

Go module

The Margo root module turns Markdown into an immutable compiled document. Your application chooses the output boundary and keeps ownership of its URLs, navigation, assets, storage, and deployment.

A small compiling example

This complete program compiles and renders one source. A host can replace the standalone page with its own HTML composition or site frame.

```
package main

import (
    "context"
    "log"

    margo "github.com/araihu/margo"
)

func main() {
    compiler := margo.New()
    document, err := compiler.Compile(context.Background(), margo.Source{
        Name:     "guide.md",
        Content: []byte("# Guide\n\nHello from Margo.\n"),
    })
    if err != nil {
        log.Fatal(err)
    }
    rendered, err := compiler.Render(context.Background(), document)
    if err != nil {
        log.Fatal(err)
    }
    page, err := margo.RenderStandalone(rendered)
    if err != nil {
        log.Fatal(err)
    }
}
```

```
_ = page // Render the component in the host-owned HTTP response.  
}
```

Compiler and render lifecycle

Compile parses Markdown, normalizes closed metadata, evaluates the host policy, and freezes the source into an immutable document. **Render** projects that document for a target and exposes semantic HTML plus dependency requirements. Reuse one compiler across requests; it is safe for concurrent compile and render calls. Pass a canceled context to stop work at the API boundary.

Public package map

Package ↕	Responsibility ↕
github.com/araihu/margo	New, Compile, Render, metadata, and HTML projections
github.com/araihu/margo/charts	Optional static SVG and interactive chart extension
github.com/araihu/margo/deck	Experimental HTML/PDF presentation projection
github.com/araihu/margo/pdf	Renderer-neutral PDF request and engine contracts
github.com/araihu/margo/site	Linked-site build, config, routes, and publication artifacts
github.com/araihu/margo/ssg	Layout-neutral frame, shell, schema, and binding contracts

Use only the packages your host needs. The **margo** executable is a separate command surface built on these same boundaries.

For exported symbols, use the [root Go API reference](#) and its [site](#), [PDF](#), [Chromium](#), [deck](#), and [charts](#) package pages. These links intentionally pin the root release so the historical nested `github.com/araihu/margo/pdf` module is not selected by accident.

Host ownership boundaries

Margo owns Markdown semantics, diagnostics, target projection, and declared dependency requirements. The host owns page shells, navigation, canonical URLs, asset staging, authentication, persistence, and publication. `RenderHTML` returns a semantic fragment; `RenderHTMLPage` supplies a generic page shape when that is useful, while a configured site may compose its own frame.

This separation lets a Go service adopt the compiler without importing a particular visual shell or deployment system.

Extensions and policy

Optional capabilities are registered by the host. For example, chart support is explicit:

```
import (
    margo "github.com/araihu/margo"
    "github.com/araihu/margo/charts"
)

compiler := margo.New(
    margo.WithExtension(charts.Extension()),
)
```

A host policy controls privileged raw HTML and iframe behavior. Frontmatter can carry document preferences and site publication actions, but it cannot grant itself capabilities. Check documents before rendering when CI needs structured diagnostics.

Select the projection

The compiler API intentionally stops before filesystem publication or browser process management. Use the package that owns the output boundary:

Projection ↕	API path ↕	Result ↕
Standalone HTML	margo.RenderStandalone	An offline templ.Component that the host renders to its response or file
Host-composed HTML	margo.RenderHTML → margo.RenderHTMLPage	A semantic fragment plus explicit dependency requirements
Linked site	site.Build / site.BuildConfig	Sorted site.Result.Artifacts; the caller writes and deploys them
PDF	pdf/chromium.New → pdf.Engine.Export	A tagged PDF from one explicitly selected installed Chromium
Deck	deck.Render → deck.Result.HTML or a PDF engine	The versioned Margo deck profile with static chart projection

For PDF and deck PDF, the host must carry the rendered runtime descriptor into `pdf.Request` and choose a stable, non-empty `ExecutionID`. This is deliberate: the engine validates the browser runtime report against the exact document and render instance rather than accepting arbitrary HTML. The CLI wraps these steps; use it when the application does not need to own them.

Documentation chapters

The repository and published site now split the contract by task:

1. CLI workflows — installation, streams, diagnostics, and command selection.
2. Site builds — `site.yaml`, routes, metadata, themes, and publication artifacts.
3. PDF output — engines, links, geometry, and corporate branding.
4. Deck output — themes, directives, compositions, charts, and overflow validation.

5. Schemas — versioned configuration, output, and runtime contracts rendered as property trees.
6. Policies and security — host authority, raw HTML, iframe projections, and exact schemas.
7. Fenced types — diagrams, charts, schema trees, and highlighted code blocks.

For the full exported API, use `go doc github.com/araihu/margo` and the package-specific docs for `site`, `pdf`, `pdf/chromium`, `deck`, and `charts`. Keep RFC 3339 metadata values quoted in YAML (`publishedAt: "2026-08-25T12:00:00Z"`) so the parser receives a string rather than a YAML timestamp node.

Dependencies and upstream boundaries

Margo composes upstream projects behind a small, versioned contract. These are the projects a newcomer is most likely to encounter while reading the source repository or Go package imports:

Upstream ↕	Margo uses it for ↕	Boundary in Margo ↕
<u>Goldmark</u>	CommonMark parsing and fenced extensions	Margo normalizes the semantic document and closes its metadata namespace.
<u>Goshtoso</u>	Accessible Go/Templ components, themes, tokens, and page shells	Hosts own composition and delivery; Margo selects only its documented shells and assets.
<u>Goshtoso Charts</u>	Optional chart fences and exact-data tables	Register charts.Extension() explicitly; deck output is static by contract.

Upstream ↕	Margo uses it for ↕	Boundary in Margo ↕
<u>Muamba</u>	Build-time materialization and provenance for local assets	Runtime assets are embedded/locked; Margo never downloads them.
<u>Mermaid</u>	Diagram runtime for Mermaid fences	Margo vendors a known runtime and sanitizes the SVG projection.
<u>templ</u>	Typed component rendering	Margo exposes semantic fragments and dependency requirements, not a server.
<u>Chromedp</u> / Chromium	Browser validation and PDF export	The host selects an installed executable; no browser download or fallback.
<u>Marpit</u>	Vocabulary and layout inspiration for decks	Margo implements a versioned profile, not universal Marpit compatibility.
<u>Dagger</u>	Portable CI adapters and artifact checks	Development/CI tooling only; it is not a runtime dependency.

Dependency versions are pinned in `go.mod` for the current release line. An upstream release changes Margo only through an intentional dependency or profile update followed by the compatibility and browser gates. For the source-level rationale, read the [unified-module decision](#).