target

```
type Target struct {
    SessionID target.SessionID
    TargetID  target.ID
    // contains filtered or unexported fields
}
```

Target manages a Chrome DevTools Protocol target.

```
func (t *Target) Execute(ctx context.Context, method string, params easyjson.Marshaler, res easyjson.Unmarshaler) error
```

type Tasks

```
type Tasks []Action
```

Tasks is a sequential list of Actions that can be used as a single Action.

func (Tasks) Do

```
func (t Tasks) Do(ctx context.Context) error
```

Do executes the list of Actions sequentially, using the provided context and frame handler.

type Transport

```
type Transport interface {
    Read(context.Context, *cdproto.Message) error
    Write(context.Context, *cdproto.Message) error
    io.Closer
}
```

Transport is the common interface to send/receive messages to a target.

This interface is currently used internally by Browser, but it is exposed as it will be useful as part of the public API in the future.

Source Files

View all Source files

allocate.go
allocate_linux.go
browser.go
conn.go
emulate.go
errors.go
js.go
nav.go
poll.go
target.go
util.go

Directories

device	Package device contains device emulation definitions for use with chromedp's Emulate action.
kb	Package kb provides keyboard mappings for Chrome DOM Keys for use with input events.