

Guidedog マニュアル

マニュアルと完全版 Odin API リファレンス

01 / ソース
意味を伝える。

02 / 構造
関係を明確にする。

03 / 公開
画面でも紙でも読む。

目次

チュートリアル	3
はじめてのプロジェクト	3
小さな本を作る	4
トピックガイド	7
文書を書く	7
ドメイン	12
プロジェクトの仕組み	16
数学と証明	18
図で説明する	19
読みやすい組版	20
テンプレート	21
画像、動画、音声、ダウンロード	28
国際化	29
他のプロジェクトへのリンク	32
オブジェクト型の宣言	33
autodoc による Python のドキュメント作成	34
Odin の文書化	37
実践的な手順	43
Guidedog のビルドとインストール	43
文書のビルドと公開	44
Sphinx プロジェクトの移行	45
コードベースの文書化	45
この文書の保守	46
リファレンス	49
コマンドラインリファレンス	49
設定リファレンス	53
リーダー、レンダラー、ビルダー	60
エラーメッセージ	61
Sphinx との互換性	64
内部の仕組み	69
Guidedoc エンジン	69
アルゴリズムより先に不変条件	70
完全な世代を公開する	71
メモリと所有権	73
このマニュアルについて	75
ビルドの検証記録	75
Guidedog	77
プロジェクトを始める	77

インストール	77
ソースからビルドする	78
Guidedog で翻訳する	78
読む・組み込む	79
謝辞とライセンス	79
API Reference	81
autodoc	82
cli	86
core	88
defaults	142
fetch	143
gds	145
gettext	151
graphdog	167
highlight	175
highlight/guidedoc	195
host	195
i18n	222
inventory	231
jinja	234
odindoc	248
project	256
pyscan	280
readers/commonmark	295
readers/rst	307
readers/rst/inline	325
readers/sphinx	330
render/html	337
render/text	338
render/typst	338
treесitter	339
typst	345
索引	351

一度書き、分かりやすく説明し、ウェブと紙に届ける。

Guidedog は reStructuredText と Markdown のソースから HTML サイトと PDF の本を作る文書生成ツールです。Python 環境や外部の LaTeX は不要です。変換エンジンは **Guidedoc** と呼ばれ、PDF の組版には Typst を使います。サイトは Guidedog が直接生成します。

小さなプロジェクトに必要なのは、ソース文書、目次、`conf.toml` の 3 つです。はじめてのプロジェクトから始めてください。作業別の手順は実践的な手順、正確な規約は [リファレンス](#) にあります。

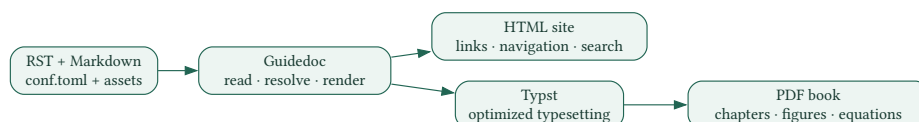


図 1 1つのソースプロジェクトから、2つの出版形式へ。

チュートリアル

順番に進めてください。各ステップで確認できる結果が得られます。

はじめてのプロジェクト

目標は、一ページをブラウザーと PDF で読むことです。Guidedog を用意してください。PDF には内蔵コンパイラー、または Typst 0.15.1 以降を使えます。Guidedog のビルドとインストールを参照してください。

macOS では、`brew install insanai/guidedog/guidedog` でリリースをインストールします。公式 tap が追加され、PDF コンパイラも含まれます。

プロジェクトの作成

`notes` がまだないディレクトリで実行します。

```
guidedog quickstart notes -q -p "Field Notes" -a "Your name" -v 1.0
```

Guidedog は設定、最初の RST 文書、編集できるテンプレートを書きます。画像・音声・動画のディレクトリも作成します。既存のプロジェクトは上書きしません。

```
notes/  
  conf.toml  
  index.rst  
  _templates/  
    layout.html  
    book.typ  
  _static/  
    guidedog.css  
    guidedog.js  
  assets/  
    images/  
    audio/  
    video/  
  _build/
```

RST ファイルは内容を、テンプレートは表示を持ちます。`quickstart` は `_templates/layout.html` と `_templates/book.typ` をプロジェクトに直接配置します。外部のテーマやキャッシュには隠されず、いつでも確認して編集できます。

最初のページを書く

`notes/index.rst` を次に置き換えます。

```
Field Notes
=====

A useful explanation begins with a small example.

Measuring a square
-----

If the side has length :math:`s`, its area is :math:`s^2`.

.. math::

    A = s^2

.. note::

    State the units before comparing measurements.
```

ビルドと確認

```
guidedog build html notes -W
guidedog build pdf notes -W
```

`notes/_build/html/index.html` と `notes/_build/pdf` 内の PDF を開きます。-w は警告でビルドを失敗させるため、文書の検査に残してください。HTML はナビゲーションと本文を、PDF は本の柱やページ番号も含みます。数式は形式ごとに組版します。設定は [数学と証明](#) を参照してください。

変更してみる

```
guidedog serve notes
```

表示されたアドレスを開き、ソースを編集して保存し、ブラウザを更新します。Guidedog は HTML を返す前にソースの更新時刻を調べ、進んでいれば再ビルドします。ブラウザの再読み込みは自動送信しません。Ctrl+C で停止します。

何も編集せずに再ビルドします。変更のない文書は再利用し、出力が最新であることを確かめるために全ページを読み直す必要はありません。次は [小さな本を作る](#) です。

小さな本を作る

サイトは文書のグラフです。本には読む順序が必要です。toctree がその両方を与えます。

章の追加

`notes/measurement.rst` を作成します。

```
Measurement
=====

A measurement has a value and a unit.
Write both. A bare number leaves the reader guessing.
```



```
.. _measurement-units:

Units
-----

Keep one unit within a comparison.
```

notes/index.rst の内容を次に置き換えます。

```
Field Notes
=====

These notes explain how to make a comparison that another person can check.

.. toctree::
   :maxdepth: 2
   :numbered:

   measurement

Begin with :doc:`measurement`.
For the unit rule, see :ref:`measurement-units`.
```

文書名には .rst を付けません。ラベルは文書内の考えに名前を付けます。章には文書リンク、特定の議論にはラベルを使います。

図の追加

measurement.rst に次を追加します。

```
.. graphviz::
   :caption: A comparison needs a common unit.
   :alt: Values are converted to one unit before they are compared.

   digraph comparison {
       rankdir=LR;
       "values" -> "common unit" -> "comparison";
   }
```

Guidedog はリンクした Graphviz ライブラリを使います。HTML には SVG、PDF には画像を出力し、ブラウザの図解インタプリタは不要です。キャプションは結論、代替テキストは図の内容を説明します。

本のビルド

```
guidedog build html notes -W
guidedog build pdf notes -W
```

章がサイドバーと本の目次に表示されます。ブラウザで節のリンクをたどり、PDF でも移動先を確認してください。

次は **文書を書く** が役立ちます。1 章で 1 つの主題を説明できる長さにしてください。

トピックガイド

この各章ではモデルを説明し、判断に必要な背景を示す。

文書を書く

Guidedog は reStructuredText (`.rst`) と MyST Markdown (`.md`) に対応します。共通の意味構造を持つ文書ツリーに解析し、HTML サイトと PDF の本を生成します。

ソース形式の選択

ドメイン、複雑な表、拡張可能なディレクティブには reStructuredText を使います。Markdown の構文で Sphinx 形式のディレクティブやロールも使いたい場合は MyST Markdown を選びます。

プロジェクト内の Markdown は MyST として読みます。単独変換の `guidedog convert` は、MyST を明示しなければ CommonMark を使います。

見出しと構造

文書のタイトルはページを示し、見出しは本文を区切ります。RST の見出しには下線を使い、同じ上線も付けられます。線は文字列以上の長さになります。

推奨する見出しの記号:

```
Document Title
=====

Section Heading
-----

Subsection
~~~~~

Sub-subsection
^^^^^^^^^^^^

Paragraph Heading
^^^^^^^^^^^^^^^^
```

プロジェクト全体で見出しの階層をそろえます。MyST Markdown では `#`、`##`、`###` を使います。

文字装飾とインライン記法

インライン記法は装飾ではなく、意味を示すために使います。

表 1 インライン記法

書式	reStructuredText	MyST Markdown
太字	<code>**strong text**</code>	<code>**strong text**</code>
強調・斜体	<code>*emphasized text*</code>	<code>*emphasized text*</code>
インラインコード	<code>`inline code`</code>	<code>`inline code`</code>
下付き文字	<code>:sub:`text`</code>	<code>{sub}`text`</code>
上付き文字	<code>:sup:`text`</code>	<code>{sup}`text`</code>

アスタリスクやバッククォートをそのまま書くには、`\*not italic\*` のようにバックスラッシュでエスケープします。

リスト

Guidedog は四種類のリストに対応します。

箇条書き

```
* First item
* Second item with multiple lines
  of continuing explanation.
* Third item
```

番号付きリスト

```
1. Numbered item
2. Second item
#. Automatically numbered item
#. Next auto-numbered item
```

定義リスト

定義リストは用語と説明を組にします。用語を一行で書き、すぐ次に字下げした定義を置きます。

```
Workspace
  A bounded memory buffer supplied by the caller for document conversion.

Session
  A host-level coordinator that manages memory budgets and document caches.
```

フィールドリスト

フィールドリストは構造化したメタデータや引数の説明に使います。

```
:Authors: Jane Doe, John Smith
:Version: 1.2
:Status: Active
```

コードブロック

`code-block` または `sourcecode` で構文を強調したコードを表示します。Guidedog は 35 種類以上の言語に対応します。

```

.. code-block:: python
:linenos:
:caption: Server entry point
:emphasize-lines: 2, 4-5

def main():
    app = create_app()
    app.run(host="0.0.0.0", port=8080)

```

コードブロックのオプション:

- `:linenos:`: コードの横に行番号を表示します。
- `:caption: Title:` コードの上下にキャプションを付けます。
- `:emphasize-lines: 1, 3-5:` 指定した行を強調します。
- `:name: label:` コードに参照用のラベルを付けます。

外部のソースをそのまま取り込むには `literalinclude` を使います。

```

.. literalinclude:: ../src/server.py
:language: python
:lines: 1-25
:linenos:

```

注記ブロック

注記ブロックは注意事項、警告、補足を目立たせます。

```

.. note::
    Helpful background context or implementation detail.

.. tip::
    Suggested best practices or workflow shortcuts.

.. important::
    Essential requirements that must not be overlooked.

.. warning::
    Conditions that could lead to unexpected behavior or lost work.

.. caution::
    Potential pitfalls or sensitive operational steps.

.. seealso::
    References to related chapters, specifications, or external guides.

```

`danger`、`error`、`hint`、`attention` も使えます。独自のタイトルには `admonition` を使います。

```

.. admonition:: Design Rationale

    Explains why a particular architecture was chosen.

```

表

Guidedog はシンプルテーブル、グリッドテーブル、ディレクティブによる表に対応します。

シンプルテーブル

シンプルテーブルは横線で列の範囲を示します。

```

=====
A      B      A and B
=====

```

```
False False False
True  False False
True  True  True
=====
```

グリッドテーブル

グリッドテーブルは複数行のセルや、行・列にまたがるセルを扱えます。

```
+-----+-----+-----+
| Header row, column 1 | Column 2 | Column 3 |
+=====+=====+=====+
| Cell with multiple   | Second   | Third     |
| paragraphs of text.  | column   | column    |
+-----+-----+-----+
```

リストテーブル

`list-table` は入れ子の箇条書きから表を作ります。幅の広い表もソース上で読みやすく、変更を管理しやすくなります。

```
.. list-table:: Project configurations
   :widths: 25 25 50
   :header-rows: 1

   * - Target
     - Builder
     - Description
   * - Website
     - ``html``
     - Static HTML documentation site
   * - Book
     - ``pdf``
     - Typeset PDF book powered by Typst
```

CSV テーブル

`csv-table` はカンマ区切りのデータから表を作ります。

```
.. csv-table:: Comparative metrics
   :header: "Name", "Time (ms)", "Memory (MB)"
   :widths: 40, 30, 30

   "Cold build", 42, 12
   "Incremental", 3, 4
```

画像と図

`image` と `figure` で画像、スクリーンショット、図を挿入します。

```
.. image:: /assets/architecture.png
   :width: 600px
   :align: center
   :alt: System architecture diagram

.. figure:: /assets/flow.png
   :scale: 80%
   :align: center
   :alt: Execution flowchart

   Data flows sequentially through reader, resolver, and renderer.
```

`figure` は画像にキャプションを付け、参照用のラベルも指定できます。

目次ツリーで構成する

`toctree` は文書の階層、ナビゲーション、HTML と PDF の読み順を定義します。

```
.. toctree::
   :maxdepth: 2
   :caption: User Guide
   :numbered:

   installation
   quickstart
   configuration
```

`toctree` の主なオプション:

- `:maxdepth: N`: 目次に含める見出しの深さ。
- `:caption: Title`: ナビゲーション項目の上に表示する分類名。
- `:numbered::` 章と見出しに番号を付けます。
- `:titlesonly::` 文書のタイトルだけを並べ、内部の見出しは省きます。
- `:hidden::` 本文内にリストを表示せず、読み順だけを記録します。
- `:glob:: tutorials/*` のようなワイルドカードで文書を選べます。

相互参照とリンク

明示的なラベルは、ファイル名とは独立した安定した参照を作ります。

ラベルと `:ref:`

見出し、表、図の直前にラベルを置きます。

```
.. _storage-model:

Storage model
-----

Caller storage is bounded and measured upfront.
```

プロジェクト内のどの文書からも参照できます。

```
See :ref:`storage-model` for details.
See :ref:`custom link text <storage-model>`.
```

`:doc:` による文書参照

拡張子を省いたパスで別の文書へリンクします。

```
Read the :doc:`configuration guide <../reference/configuration>` for details.
```

`:term:` による用語参照

`glossary` で用語を定義します。

```
.. glossary::

   Workspace
       A fixed memory arena allocated by the caller for conversion passes.

   Pass
       An in-memory transformation step operating on the document AST.
```

`:term:` で用語へリンクします。

```
Conversion runs within an allocated :term:`workspace`.
```

`:download:` によるファイル配布

ダウンロード用ディレクトリにファイルをコピーし、リンクを作ります。

```
Download the :download: starter template <files/starter.toml>.
```

外部リンク

外部の Web アドレスへリンクします。

```
Visit `Typst` <https://typst.app> for typography details.
```

置換と取り込み

再利用する語句、記号、画像を定義します。

```
.. |version| replace:: 1.0.0
.. |brand| replace:: Field Notes

Welcome to |brand|. version |version|.
```

複数のファイルで reStructuredText の内容を共有します。

```
.. include:: ../shared/warnings.rst
```

MyST Markdown を書く

MyST Markdown はフェンス付きのブロックで、同じディレクティブやロールを表現します。

```
# Project overview

Here is a paragraph with bold text, italic emphasis, and `inline code`.

```{note}
This note is rendered identically to a reStructuredText note directive.
```

```{code-block} python
:caption: Example function
:linenos:

def add(a, b):
 return a + b
```

```{toctree}
:maxdepth: 2
:caption: Navigation

first-chapter
second-chapter
```

See {ref}`storage-model` or consult the {doc}`../reference/configuration`.
```

ドメイン

ドメインは、特定の言語や分野のオブジェクトを記述し、相互参照するためのディレクティブとロールの集まりです。

Guidedog には、次の Sphinx 形式のドメインがネイティブ実装されています。

- **標準ドメイン** (`std`) : コマンドラインオプション、環境変数、用語、UI 表記など。
- **Python ドメイン** (`py`) : モジュール、クラス、関数、メソッド、属性、プロパティ、例外。
- **C ドメイン** (`c`) : 関数、型、構造体、共用体、列挙型、列挙子、変数、マクロ。
- **C++ ドメイン** (`cpp`) : クラス、コンセプト、テンプレート、名前空間、メソッド、式。
- **JavaScript ドメイン** (`js`) : 関数、メソッド、クラス、プロパティ。
- **Odin ドメイン** (`odin`) : Odin のパッケージ、手続き、構造体、共用体、列挙型。Odin の文書化 を参照してください。

主ドメインの設定

既定のドメインは `conf.toml` の `primary_domain` で設定します。初期値は `"py"` です。指定後は接頭辞を省略でき、`.. py:function::` を `.. function::`、`:py:func:`name`` を `:func:`name`` と書けます。

文書内だけでドメインを切り替えるには、次のようにします。

```
.. default-domain:: c
```

標準ドメイン

標準ドメインはコマンドラインツール、環境変数、文書参照を扱います。

ディレクティブ

```
.. program:: mytool

.. option:: -c <config>, --config <config>

    Path to the configuration file.

.. envvar:: GUIDEDOG_THEME

    Specifies the default theme stylesheet.
```

ロール

- `:doc:`path`` : 相対パスでプロジェクト内の文書へリンクします。
- `:ref:`label`` : 明示的なラベル `.. _label:` にリンクします。
- `:term:`term`` : `glossary` で定義した用語へリンクします。
- `:option:`--config`` : `option` で記述したコマンドラインオプションにリンクします。
- `:envvar:`VARIABLE`` : `envvar` で記述した環境変数にリンクします。
- `:command:`name`` : システムコマンド名を示します。
- `:file:`path`` : ファイルやディレクトリのパスを示します。`{variable}` も使えます。
- `:kbd:`Ctrl+C`` : キー操作を示します。
- `:menuselection:`File --> Save As`` : メニューの操作順を示します。

- `:guilabel:`Submit``: ボタンや UI 要素を示します。
- `:pep:`8``: Python Enhancement Proposal にリンクします。
- `:rfc:`7231``: IETF の RFC にリンクします。

Python ドメイン

Python ドメインは Python のインターフェースとシグネチャを記述します。

ディレクティブ

```
.. py:module:: pipeline.reader
   :synopsis: Document reading and normalization.

.. py:class:: DocumentReader(source_path: str, encoding: str = "utf-8")

   Base class for document readers.

   .. py:method:: parse(content: bytes) -> Document

       Parses raw bytes into a document tree.

       :param content: Raw source content bytes.
       :return: Parsed document tree instance.
       :raises ValueError: If the source format is unrecognized.

   .. py:attribute:: encoding
       :type: str

       The character encoding used for text decoding.
```

対応するディレクティブ: `py:module`、`py:currentmodule`、`py:function`、`py:data`、`py:class`、`py:method`、`py:staticmethod`、`py:classmethod`、`py:attribute`、`py:property`、`py:exception`、`py:decorator`、`py:type`。

ロール

- `:py:func:`name``: Python 関数にリンクします。
- `:py:class:`name``: Python クラスにリンクします。
- `:py:meth:`name``: Python のクラスメソッドまたはインスタンスメソッドにリンクします。
- `:py:attr:`name``: Python 属性にリンクします。
- `:py:data:`name``: Python のモジュールレベルのデータにリンクします。
- `:py:exc:`name``: Python の例外クラスにリンクします。
- `:py:mod:`name``: Python モジュールにリンクします。

C ドメイン

C ドメインは C のライブラリとヘッダーファイルを記述します。

ディレクティブ

```
.. c:namespace:: gd

.. c:struct:: Workspace

   A contiguous memory buffer for in-memory document parsing.

   .. c:member:: size_t capacity

       Total byte size of the storage slab.
```

```
.. c:function:: int gd_convert(Workspace *ws, const char *input, char *output,
size_t out_len)
```

Converts source text into HTML.

```
:param ws: Pointer to the initialized workspace.
:param input: Null-terminated input string.
:param output: Destination buffer.
:param out_len: Size of destination buffer.
:returns: 0 on success, or an error code.
```

ルール

- `:c:func:`name``: C 関数にリンクします。
- `:c:member:`name``: C の構造体または共用体のメンバーにリンクします。
- `:c:data:`name`` / `:c:var:`name``: C 変数にリンクします。
- `:c:type:`name``: C の typedef または型にリンクします。
- `:c:struct:`name``: C 構造体にリンクします。
- `:c:union:`name``: C 共用体にリンクします。
- `:c:enum:`name``: C 列挙型にリンクします。
- `:c:macro:`name``: C のプリプロセッサマクロにリンクします。

C++ ドメイン

C++ ドメインは現代の C++ の構文、コンセプト、名前空間、スコープを扱います。

ディレクティブ

```
.. cpp:namespace:: guidedog

.. cpp:concept:: template<typename T> Reader

Specifies requirements for document reader types.

.. cpp:class:: template<typename Allocator> Builder

Constructs AST document nodes.

.. cpp:function:: NodeId add_text(std::string_view text)

Appends text to the active node.
```

ルール

- `:cpp:class:`name``: C++ のクラスまたは構造体にリンクします。
- `:cpp:func:`name``: C++ の関数またはメソッドにリンクします。
- `:cpp:member:`name`` / `:cpp:var:`name``: メンバーまたは変数にリンクします。
- `:cpp:type:`name``: C++ の型エイリアスまたは typedef にリンクします。
- `:cpp:concept:`name``: C++20 のコンセプトにリンクします。
- `:cpp:enum:`name``: C++ の列挙型にリンクします。

JavaScript ドメイン

JavaScript ドメインはブラウザー스크リプト、ライブラリ、Node.js モジュールを記述します。

ディレクティブ

```
.. js:module:: theme

.. js:class:: ThemeController(options)

    Manages color themes and local storage persistence.

.. js:method:: toggleTheme()

    Toggles between light and dark modes.
```

ロール

- `:js:func:`name``: JavaScript 関数にリンクします。
- `:js:meth:`name``: JavaScript メソッドにリンクします。
- `:js:class:`name``: JavaScript クラスにリンクします。
- `:js:data:`name``: JavaScript 変数にリンクします。
- `:js:attr:`name``: JavaScript オブジェクトの属性にリンクします。
- `:js:mod:`name``: JavaScript モジュールにリンクします。

相互参照の規則

ドメインの相互参照では次の修飾が使えます。

- **接頭辞を隠す**: 対象の前に `~` を付けると、名前の末尾だけを表示します。 `:py:func:`~pipeline.reader.DocumentReader.parse`` は `parse()` になります。
- **現在のスコープ**: 相対参照は、まず現在のモジュールや名前空間を調べ、その後で全体を検索します。
- **完全一致**: `.` を付けると、現在のモジュールや包含するオブジェクトから検索します。

プロジェクトの仕組み

プロジェクトルートは `conf.toml` のあるディレクトリー。 `root_doc` は読む順序の起点。 Guidedog は文書を探索・読み取り、関係を解決して完全な世代を公開する。

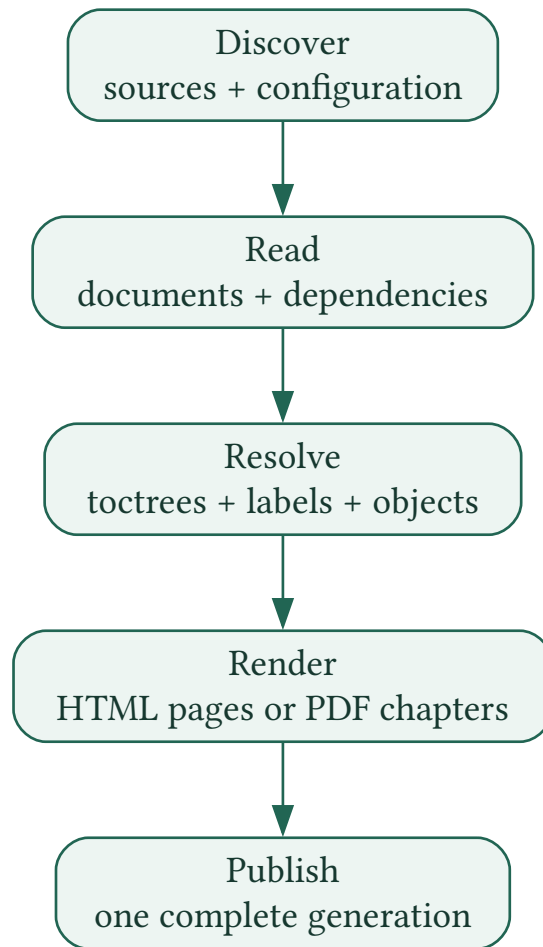


図 2 ビルドは探索、意味の処理、表示、公開を分離する。

文書とリソース

拡張子でリーダーを選ぶ。RST とプロジェクト Markdown は同じグラフを使う。画像とダウンロードはリソースで、文書が使うものをコピーする。テーマ資産は `html_static_path`。インクルードはソースまたは `include_roots` 内に限る。

独立したプロジェクト

マニュアルと設計記録は読者も読む順も違うため分ける。それぞれ設定、テンプレート、キャッシュ、出力を持つ。このリポジトリの `docs/manual` と `docs/gds` がその例だ。

差分ビルド

各出力の入力と参照結果を記録する。ソース変更は再読、依存変更は利用側の無効化、テンプレート変更は全ページに影響し得る。未変更文書は保存したオブジェクトから読める。

キャッシュは以前のビルドの証拠で、真実の源ではない。-E は保存環境を次回に使わず、-a は全出力を書く。解く問題は異なる。

再利用可能 = 入力と同じ ∧ 参照結果が同じ ∧ 出力が存在する

テンプレートの時刻もページへの入力だ。現在の分を表示するページは原文不変でも変わる。固定時刻には SOURCE_DATE_EPOCH を使う。

公開

変更は同階層のステージングに書く。完成とファイル・記録の永続化後に世代を切り替える。読者には完全な旧版か新版を見せ、中断コミットは次回処理する。設計と証明責務は **完全な世代を公開する**。

数学と証明

関係を調べやすくする数式を使う。記号は使う前に定義し、必要な単位を示す。数式は説明を支え、置き換えない。

インライン数式と独立した数式

短い式には math ロール、独立した式には math ディレクティブを使う。両形式が理解する共通の LaTeX 数式サブセットで書く。

```
A side of length :math:`s` encloses area :math:`s^2`.

.. math::
    :label: area-square

    A = s^2

The relation is :eq:`area-square`.
```

一辺の長さが s の正方形の面積は s^2 である。

$$A = s^2 \tag{1}$$

関係式は (1)。ラベル付きの式には安定したターゲットを与える。HTML は MathJax 記法を渡し、PDF は対応する LaTeX サブセットを Typst 数式へ変換する。未対応命令は報告し、任意の LaTeX パッケージは実行しない。

小さな証明

計上済みのバイト数を U 、上限を B 、要求量を n とする。全て非負で $U \leq B$ と仮定する。

受け入れ規則。 次の条件が成り立つ場合に限り、要求を受け入れる。

$$n \leq B - U. \tag{2}$$

命題。 要求を受け入れても $U + n \leq B$ が保たれる。

証明。 (2) の両辺に U を加えると $U + n \leq B$ 。実装は減算形を検査し、整数をあふれさせ得る $U + n$ を先に計算しない。証明と演算順序は一致する。

境界。 対象は計上したホスト領域で、別の Typst コンパイラ内部は制限しない。証明は明記した範囲内でのみ有効だ。

HTML の設定

これらのプロジェクトは MathJax 3.2.2 と SVG フォントデータを同梱し、数式のあるページだけで読み込む。ビルド後はオフラインで読め、図のスク립トや Web フォントサービスは不要。

自分のプロジェクトでは、既定で CDN の MathJax を使う。ローカルのコピーを配信するには、`_static` に置いて次を設定する。

```
html_static_path = ["_static"]
mathjax_path = "_static/tex-svg-full.js"
```

Web 数式の組版には JavaScript が必要で、無効時はソースが見える。PDF はブラウザの JavaScript を必要としない。MathJax コンポーネント契約を参照。

図で説明する

問いに合う図を選ぶ。フローチャートは選択、ブロック図は責任、シーケンス図は順序、グラフは関係を説明する。文章がより直接答えられるなら文章を使う。

フローチャート

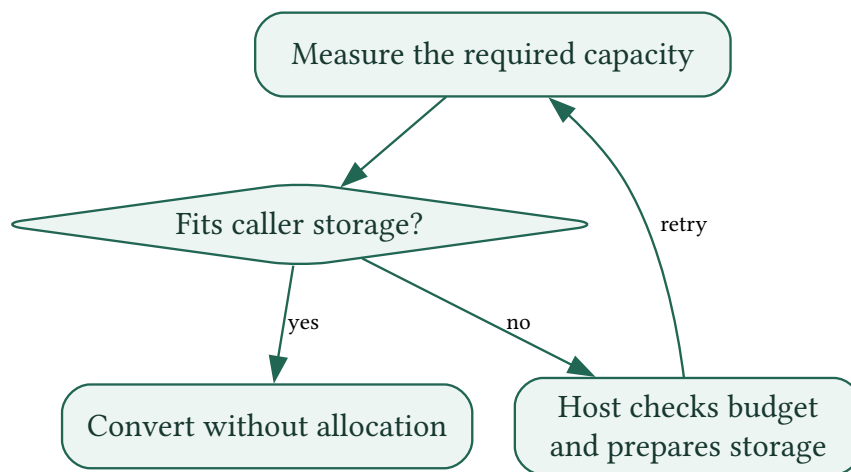


図 3 記憶領域を借用または拡張する前に、容量を確認する。

図は拡張時の所有者を示す。変換コアが割り当てるという意味ではない。この区別は契約の一部だ。

グラフを書く

```
.. graphviz::
    :caption: Parsing precedes resolution.
    :alt: The source becomes a tree, then references are resolved.

    digraph pipeline {
        graph [rankdir=LR, bgcolor="transparent"];
```

```
node [shape=box];
"source" -> "tree" -> "resolved tree";
}
```

graphviz、graph、digraph は組み込み。Guidedog が Graphviz で描画し、HTML は SVG、PDF は描画した図を使う。元のグラフは共通だ。

読みやすく仕上げる

ノードのラベルは短く、長い規則は図の下で説明する。主経路は一方向を基本にし、戻りの辺に条件を付ける。色だけでなく形と文字も使う。印刷では色がないかもしれない。

代替テキストとキャプションを付け、狭いブラウザと PDF で試す。小さすぎる字でなければ収まらない図は二つに分ける。寸法は 画像、動画、音声、ダウンロード を参照。

読みやすい組版

熟読前に構造が見えるページにする。明確な見出し、静かなナビゲーション、適度な本文幅が中心だ。色は関係や操作を示す補助とし、それだけに意味を託さない。

ウェブページ

独自レイアウトはプロジェクト案内、本文、ページ目次の三領域で、小画面はドロワーを使う。Libertinus Serif と Sans を同梱する。見出しは太いセリフ体、案内・ラベル・操作は対応するサンセリフ体。WOFF2 をサイトから配信し、フォントサービスに接続しない。

既定はライト。 ダークとシステムも選べ、同一オリジンで選択を記憶する。システムは OS に従う。印刷 HTML は明るい色を使う。JavaScript なしでも本文と案内は使える。テーマ操作と高度な検索は段階的な機能追加だ。

英語の本文はブラウザでも段落全体の **Knuth–Plass 改行** を使える。小さなスクリプトが読み込んだフォントを測り、箱・空白・ハイフネーションのペナルティーを作って段落の改行を選ぶ。Frank Liang のパターン法で候補を求め、元の TeX パターンとライセンスを同梱する。

強調と原文を保ち、本文幅の変更時に再計算する。最適化した一段落内のコピーは見かけの改行やハイフンを含まない原文になる。印刷時は通常段落に戻し、印刷レイアウトで改ページする。

リンク、数式、コード、操作要素、狭い幅、他言語には標準レイアウトを使う。良い間隔の解がない段落も同様だ。JavaScript なしでも読める。代替は言語対応の `hyphens: auto` と `text-wrap: pretty` だが、CSS だけで Knuth–Plass 最適化を保証するものではない。

PDF のページ

書籍も Web と同じ Libertinus を使い、両端揃え、最適化した改行、言語別ハイフネーションを設定する。間隔、柱、キャプション、コード、罫線も

制御する。フォント・分割コスト・段落幅は共同で働く。不自然な空きに強制改行を足すのは大抵適切でない。

Typst は PDF の段落を、Web の補助は対象の本文を最適化する。ソース・フォント・階層は共有するが改行は異なり得る。Typst テキスト契約とブラウザーのハイフネーション契約を参照。

デザインの変更

見た目の変数は `_static/editorial.css`、構造は `_templates/layout.html`、印刷は `_templates/book.typ` で編集する。読む順、フォーカス表示、ラベル、十分なコントラストを保つ。

変更後は両形式を作り、コード・表・図・数式を含むページを確認する。美しい表紙は本文の保証ではない。契約は [テンプレート](#)、全 API は [API Reference](#) と同じ PDF 本にある。

テンプレート

テンプレートは表示を決めるもので、本文を複製するものではありません。Guidedog は HTML に Jinja、PDF に Typst を使います。

Guidedog のテンプレートは普通のプロジェクトファイルです。自由に開いて編集できます。guidedog quickstart は実際に使うテンプレートをプロジェクトに配置します。

- `_templates/layout.html` は Jinja で HTML サイトの構造を定義します。
- `_templates/book.typ` は Typst で PDF のレイアウト、文字組み、表紙を定義します。
- `_static/guidedog.css` と `_static/guidedog.js` は既定のスタイルとブラウザーの動作を提供します。

テンプレートは外部パッケージやバイナリのキャッシュではなく、プロジェクト内にあります。編集後、`guidedog build` で再ビルドし、`guidedog serve` で確認できます。

HTML テンプレート

Guidedog は内蔵の Jinja エンジンで `layout.html` を描画します。一つの巨大なファイルにする必要はありません。部分テンプレート、マクロ、段階的な継承に分けられます。

テンプレートの分割

`templates_path` のディレクトリ内で、テンプレートを複数のファイルやサブディレクトリに分けられます。初期値は `["_templates"]` です。

```
my-docs/
├── conf.toml
├── index.rst
└── templates/
    ├── layout.html
    ├── base.html
    └── partials/
        ├── header.html
        ├── navigation.html
        └── searchbox.html
```

```
└─ footer.html
└─ macros/
    └─ components.html
```

Guidedog は `templates_path` のファイルとサブディレクトリを走査します。取り込みやインポートには、そのディレクトリからの相対パスを使います。

リスト 1 `_templates/layout.html`

```
{% extends "base.html" %}

{% block header %}
    {% include "partials/header.html" %}
{% endblock %}

{% block footer %}
    {% include "partials/footer.html" %}
{% endblock %}
```

Jinja マクロも別ファイルに定義し、必要な場所でインポートできます。

リスト 2 `_templates/macros/components.html`

```
{% macro badge(label, type="info") %}
    <span class="badge badge-{{ type }}">{{ label }}</span>
{% endmacro %}
```

リスト 3 テンプレートでマクロを使う

```
{% import "macros/components.html" as ui %}
{{ ui.badge("New", type="success") }}
```

!layout.html によるテーマの継承

既定のテーマの一部分だけを変えるなら、ページ全体を書き直す必要はありません。Sphinx と同じ感嘆符の構文で、組み込みのレイアウトを継承できます。

リスト 4 `_templates/layout.html`

```
{# Inherit Guidedog's built-in layout #}
{% extends "!layout.html" %}

{# Inject custom metadata or web fonts into the document head #}
{% block extrahead %}
    {{ super() }}
    <link rel="stylesheet" href="{{ pathto('_static/custom.css', 1) }}">
{% endblock %}

{# Replace or extend the footer with custom content #}
{% block footer %}
    {% include "partials/footer.html" %}
{% endblock %}
```

`{% extends "!layout.html" %}` または `{% extends "basic/layout.html" %}` は、プロジェクトの `_templates/layout.html` を再帰的に読む代わりに、テーマ本来のテンプレートを読み込みます。

上書きしたブロックで `{{ super() }}` を呼べば、親の内容を残したまま前後に要素を追加できます。

レイアウトのブロック

組み込みの `layout.html` は Sphinx の慣例に対応するブロックを定義します。

| ブロック | 用途 |
|--------------------------|---|
| <code>doctype</code> | 文書型宣言。初期値は <code><!DOCTYPE html></code> です。 |
| <code>htmltitle</code> | <code><head></code> 内の <code><title></code> 要素。 |
| <code>linktags</code> | ナビゲーションとメタ情報のリンク: <code>favicon</code> 、 <code>index</code> 、 <code>search</code> 、 <code>prev</code> 、 <code>next</code> 。 |
| <code>css</code> | スタイルシートのリンクとインライン CSS のルート変数。 |
| <code>scripts</code> | 検索インデックスや対話機能の JavaScript タグ。 |
| <code>extrahead</code> | フォント、解析ツール、追加メタタグ用の <code><head></code> 末尾の挿入位置。 |
| <code>header</code> | メインナビゲーションの直前に置くヘッダー。 |
| <code>relbar1</code> | 名称、バージョン、検索、テーマ切替を含む上部ナビゲーション。 |
| <code>rootrellink</code> | 関連リンクの前にあるナビゲーションの挿入位置。 |
| <code>relbaritems</code> | ナビゲーションバーの追加項目。 |
| <code>sidebar1</code> | 左のナビゲーションサイドバー。 |
| <code>sidebartoc</code> | <code>sidebar1</code> 内の目次ツリー。 |
| <code>breadcrumbs</code> | 本文上部のパンくずナビゲーション。 |
| <code>document</code> | 本文を囲むコンテナ。 |
| <code>body</code> | 現在の文書の HTML 本文 (<code>{{ body }}</code>)。 |
| <code>relbar2</code> | 前後の章へ移動する下部ナビゲーション。 |
| <code>footer</code> | 著作権、更新日時、ソースへのリンクを含むフッター。 |
| <code>sidebar2</code> | 右側の補助欄。ページ内の見出し一覧を表示します。 |

テンプレートの検索

Guidedog は `conf.toml` の `templates_path` を順に検索し、最後に組み込みテンプレートを調べます。名前は各ディレクトリからの相対パスで、`..` による外部への移動はできません。

`templates_path` のファイルとサブディレクトリはすべてビルドの依存入力です。追加、編集、削除を `guidedog build` と `guidedog serve` が検出し、サイトを再描画します。

ページ変数

Sphinx に変数名がある場合は同じ名前を使うため、Sphinx テーマの一部を移行できます。

| 変数 | 値 |
|--|---|
| <code>body</code> | HTML 形式の文書。 |
| <code>title</code> | 文書のタイトル。 |
| <code>pagename</code> , <code>docname</code> | 文書名。例は <code>usage/install</code> 。文書でないページでは <code>docname</code> は空、 <code>pagename</code> は Sphinx 同様に <code>genindex</code> 、 |

| 変数 | 値 |
|--|--|
| | <code>py-modindex</code> 、 <code>search</code> 、テンプレート生成ページの名前となる。 |
| <code>toc</code> | 目次ツリーから作る HTML のナビゲーション。 |
| <code>outline</code> | 文書内の節を HTML で示します。節がなければ空です。 |
| <code>prev</code> , <code>next</code> | 読む順で前後にあるページです。url (link も可) と title を持ち、端では存在しません。 |
| <code>project</code> , <code>version</code> ,
<code>release</code> , <code>copyright</code> ,
<code>language</code> | 同名の設定値。 |
| <code>html_title</code> , <code>docstitle</code> | <code>html_title</code> 。既定値は「<project> <release> documentation」です。 |
| <code>html_short_title</code> ,
<code>shorttitle</code> | <code>html_short_title</code> 。 |
| <code>root_doc</code> , <code>master_doc</code> | ルート文書の名前。 |
| <code>pathto_root</code> | ページからサイトのルートへのパスです。例は <code>../</code> 。 |
| <code>root_url</code> , <code>search_url</code> ,
<code>genindex_url</code> | ルートページ、検索ページ、総合索引へのリンク。 |
| <code>css_files</code> , <code>js_files</code> | url を持つファイルのリストです。Guidedog のファイルの後に <code>html_css_files</code> と <code>html_js_files</code> を並べます。 |
| <code>logo_url</code> , <code>favicon_url</code> | <code>_static</code> 内の <code>html_logo</code> と <code>html_favicon</code> です。未設定なら空です。 |
| <code>accent</code> | テーマの色 <code>html_theme_options.accent</code> 。 |
| <code>sourcelink_url</code> | ソース表示が有効なら <code>_sources</code> 内の文書ソース、それ以外は空です。 |
| <code>last_updated</code> | <code>html_last_updated_fmt</code> があれば、その書式のビルド日付です。なければ空です。 |
| <code>show_copyright</code> ,
<code>show_sphinx</code> ,
<code>show_guidedog</code> ,
<code>has_source</code> ,
<code>show_source</code> | <code>html_show_*</code> と <code>html_copy_source</code> の設定。 |
| <code>builder</code> , <code>file_suffix</code> | html などのビルダー名とページの拡張子。 |
| <code>html_context</code> の各キー | 型を保った値を、Sphinx が Python 値を渡すように渡す。false は <code>{% if %}</code> で偽、数値は数値、配列はリスト、テーブルは <code>conf.toml</code> のキー順を保つ辞書となる。 <code>guidedog migrate</code> は <code>conf.py</code> の <code>html_context</code> のリテラルを引き継ぐ。計算値は未定義となり、テンプレートは偽と扱う。 |

`pathto` は Sphinx のテンプレート関数。`pathto("usage/install")` は文書ページ、`pathto("_static/logo.svg", 1)` はサイト配下のファイルの URL を現在のページからの相対で返す。`singlehtml` では Sphinx 同様、文書は一ページの節となる。前者はそのページでは `#document-usage-install`、索引や検索では `index.html#document-usage-install`。`genindex` やテンプレートページなど文書以外の名前は、どのビルダーでもサイト直下のページを指す。

テンプレートから作るページ

`html_additional_pages` は Sphinx 同様、テンプレートだけでページを作る。キーはページ名、値は `templates_path` 内のテンプレート。

リスト 5 `conf.toml`

```
root_doc = "contents"

[html_additional_pages]
index = "indexcontent.html"
download = "download.html"
```

通常は `layout.html` を継承してブロックを埋め、他のページと見た目を揃える。

リスト 6 `_templates/indexcontent.html`

```
{% extends "layout.html" %}
{% block htmltitle %}<title>{{ shorttitle }}</title>{% endblock %}
{% block body %}
  <h1>{{ docstitle|e }}</h1>
  <p><a href="{{ pathto("tutorial/index") }}">Tutorial</a></p>
{% endblock %}
```

文書でないページの変数が渡る。`pagename` はページ名 (`index`)、`title` と `body` は空。`pathto`、`toc`、`toctree()`、`html_context` は通常通り使える。`html`、`dirhtml`、`singlehtml` は全てサイト直下に `<name>.html` (`html_file_suffix`) として毎回書く。テンプレート、または `templates_path` のどのファイルの変更も全ページを再構築する。

ページの配置と Sphinx との差分は次のとおりです。

- 同名の文書ページを置き換える。Sphinx で最後に書くのと同じだ。
`index = "landing.html"` は `index.html` をランディングページにし、目次は引き続き `index.rst` から作る。`singlehtml` では単一ページを残し、テンプレートページを警告付きで省く。
- `dirhtml` は `genindex.html` や `search.html` と並ぶ `download.html` に書く。Sphinx の `download/index.html` とは異なる。`pathto("download")` はそのファイルを返す。
- 名前はサイト直下のファイル名に限る。リンクがルート基準なので `"sub/page"` は警告 (`build.additional_page`) で拒否する。`genindex` など既存ファイル名も拒否する。Sphinx では一方を上書きする場合だ。
- `guidedog migrate` は `conf.py` の `html_additional_pages` を移します。

エラー

テンプレートの誤りはビルドを止め、ファイル、行番号、該当行、修正案を示します。

```
TEMPLATE ERROR                                     template.error

  _templates/base.html:1

No filter named 'default'.

Did you mean the filter 'default'?
```

未定義の変数は Jinja と同様に何も表示しません。

暴走するテンプレートもクラッシュせずエラーで停止する。構文（括弧、タグ、演算子・フィルター・elif の連鎖）が 100 段、マクロ・インクルード・再帰ループが 200 段を超える場合、または再帰が 512 KiB 以上のスタックを要する場合だ。自身を含むリストは Python 同様 [...] と表示する。

PDF テンプレート

PDF は Typst で組版する。_templates/book.typ は book 関数を定義する通常の Typst ファイル。Guidedog は次の形でソースを書く。

```
#import "/_templates/book.typ": book
#show: book.with(title: ..., author: ..., version: ..., date: ...,
  lang: ..., paper: ..., numbering: ..., logo: ...)
// the chapters, one per document of the root toctree
```

本の見た目を決める要素はすべてそのファイルにあります。フォント、用紙と余白、見出し、扉、柱、目次です。

パラメータ

パラメータ	値
title	pdf_documents の本の title、なければ project。
author	本の author、なければ設定の author。
version	release。
date	today は today の設定値、なければ today_fmt 形式のビルド日付です。
lang	language。
paper	"a4"、または pdf_paper_size が letter なら "us-letter"。
numbering	toctree に numbered があれば true。
logo	pdf_logo のパス。未設定なら省きます。
copyright	奥付用の copyright です。この引数を受け取るテンプレートにだけ渡します。
body	各章の内容。

テンプレートは既定値付きの引数を追加できる。quickstart のものには accent、フォント serif・sans・mono、文字の size、page-ref がある。最後を $n \Rightarrow [p. \#n]$ などの関数にすると、他のページへの参照にページ番号が付く。

Typst テンプレートの分割

Typst も一つのファイルにまとめる必要はありません。スタイル、マクロ、表紙を _templates/ の .typ ファイルに分け、#import と #include で組み合わせられます。

リスト 7 _templates/book.typ

```
#import "cover.typ": title-page
#import "typography.typ": apply-styles

#let book(
  title: "",
  author: "",
  version: "",
  date: "",
  lang: "en",
  paper: "a4",
  numbering: false,
  logo: none,
  body,
) = {
  apply-styles()
  title-page(title: title, author: author, version: version, logo: logo)
  body
}
```

テンプレートで使えるもの

Typst パッケージ

`#import "@preview/cetz:0.4.2"` などのパッケージは初回にダウンロードして使える。`pdf_packages = "offline"` ならディスク上のものだけでビルドする。

フォント

typst と同様にシステムフォントで使える。`pdf_font_paths` でフォルダーを追加する。`pdf_fonts = "embedded"` なら Typst 内蔵だけを使い、バイト単位で再現可能な本を作れる。

プリアンブル

`pdf_preamble` は `#show: book` の後に挿入する Typst ファイルです。独自テンプレートを作らずに、少数の `set • show` 規則を追加できます。

文書内の Typst

文書は reStructuredText と Markdown で、Typst ファイルは文書ではない。書籍に Typst 記法を入れるには raw ブロックを使う。Web ページでは省かれる。

```
.. raw:: typst
    #align(center)[#text(size: 14pt)[Only in the book]]
```

`.. only:: pdf` も通常の内容を本だけに限定します。

Typst のエラーは、テンプレートでも生成ソースでもファイルと行を示す。各 PDF のソースは `_build/pdf/sources/<pdf-filename>.typ`。`reference-en.pdf` なら `sources/reference-en.pdf.typ`。一冊のビルドでは従来の `_build/pdf/book-<language>.typ` も確認用に残す。失敗時は公開せず、ソースをエラーに示す `_build/.doctrees/failed` の場所に残す。複数冊の失敗ソースは別々に保つ。

新しいテンプレートの作成

quickstart が書いた、各部分の使い方を示すテンプレートから始めます。

1. 既定の CSS が前提とする `layout.html` のマークアップを保つか、`_static/guidedog.css` も置き換えてください。
2. 繰り返す部分を別ファイルにして `include` するか、`base.html` を作って `layout.html` から `extends` します。
3. 本では `book` の規則を変えるか、同じ引数を持つ新しい `book` 関数を書きます。

編集時は `guidedog serve` で確認します。保存してブラウザを更新すると、変更された入力を再ビルドしてからページを返します。

画像、動画、音声、ダウンロード

リソースは説明する文書の近くに置きます。相対パス、または `/` で始まるソースルートからのパスで指定します。文書が使うリソースをコピーします。動画はウェブで再生でき、PDF では録画の情報を示します。

画像

```
.. image:: diagrams/pipeline.png
   :alt: The build pipeline
   :width: 480px

.. figure:: /screenshots/search.png

   Searching the site.
```

ページは画像を `_images` にコピーし、本は埋め込みます。PNG、JPEG、GIF、SVG、WebP は両方で使えます。画像がない場合は警告し、ページは代替テキストを残し、本は画像を省きます。

動画と音声

`image` や `figure` が動画（`.mp4`、`.webm`、`.ogv`、`.mov`、`.m4v`）または音声（`.mp3`、`.ogg`、`.oga`、`.m4a`、`.opus`、`.wav`、`.flac`、`.aac`）を指す場合は再生します。動画に対する Docutils と同じ動作です。

```
.. figure:: media/tour.mp4
   :alt: A tour of the site
   :width: 640

   The two-minute tour.

.. image:: media/pronunciation.mp3
```

ページではブラウザのプレーヤーと、再生できない場合のファイルリンクを表示します。本は再生できないため、その位置の枠内に録音・録画を示します。テキスト出力は `[video: ...]` または `[audio: ...]` を書きます。

ダウンロード

`download` ロールは任意のファイルにリンクし、`_downloads` にコピーします。

```
Get :download:`the example project <examples/project.zip>`.
```


テーマのファイル

`_static` にテーマのスタイルシート、スクリプト、フォント、ロゴを置き、そのままサイトの `_static` にコピーします。 `html_static_path` でディレクトリを指定し、 `html_css_files`、 `html_js_files`、 `html_logo`、 `html_favicon` でファイルを指定します。本文のメディアは文書と同じ場所に置けば、一緒に移動してもリンクが保たれます。

国際化

構造を保って説明を翻訳します。Guidedog はメッセージを抽出し、翻訳をマージして指定言語でビルドします。原文を基準とし、訳がない場合は文章を作らず原文に戻します。

作業の流れ

1. `gettext` ビルダーでメッセージテンプレートを作ります。

```
guidedog build gettext
```

`_build/gettext` に、テキストドメインごとのテンプレート (`.pot`) を書き出す。段落、見出し、リスト項目、表のセル、キャプション、注意書き、用語、フィールドがメッセージになる。出典のファイルと行番号も記録される。

2. `sphinx-intl update` と同様に、各言語のカatalogを作成・更新します。

```
guidedog intl update -l de -l fr
```

カatalogは `locales/de/LC_MESSAGES/*.po` に作られる。次回以降は `msgmerge` と同様に新しいテンプレートをマージする。訳文は保持され、原文が変わったものは `fuzzy` として確認対象になる。文書から消えたメッセージは末尾に廃止項目 (`#-`) として残り、再登場したときに訳を復元できる。

3. 各 `msgstr` を訳文で埋めます。確認が済んだらそのメッセージの `#`、 `fuzzy` を削除します。 `fuzzy` な訳文は使いません。

```
msgid "Hello *world*, see the site_."
msgstr "Hallo *Welt*, siehe site_."
```

4. `sphinx-intl stat` と同様に翻訳の進捗を確認します。

```
guidedog intl stat -l de
```

5. 指定言語でビルドします。

```
guidedog build html -D language=de
guidedog build pdf -D language=de
```

または `conf.toml` に `language = "de"` を設定します。

書籍名には言語の接尾辞が付く。 `manual.pdf` は `manual-de.pdf` になる。ビルドは出力全体を一代として公開するため、複数の言語版には別々の出力先を使う。例えばドイ

ツ語は `guidedog build -b pdf docs build/books/de -D language=de`、英語は `guidedog build -b pdf docs build/books/en -D language=en` で作る。

他の入力と同じく、カタログが変わった文書だけを読み直します。カタログの追加・削除時はすべて読み直します。

`guidedog intl` は `sphinx-intl` のオプションを受け付ける。`-p DIR` はテンプレート（既定 `_build/gettext`）、繰り返し指定できる `-l LANG` は言語（既定はプロジェクトの言語）、`-d DIR` はカタログ（既定は `locale_dirs` の先頭）、`-w N` は行幅（既定 76）で、`--no-obsolete` も使える。

翻訳を書く

メッセージは段落や見出しなどの原文で、マークアップを含みます。`.rst` は `reStructuredText`、`.md` は `Markdown` です。訳文も同じ形式で読むため、原文と同様に強調、リンク、参照を使えます。

参照は同じままにしてください。すべての参照を同じ形で残し、周囲の文章と表示タイトルだけを変えます。

```
msgid "See :ref:`install` and the site_."
msgstr "Siehe :ref:`Installation <install>` und die site_."
```

訳文から参照が抜けると、Sphinx と同じ警告（`inconsistent references in translated message`）を出し、訳文を使う。原文にない参照を訳文が加えると解決できないため、警告して原文を使う。リーダーが訳文のマークアップを受け付けない場合も原文を使い、警告にその訳文を示す。

節のアンカーは言語によって変わりません。訳した見出しも原文のアンカーを持つため、サイト内外のリンクが維持されます。

設定項目

標準設定は Sphinx の名前と既定値に従います。Guidedog は訳さない参照付録向けに、抽出だけに適用する除外設定も提供します。

設定	意味
<code>language</code>	ビルドする言語。例は <code>"de"</code> と <code>"pt_BR"</code> 。
<code>locale_dirs</code>	ソースフォルダーからの相対位置でカタログを指定する。各フォルダーには <code>LANGUAGE/LC_MESSAGES/DOMAIN.po</code> （またはコンパイル済みの <code>.mo</code> ）が入る。既定は <code>["locales"]</code> 。複数ある場合は、訳が見つかった最初のものを使う。
<code>gettext_compact</code>	文書のメッセージを入れるカタログを決める。 <code>true</code> （既定）では、最上位の文書はそれぞれ独立し、同じフォルダー内の文書は共有する（ <code>guide/usage.rst</code> は <code>guide.po</code> ）。 <code>false</code> では文書ごとに作る。名前を指定すると、全ての文書がその名前のカタログを使う。
<code>gettext_location</code>	各メッセージの位置（ <code>#: ../../index.rst:12</code> ）を書きます。既定は <code>true</code> 。
<code>gettext_uuid</code>	各出現に識別子を付けます。文書、メッセージ、出現位置から作るため、ビルドごとに変わりません。

設定	意味
<code>gettext_auto_build</code>	各 <code>.po</code> を隣の <code>.mo</code> にコンパイルし、他のツールで使えます。既定は <code>true</code> 。Guidedog 自身は <code>.po</code> を読みます。
<code>gettext_additional_targets</code>	追加の翻訳対象： <code>index</code> （索引項目）、 <code>literal-block</code> 、 <code>doctest-block</code> 、 <code>raw</code> 、 <code>image</code> （代替テキスト）。
<code>gettext_allow_fuzzy_translations</code>	あいまいな翻訳も使います。既定は <code>false</code> 。
<code>gettext_exclude_patterns</code>	POT の抽出だけから除く文書パスのパターンです。例は <code>["api/**"]</code> 、既定は <code>[]</code> 。文書は HTML と PDF に残ります。
<code>gettext_last_translator</code> , <code>gettext_language_team</code>	テンプレートヘッダーの Last-Translator と Language-Team。
<code>figure_language_filename</code>	ビルド言語に対応する画像ファイル。既定は <code>"{root}.{language}{ext}"</code> 。 <code>language = "de"</code> なら、存在する場合に <code>img/logo.png</code> の代わりに <code>img/logo.de.png</code> を使う。フィールドは <code>{root}</code> 、 <code>{path}</code> 、 <code>{basename}</code> 、 <code>{ext}</code> 、 <code>{docpath}</code> 、 <code>{language}</code> 。
<code>translation_progress_class</code>	翻訳済みの要素に <code>translated</code> 、それ以外に <code>untranslated</code> クラスを付ける（ <code>true</code> ）。片方だけ（ <code>"translated"</code> または <code>"untranslated"</code> ）も選べる。スタイルシートで未翻訳箇所を示せる。

Guidedog の表示文言

Guidedog 自身も、注意書きの見出し、「Added in version 2.1」「Fig. 1」「Contents」、ページの「Previous」「Next」「Navigation」「Search」、検索と索引ページ、書籍の「Version」などを出力する。これらはテキストドメイン `guidedog` から、次の順に検索する。

1. プロジェクトの `locales/LANGUAGE/LC_MESSAGES/guidedog.po`。
2. Sphinx と 同じ く 、 プ ロ ジ ェ ク ト の `locales/LANGUAGE/LC_MESSAGES/sphinx.po`。
3. 組み込みカタログはドイツ語、スペイン語、フランス語、日本語、ブラジルポルトガル語、ロシア語、簡体字中国語に対応します。

メッセージ ID は英語で、Sphinx に同じ語があればそれと共通なので、プロジェクトの `sphinx.po` を引き続き使える。独自の `guidedog.po` で任意の語を置き換えたり、言語を追加したりできる。

テンプレートは Sphinx と同じく `_()`、`gettext()`、`ngettext()` で翻訳します。

```
<small>{{ _('Previous') }}</small>
{{ _('Last updated on %s.')|format(last_updated) }}
```

`conf.toml` で指定しなければ、`numfig_format`、`html_title`、`html_short_title` は言語に合わせます。

日付

`|today|`、書籍の日付、ページの `last_updated` は Sphinx と同じ規則で整形する。`today_fmt` と `html_last_updated_fmt` は `strftime` 形式で、月と曜日の名前は `language` に従う。`%b %d, %Y` は英語で「Sep 29, 2026」、ドイツ語で「Sept. 29, 2026」。空の形式は `_( '%b %d, %Y' )` となり、カタログで訳せる。上記の言語の名前は Unicode CLDR に基づく。その他は、Babel が知らない言語に対する Sphinx と同じく英語を使う。

日付はローカルタイムゾーンの日付を使う。`SOURCE_DATE_EPOCH` (1970 年からの秒数、UTC) を設定すると、どのマシンでも同じ日付で再現可能なビルドになる。

他のプロジェクトへのリンク

`sphinx.ext.intersphinx` を使うと、このプロジェクトで解決できない参照を他のプロジェクトのインベントリで探します。これは各 Sphinx・Guidedog サイトが公開する `objects.inv` です。`conf.toml` にプロジェクトを指定します。

```
extensions = ["sphinx.ext.intersphinx"]

[intersphinx_mapping]
python = ["https://docs.python.org/3/", ""]
click = ["https://click.palletsprojects.com/", ["click.inv", ""]]
```

各項目には相手のページのアドレスとインベントリの場所を書きます。URL、ソースからの相対ファイル、またはそのアドレスの `objects.inv` を表す `""` を使えます。リストは順に試します。Sphinx の `intersphinx_mapping` と同じですが、Python の `None` の代わりに `""` を使います。`guidedog migrate` が変換します。

参照に新しい書き方は不要です。

```
:func:`len`, :class:`list`, and :ref:`tut-packages` link into Python's
documentation.
:ref:`click:testing` looks only in Click's inventory.
:external+python:ref:`tut-packages` never looks in this project.
```

ラベルと文書へのリンクには相手のタイトルと「(in Python v3.14)」などのツールチップを付けます。Sphinx と同じく、`:doc:` 参照にはプロジェクト名 (`:doc:`python:tutorial/index``) が必要です。`intersphinx_disabled_reftypes` の既定値が `["std:doc"]` のためです。

ビルド開始時にインベントリを取得し、`_build/.doctrees/__intersphinx_cache__` に `intersphinx_cache_limit` 日保存します。既定は 5 日、負値なら保持します。ローカルファイルは毎回読みます。Sphinx と同じく URL の転送先ディレクトリをリンクの基準にし、キャッシュにも記録するため、キャッシュ利用時も同じリンクになります。例えば `https://werkzeug.palletsprojects.com/objects.inv` は `/en/stable/` に転送されます。`intersphinx_timeout` は取得ごとの秒数制限です。到達できないプロジェクトは警告し、参照は未解決のままです。

Guidedog は各ページ、ラベル、用語、オブジェクトを含む `objects.inv` を書き、他のプロジェクトから同じ方法でリンクできます。

オブジェクト型の宣言

標準ドメインにない名前が必要な場合は、TOML でオブジェクト型を宣言します。Python 拡張を読み込まずに参照先、索引項目、ルールを作れます。宣言が表すのは構文であり、実行する動作ではありません。

参照先：add_crossref_type

```
[[crossref_types]]
directive = "setting"
role = "setting"
index = "pair: %s; setting"
```

.. setting:: DEBUG は setting-DEBUG という ID の参照先と索引項目を作り、:setting:`DEBUG` からリンクできます。Sphinx と同じです。index は Sphinx の indextemplate に相当し、最初のコロンの前に種類 (single、pair、triple、see、seealso)、後に項目を書き、%s が名前を表します。

説明：add_object_type

```
[[object_types]]
directive = "django-admin"
role = "djangadmin"
index = "pair: %s; django-admin command"
name = "first-word"
display = "django-admin %s"
program = true
```

.. django-admin:: migrate [app_label] は普通のオブジェクトと同様に、シグネチャ、本文、ID (django-admin-migrate)、索引項目でコマンドを記述します。Sphinx では parse_node 関数で解析できますが、通常の処理は次のフィールドで指定します。

name

"whole" (既定) はシグネチャ全体を名前にします。"first-word" は最初の空白までを使います。例えば pdb の b(reak) [lineno] の名前は b(reak) です。

display

シグネチャの表示形式で、%s がその値です。既定では原文のままです。

program

true にすると、その名前が後続のオプションの所属するプログラムになります。.. program:: と同じです。

ディレクティブの別名

拡張は Sphinx のディレクティブを別名で登録できます。

```
[directive_aliases]
django-admin-option = "option"
awaitablefunction = "py:function"
```

ドメインのない参照先は標準または既定ドメインのディレクティブです。py:function のような参照先はドメインを明示します。

移行

`guidedog migrate` は `conf.py` に列挙したプロジェクト内の拡張モジュールを探し、`add_crossref_type`、`add_object_type`、`add_directive` の呼び出しを実行せずに読み、表を書きます。宣言できない `parse_node` (`name`、`display`、`program` に対応させます) や Python 製のディレクティブ、ロール、イベント処理を報告します。

autodoc による Python のドキュメント作成

Guidedog はソースコードを読んで Python のドキュメントを生成します。パッケージはインポートしません。`docstring` とシグネチャはドキュメントグラフに組み込まれます。実行時にしか作られないオブジェクトは、明示的に記述する必要があります。これは意図的な境界です。

設定

拡張機能を指定し、ソースディレクトリを基準にパッケージの場所を Guidedog に伝えます。Python の `sys.path` に相当します。

```
extensions = ["sphinx.ext.autodoc"]
autodoc_source_paths = ["../src"]
```

`guidedog migrate` は、`sys.path.insert(0, os.path.abspath('.'))` など、`conf.py` がパッケージを探すために使う行から `autodoc_source_paths` を生成します。別の場所にインストールされたパッケージは、そのディレクトリを追加した場合にだけ見つかります。依存パッケージのソースを取得したディレクトリなどを指定してください。Guidedog は Python 自身の `site-packages` を検索しません。

各ディレクティブは Sphinx と同じように使えます。

```
.. automodule:: shop.cart
   :members:
   :show-inheritance:

.. autoclass:: shop.Cart
   :members: add, remove
   :inherited-members:
```

Sphinx 9.1 の全オプションを同じ意味で受け付けます。 `:members:`、`:undoc-members:`、`:private-members:`、`:special-members:`、`:inherited-members:`、`:exclude-members:`、`:member-order:`、`:show-inheritance:`、`:imported-members:`、`:ignore-module-all:`、`:class-doc-from:`、`:no-value:`、`:annotation:`、`:no-index:`、`:no-index-entry:`、`:synopsis:`、`:platform:`、`:deprecated:` と、`autodoc_default_options` を取り消す `no-` 形式です。

設定項目

これらの `conf.toml` 設定は、Sphinx と同じ名前、値、既定値を使います。

`autodoc_source_paths`

Guidedog 独自の設定です。Python モジュールを探すディレクトリを、ソースディレクトリからの相対パスで検索順に指定します。指定した

ディレクトリ内のファイルだけを読み、外部に通じるシンボリックリンクは拒否します。

autoclass_content

クラスの説明に使う docstring を指定します。"class" (既定)、"init"、"both" から選びます。

autodoc_class_signature

"mixed" (既定) または "separated"。後者では `__init__` をメソッドとして記載します。

autodoc_default_options

すべてのディレクティブに適用するオプションの表です。例は `members = true` や `member-order = "bysource"`。値を `false` にすると、そのオプションを省きます。

autodoc_docstring_signature

`true` (既定) : docstring の先頭行が `name(args) -> result` のような形なら、シグネチャとして使います。

autodoc_inherit_docstrings

`true` (既定) : docstring のないメンバーは基底クラスのものを引き継ぎます。

autodoc_member_order

"alphabetical" (既定)、"groupwise"、"bysource" から選びます。

autodoc_preserve_defaults

`false` (既定) : 既定値を `repr()` の表現で表示します。`true` ではソースに書かれた形を使います。

autodoc_typehints

"signature" (既定)、"description" (:type: と :rtype: フィールド)、"both"、"none" から選びます。

autodoc_typehints_description_target

"all" (既定)、"documented"、"documented_params" から選びます。

autodoc_typehints_format

"short" (既定) または "fully-qualified"。

autodoc_use_type_comments

`true` (既定) : `# type:` コメントを型注釈として扱います。

strip_signature_backslash

Sphinx と同様に、シグネチャ内のバックスラッシュを二重にします。

autodoc_mock_imports, autodoc_type_aliases, autodoc_warningiserror

設定は受け付けます。インポートしないためモックは不要です。型エイリアスは適用しません。

Guidedog が Python の動作を判断する仕組み

名前は Python の束縛規則に従って解決します。パッケージの `__init__.py` にある `from .app import Flask` により、`flask.Flask` は `flask/app.py` のクラスを指します。autodoc は `:canonical:` `flask.app.Flask` を表示します。基底クラ

スも同様に解決し、Python と同じ規則でメソッド解決順序を計算します。継承したメンバーは基底クラスのソースから取得します。if TYPE_CHECKING: の中だけでインポートした名前は実行時には存在しないため、それを使う注釈は Sphinx と同じく原文のままです。

docstring は Python が保存する文字列です。エスケープを処理し、Python 3.13 のコンパイラと同じ規則でインデントを取り除きます。属性の説明は Sphinx の解析器と同じ場所から読みます。代入の行末または直前の #: コメントと、代入直後の文字列リテラルです。シグネチャは def から取得します。クラスでは Python の検索順に従い、メタクラスの __call__、__new__、__init__ を調べます。dataclass と NamedTuple ではフィールドを使います。@overload の各定義は実装のシグネチャに優先します。

Python を実行しないと分からないこと

ソースだけでは Python の動作を判断できない場合、Guidedog は対象の名前を示して報告します。文書化できなければ警告を出します。Sphinx より情報が少ない状態で文書化した場合は、-v で表示する注記を出します。

- コンパイル済みのモジュール (.so、.pyd) には読めるソースがないため、オブジェクトを見つけられません。
- コードの実行で得る値 (app = Flask(__name__)、now = datetime.now()) には :value: を付けません。計算が必要な既定値はソースの表記を使います。
- ソース検索パスの外にあるデコレータ (@click.command()) が生成する関数は、装飾前の関数として文書化します。
- ソース検索パスの外にあるパッケージのクラスは、インポート元のパスで名前を表示します。そのパスは定義元のモジュールと異なる場合があります。メンバー、コンストラクタのシグネチャ、継承される docstring は分かりません。
- 組み込み型の docstring は分かりません。そのため、独自の docstring がなく dict から継承するはずのメソッドは省きます。
- autodoc のイベント (autodoc-process-docstring、autodoc-skip-member など) を処理する Python 拡張は実行しません。
- 現実のコードを大きく超える深入れれ子は解析せず、autodoc.too_deep を警告します。Python 自体も 200 段の括弧の入れ子を拒否します。Guidedog は数百のオペランドを含む + の連鎖も拒否します。
- 増大し得る処理には上限があります。インポートと別名をたどる名前解決は、1 経路で 48 ステップ、全体で 100,000 ステップまでです。型エイリアスと定数の展開は 10,000 ステップまでで、それを超えると原文を表示します。クラスの MRO は 100 クラスまでたどります。1 つのディレクティブが生成できる数は 10,000 個までで、超過時は autodoc.limit を警告します。この上限に達するのは、自分の名前で自分自身を含むクラスです。

不足するパッケージのソースディレクトリを autodoc_source_paths に追加すれば、これらの制約の多くを解消できます。

再ビルド

autodoc が読んだ Python ファイルはすべて、そのページの依存関係になります。編集すると次のビルドでページを読み直します。オブジェクトが見つからなかったページは毎回読み直すため、ソースに追加されれば次のビルドで検出します。

安全性

文書のビルドでプロジェクトのコードを実行することはありません。Guidedog は Python ソースをテキストとして読み、include と同じ範囲制限を適用します。autodoc_source_paths の下だけを読み、外部に通じるリンクはたどりません。

Odin の文書化

Odin ドメインはパッケージ、プロシージャ、型、メンバーに名前付きターゲットを与える。Guidedog は Odin ソースから API ページも生成するが、コンパイラーもパッケージも実行しない。生成したシグネチャにも有用な契約が必要。

オブジェクトの記述

パッケージは、Python モジュールのように後続の記述の文脈を定める。

```
.. odin:package:: core:strings
   :synopsis: Procedures to manipulate UTF-8 encoded strings.
   :imports: rt=base:runtime

.. odin:procedure:: @(require_results) clone :: proc(s: string,
   allocator := context.allocator) -> (res: string, err: rt.Allocator_Error)
   #optional_allocator_error

   Clones a string.

   :param s: The string to be cloned.
   :result res: The cloned string.
   :result err: An allocator error, or ``nil``.

.. odin:struct:: Builder :: struct

   A dynamic byte buffer.

.. odin:field:: buf: [dynamic]byte
```

シグネチャは Odin の宣言と同じく一行で書き、そのまま表示する。以下の例は次のように見える。

```
shapes.Kind :: enum u8
```

```
Circle = 1
```

```
@(require_results) shapes.area :: proc(side: f32, scale: f32
= 1) -> (a: f32, ok: bool) #optional_ok
```

正方形を計測します。

パラメーター: `side` – 一辺の長さ。

戻り値:

- `a` – 面積。
- `ok` – 種類が分かっているか。

ディレクティブと受け付けるシグネチャ。

odin:package と odin:currentpackage

`core:fmt` やプロジェクト配下の `readers/sphinx` を指定する。オブジェクト名はインポートパスを使う (`core:fmt.println`)。 `odin:currentpackage` はターゲットを作らず文脈を変え、`None` は解除する。オプションは `:synopsis:`、`:platform:`、`:deprecated:`、`:no-index:`、`:imports:`。最後はシグネチャの別名を `alias=package` (`gd=core rst=readers/rst`) で指定し、`gd.Node_Id` を `core.Node_Id` に結ぶ。`:private:` は私有型とプロシージャー群の私有メンバー (`State Rules catalog_text`) を列挙する。リンク先の説明がないため文字列で表示し、生成ページは自動で列挙する。

odin:procedure (または odin:proc)

`name :: proc(params) -> results`。名前前の属性 (`@(require_results)`)、`proc` 前の `#force_inline`、呼出規約 (`proc "c" (...)`)、既定引数 (`x := 1`、`x: int = 1`)、`$T` 多相引数、`..` 可変長、`using`、`#c_vararg`、名前付き結果、タグ (`#optional_ok`)、`where` 節を扱う。`name(params) -> results` は短縮形。

odin:procgroupp

`name :: proc{a, b}` : 各メンバーは対応するプロシージャーにリンクする。

odin:struct, odin:union, odin:enum, odin:bitset, odin:bitfield, odin:type

`Name :: struct($T: typeid) #packed`、`Name :: union #no_nil {A, B}`、`Name :: enum u8`、`Name :: bit_set[Flag; u8]`、`Name :: bit_field u32`。
`odin:type` は `distinct` 型と別名 (`Handle :: distinct uintptr`、`Callback :: proc(x: int) -> bool`) を扱う。型の内容にメンバーを記述する。

odin:field と odin:enumerator

型の中では `name: Type` にタグ (`name: string `json:"n"``) やビット幅 (`low: u8 | 3`) を付けられ、列挙子は `Name` または `Name = 3`。名前は `Type.member` となる。

odin:const, odin:var, odin:foreign

`NAME :: 64` または `NAME : int : 64`、`name: Type`、`name := value`、`name: Type = value`。外部インポートは `libc "system:c"`。

全オブジェクトが `Sphinx` の `:no-index:`、`:no-index-entry:`、`:no-contents-entry:`、`:no-typesetting:` を受け付ける。独自には `:package:` (別パッケージで説明)、`:private:` と `:deprecated: message` (シグネチャになれば `@(private)` と `@(deprecated="message")` を表示)、`:availability: Windows, Linux` (本文先頭に対応先)、`:foreign: libc` (外部ブロックの手続き) がある。複数行のシグネチャはターゲットで異なる同じオブジェクトを表し、ID と索引は最初だけに付く。

型の位置にある名前は、まずシングネチャのパッケージと型で、次にそのまま検索してリンクする。組み込み型 (`int`、`string`、`rawptr`、`typeid`、`any` など)、予約語、多相名 (`$T` と以後の `T`、外側の型を含む)、配列長、既定値、定数、タグ、`where` 節はそのまま表示し、リンクにしない。

ロールと ID

`:odin:pkg:`、`:odin:proc:`、`:odin:type:` (構造体、共用体、列挙、ビット集合・フィールドなど)、`:odin:const:`、`:odin:var:`、`:odin:field:`、`:odin:enumerator:`、`:odin:obj:` (全種類) がリンクする。Python 同様 `~` は末尾だけを表示し (`:odin:proc:~core:fmt.println` は `println`)、先頭の `.` は名前の末尾を探す。参照元の型とパッケージ、そのままの名前、パスの `:` や `/` の後の順に検索する。`fmt.println` は `core:fmt.println`、`sphinx.Config` は `readers/sphinx.Config` を見つける。Odin と同じ末尾のパッケージ名で、`:imports:` の別名を優先する。`default-domain:: odin` または `primary_domain = "odin"` なら `odin:` は省ける。

ID は `odin-` と完全名に Sphinx の `make_id` を適用する。例は `odin-core-fmt.println`、`odin-readers-sphinx.Config.docname`。Python 同様大小文字とドットを保ち、パスの `:` と `/` はハイフンになる。URL で読め、ビルド間で安定し、末尾同名のパッケージも区別する。パッケージ ID は `odin-package-` とパス。索引は「println (procedure in core:fmt)」。`objects.inv` は `odin` と種別 (`odin:procedure`、`odin:struct`、`odin:field` など) で全オブジェクトを、Sphinx のモジュール同様パッケージを先に列挙する。

内容中の情報フィールドは他のドメイン同様にまとめる。`:param name:` (`:type name:` と併用)、名前付き結果の `:result name:`、`:returns:`、`:rtype:`。

ソースからの生成

ソースフォルダーからの相対位置でパッケージを指定する。

```
odin_autoapi_dirs = ["../src"]
odin_collections = {core = "/usr/local/lib/odin/core"} # optional
```

`odin_autoapi_dirs` 配下は相対パスで命名する (`src/shapes/round` は `shapes/round`)。コレクション内は Odin のインポート名 (`core:strings`) を使う。生成ディレクティブは `autodoc` と同様に動く。

```
.. odin:autopackage:: shapes
   :members:
   :undoc-members:
   :member-order: groupwise

.. odin:autoproc:: shapes.area
.. odin:autotype:: Shape
```

`odin:autopackage` はパッケージと文書コメントを、`:members:` で宣言全てまたは指定分も出力する。`odin:autoproc`、`odin:autoprocgroup`、`odin:autotype`、`odin:autoconst`、`odin:autovar` は現在のパッケージ名または `package.name` の宣言一つを出す。オプションは `:members:`、`:undoc-members:`、`:private-members:` (通常除く `@(private)`)、`:exclude-members:`、`:member-order:` (既定の `source` はファイル名と位置順、`alphabetical`、`groupwise` は「Types」「Procedures」)

「Procedure groups」「Constants」「Variables」「Foreign imports」で分類)、
`:no-index:`。

文書コメントは宣言直上の `//` 行や `/* */` ブロック、フィールドや列挙子の行末コメント。プレーンテキストを次の規則で `reStructuredText` にする。

- 段落は段落に、`- item` の行はリストになる。
- 空行またはコロンの終わる行 (`Example:`) の後の字下げブロックは `code-block:: odin`、`Output:` の後は `text` になる。段落行直後のテキストより深い字下げの行は、その段落を続ける。
- `Inputs:` の `- name: text` 項目は `:param name:`、`Returns:` は `:result name:` (名前なしなら `:returns:`) になる。Odin のコアライブラリーと同じ書式だ。
- `NOTE:` または `WARNING:` で始まる行は注記または警告になる。
- ``code`` はリテラル。 `reStructuredText` が記法と読む他の文字 (`*`、`|`、語末の `_` など) はエスケープし、コメントをそのまま表示して警告を出さない。

API ページ

`odin_autoapi_dirs` を設定すると、各パッケージに `sphinx-autoapi` のようなページを読み取り時に生成する。`api/shapes` は `odin_autoapi_options` 付きの `odin:autopackage:: shapes`、`api/index` は各パッケージと概要を含む。 `toctree`、検索、総索引、`singlehtml`、PDF に入るプロジェクト文書だが、ソースフォルダーには書かない。設定は次の通り。

`odin_autoapi_dirs`

ページを作るパッケージのフォルダーをソースからの相対位置で指定する。その配下と `odin_collections` のファイルだけを読み、外へ向かうリンクは拒否する。

`odin_collections`

`core:strings` などの引数に使う、コレクション名とフォルダーの表。これらのパッケージにはページを作らない。

`odin_autoapi_root`

ページのフォルダー。既定は `"api"`。同名のソース文書があればそちらを残し、警告する。

`odin_autoapi_options`

既定は `["members", "undoc-members"]`。 `private-members` も追加できる。

`odin_autoapi_member_order`

`"source"` (既定)、`"alphabetical"`、`"groupwise"`。

`odin_autoapi_add_toctree_entry`

`true` (既定) : `api/index` をルート文書の最初の `toctree` に加える。

`odin_autoapi_generate_api_docs`

`true` (既定)。 `false` はディレクティブ用にだけフォルダーを読む。

文書化しないパッケージの型 (`core:io.Writer`) にはリンクできない。 `-n` では `nitpick_ignore_regex = ["odin:type", "(core|base):.*"]` で抑制する。 `Guidedog` の `docs/api` もこの設定を使う。

プラットフォーム

Odin と同じファイル判定で、パッケージの全ターゲットを文書化する。`*_windows.odin`、`*_linux_arm64.odin`、`*_amd64.odin` は該当先のみ。`#+build linux, darwin`、`#+build !windows` (旧 `//+build` も) が絞る。ファイルの `when ODIN_OS == .Windows` が `ODIN_OS` と `ODIN_ARCH` の比較だけなら、分岐と `else` を対応先へ分ける。`*_test.odin` と `#+ignore` は除く。全ターゲット共通の宣言は一回、部分対応は「Availability: Windows」などで示す。異なるシグネチャはそれぞれ対応先付きで示し、ID は最初だけ。部分対応のパッケージは `:platform:` で示す。

再ビルド

読んだ各 Odin ファイルはページの依存となる。ソース編集後は、そのパッケージを説明するページだけを再読する。パッケージの追加・削除・概要変更時は索引も再読する。`guidedog serve` は HTML を返す前に `odin_autoapi_dirs` と `odin_collections` の変更時刻を調べる。編集後にブラウザーを更新する。

制約

- ・ソースは解析だけで、コンパイルや型検査はしない。型と定数値はそのまま表示し、100 文字超の値は省く。推論もない。`A :: B` の型か定数かは名前で判定し、全大文字でない大文字始まりの名前や組み込み型を型とする。
- ・`when` の条件は `ODIN_OS` と `ODIN_ARCH` の比較以外は評価せず、両方の分岐を記載する。
- ・私有型は文字列で示す。`:private-members:` がなければリンク先の項目がないためだ。生成したパッケージは `:private:` に列挙するので `-n` でも報告しない。
- ・`core:odin/parser` は `#optional_ok` と `where` を併用するプロシーチャーを拒否する。そのファイルを報告し、読めた部分は文書化する。
- ・Odin のパーサーは深い入れ子で多くのスタックを要し、数百の括弧でスレッドをあふれさせる。実用を大きく超えるファイル (括弧・型が約 50 段、一連の `else` が約千個) は `odin.autodoc.nesting` で除き、パッケージの残りは記載する。`when` の `&&`、`||`、`!` は 64 段まで評価し、それ以上は未評価とする。

実践的な手順

各ページは具体的な作業から始まり、コマンドと結果を示します。

Guidedog のビルドとインストール

macOS に Homebrew でインストールする

Apple シリコンまたは Intel 搭載の Mac では、公式 `tap insanai/homebrew-guidedog` を使います。

```
brew tap insanai/guidedog
brew install insanai/guidedog/guidedog
guidedog --version
```

Homebrew はプロセッサに合ったリリースアーカイブを選び、SHA-256 を検証します。パッケージには Typst、Graphviz、tree-sitter が含まれています。HTML と PDF の生成に Odin コンパイラ、Python 環境、別途の Typst インストールは不要です。対応する OS は macOS 12 以降です。

初めてのプロジェクトに沿って最初のプロジェクトを作成してください。インストール済みのバージョンを更新するには、次を実行します。

```
brew update
brew upgrade insanai/guidedog/guidedog
```

`brew test insanai/guidedog/guidedog` を実行すると、プロジェクトを作成して HTML と PDF を生成し、インストールしたパッケージを確認できます。

ソースからビルドする

`tools/odin-release.txt` で固定された Odin ツールチェーンを使ってください。リリースは Odin dev-2026-10 と Typst 0.15.1 を使用しています。共有ライブラリを使うビルドには Graphviz の開発用ライブラリが必要です。

開発用ビルド

リポジトリのルートで実行します。

```
mkdir -p build
native/tree-sitter/build.sh
odin build cmd/guidedog -out:build/guidedog -o:speed -vet -strict-style
build/guidedog --version
```

```
build/guidedog formats
```

tree-sitter のビルドは、ソースベースの autodoc 用に固定版の Python 文法を用意する。Graphviz の開発ファイルはシステムのパッケージ管理で導入する。Typst ブリッジのないビルドでは、インストール済みの typst で PDF を作る。--typst=/absolute/path/to/typst で明示的に選べる。

組み込みコンパイラ

ネイティブ依存を一度ビルドし、Odin 実行ファイルにリンクします。

```
cargo build --release --manifest-path native/typst_bridge/Cargo.toml
native/graphviz/build.sh
odin build cmd/guidedog -out:build/guidedog -o:speed \
  -define:GUIDEDOG_TYPST_BRIDGE=true -define:GRAPHDOG_STATIC=true
```

Typst ブリッジは Rust コードで、Guidedoc の無割り当て境界の外にある。埋め込みビルドでは別の Typst 実行ファイルを起動せずに済むが、コンパイラ自身のメモリ消費はなくなる。

Windows では出力名を build/guidedog.exe にする。パスなしで guidedog を使うには、実行ファイルを PATH に入れる。

文書のビルドと公開

通常の実出力ディレクトリにビルドします。

```
guidedog build html docs/manual -j auto -W
guidedog build pdf docs/manual -W
```

サイトは docs/manual/_build/html、PDF と生成した Typst ソースは docs/manual/_build/pdf にある。HTML は静的ファイルとして配信でき、本番環境に Odin サーバーは不要。

出力先の選択

Sphinx 形式ではソースと出力先を明示します。

```
guidedog build -b html docs/manual build/manual-site -W
```

出力はソースの外に置く。公開によってソースツリーを置き換える配置は Guidedog が拒否する。独自の出力先は build/.manual-site.guidedog/.doctrees のような同階層の安定したキャッシュを使うため、公開ディレクトリーには公開物だけが入る。

プレビューと検査

```
guidedog serve docs/manual
guidedog build dummy docs/manual -n -W
```

dummy ビルダーはプロジェクトを読み、参照を解決するが公開物を作らない。意味の簡易チェックに使える。最終レイアウトのテストにはならない。

再現可能な時刻

```
SOURCE_DATE_EPOCH=1790812800 guidedog build html docs/manual -W
```

これでビルド時刻を固定する。再現性はソース、テンプレート、フォント、コンパイラーの版、リソースにも依存する。時刻の固定だけでは足りない。

ビルド成功後にディレクトリー全体を公開する。正常終了は必要だが、テンプレート変更後は見た目の確認も必要。

Sphinx プロジェクトの移行

Guidedog は Sphinx のプロジェクト構造と多くの設定名に従うが、`conf.py` や Python 拡張は実行しない。移行はデータと実行可能な設定を分けるところから始める。

```
guidedog migrate path/to/conf.py
guidedog build html path/to/docs --diagnostics=json
```

`migrate` はリテラルの設定を読み、`conf.toml` に書く。計算で得る値は人の判断が必要として報告する。値を調べるためにプロジェクトをインポートしたり Python を実行したりしない。同等と見なす前に生成した設定を確認する。

3 段階で進める

1. まず文書グラフを正しくします。ルート、拡張子、除外、include を確認します。
2. 次に参照を正しくします。ドメイン、インベントリ、独自のオブジェクト型を確認します。
3. 最後に表示を正しくします。テンプレート、図、数式、両方の出力を確認します。

未知のディレクティブは独自拡張のものかもしれない。HTML ビルドの成功だけでは、その意味まで再現したとは言えない。診断を読み、置き換えるか、対応するエイリアスを設定するか、制限を明記して残すかを判断する。

対応範囲と CPython、Django、Flask の検証結果は [Sphinx との互換性](#) を参照。宣言的なカスタムオブジェクトは [オブジェクト型の宣言](#) を参照。

コードベースの文書化

公開する名前には契約が必要だ。入力、結果、所有権、失敗する条件を説明する。シグネチャだけではマニュアルにならない。

Odin

```
odin_autoapi_dirs = ["../lib"]
odin_autoapi_root = "api"
odin_autoapi_options = ["members", "undoc-members"]
```

Guidedog は Odin のパーサーでソースを読む。対象パッケージをコンパイルしたり実行したりしない。生成ページは文書グラフとオブジェクト一覧に加わる。このリポジトリの独立した API プロジェクトは `docs/api`。ディレクティブ、シグネチャ、オプションは [Odin の文書化](#) を参照。

Python

```
extensions = ["sphinx.ext.autodoc"]
autodoc_source_paths = ["../src"]
```

Guidedog は Python ソースを解析し、インポートしない。これでインポートの副作用を避けるが、実行時に生成されるオブジェクトは見つからないことがある。公開インターフェースの一部なら明示的に説明する。探索規則の詳細は [autodoc による Python のドキュメント作成](#) を参照。

プロジェクト間のリンク

生成した `objects.inv` をサイトと一緒に公開する。他のプロジェクトから `intersphinx` のマッピングで名前を解決できる。再現可能なビルドには明示的なマッピングを使う。[他のプロジェクトへのリンク](#) を参照。

この文書の保守

Guidedog のプロジェクトは `docs/manual`、`docs/api`、`docs/gds` の三つ。マニュアルと GDS は `guidedog quickstart` で作った。生成されたテンプレートを直接編集する。設計記録はマニュアルの目次に入れない。

マニュアルのビルド

リポジトリのルートで実行します。

```
build/guidedog build html docs/manual -j auto -W
build/guidedog build pdf docs/manual -W
build/guidedog serve docs/manual
```

マニュアルの目次には完全な Odin API リファレンスも入る。PDF コマンドは説明と宣言をリンクで結ぶ一冊の本を作る。マニュアルと独立した API プロジェクトはどちらも `lib/` から生成するため、複製した原稿を同期する必要はない。

API プロジェクトだけをビルドするには、次を実行します。

```
build/guidedog build html docs/api -W
build/guidedog build pdf docs/api -W
```

議論文書のビルドと検査

```
build/guidedog gds index
build/guidedog gds check
build/guidedog gds build --target=both
```

GDS CLI はメタデータ、番号、状態、復旧を管理する。RST の公開処理はプロジェクトエンジンと呼び、完全な HTML サイトと記録ごとの PDF を作る。セクターが制限するのは PDF だけで、HTML は完全なカ

タログを持つ一貫したサイトのまま。選択した PDF のプレビューは `_build/selected/<artifact>/pdf` を使い、`_build/pdf` の全体カタログを保つ。

```
build/guidedog gds build guidedoc-odin --target=pdf
build/guidedog build html docs/gds -W
build/guidedog build pdf docs/gds -W
```

通常のプロジェクトコマンドも使える。GDS は独自の `conf.toml` と `_build` を持つ。執筆手順は `docs/gds/workflow.rst` を参照。

公開する前に

リンクと診断を確認し、幅広い表やコードブロックを調べる。組版を変更したら PDF を画像化して確認する。HTML はデスクトップとスマートフォンの幅、ライトとダークの両方で試す。数式と図がネット接続なしでも動くことを確かめる。エンジンの変更には関連するライブラリーと統合テストを実行する。

リファレンス

これらのページは規約を示し、検索・参照するためにあります。最初の例が必要ならチュートリアルから始めてください。

コマンドラインリファレンス

`guidedog` では、文書プロジェクトの管理、ビルド、プレビュー、翻訳、変換をコマンドラインから行えます。

基本構文

```
guidedog <command> [arguments] [options]
guidedog --version
guidedog --help
```

終了コード

表 6 終了状態

終了コード	条件
0	成功。すべての処理が完了し、未処理のエラーはありません。
1	ビルドまたは変換のエラー。 <code>-w</code> によって警告をエラーとした場合も含みます。
2	コマンドラインの構文、フラグ、オプションに誤りがあります。
3	設定した資源の上限に達したか、メモリを確保できません。
4	ファイル操作または外部コンパイラーが失敗しました。
5	内部エラー。診断に報告方法が記載されます。
70	クラッシュ。前回の公開済み出力は残ります。

コマンド一覧

表 7 CLI コマンド

コマンド	用途
<code>quickstart [DIR]</code>	文書プロジェクトを作成し、編集できるテンプレートと設定を配置します。
<code>build [BUILDER] [DIR]</code>	指定した出力形式で文書プロジェクトをビルドします。
<code>serve [DIR]</code>	ローカルの HTTP プレビューサーバーを起動し、ファイルの変更時に自動で再ビルドします。
<code>clean [DIR]</code>	生成した出力とキャッシュを削除します。
<code>convert INPUT</code>	単一のファイルまたは標準入力を直接変換します。
<code>migrate [conf.py]</code>	Sphinx の <code>conf.py</code> を宣言的な <code>conf.toml</code> に変換します。
<code>intl <update stat></code>	PO 翻訳カタログを更新するか、翻訳の進捗を表示します。
<code>formats</code>	組み込まれたリーダー、レンダラー、形式への適合状況を一覧表示します。
<code>gds <COMMAND></code>	Guidedog Discussions の設計記録とライフサイクルを管理します。

guidedog quickstart

プロジェクトディレクトリに `conf.toml`、`index.rst`、`_templates/` のテンプレート、`_static/` のスタイルを作成します。

```
# Interactive prompt
guidedog quickstart docs

# Non-interactive creation with predefined options
guidedog quickstart docs -q -p "My Project" -a "Author Name" -v 1.0 -l en
```

オプション:

- `-q`: 対話入力を省き、コマンドラインの指定値を使います。
- `-p`, `--project NAME`: 表示用のプロジェクト名を指定します。
- `-a`, `--author NAME`: 著者または組織名を指定します。
- `-v VERSION`: 短いバージョン番号を指定します。例: `1.0`。
- `-r RELEASE`: 完全なリリース識別子を指定します。例: `1.0.0-rc1`。
- `-l LANGUAGE`: 既定の言語を指定します。初期値は `en` です。
- `--sep`: ルートにソースを置かず、`source/` と `build/` を分けて作成します。
- `--suffix EXT`: ソース文書の既定の拡張子を指定します。初期値は `.rst` です。

guidedog build

ソースの読み取り、目次ツリーと相互参照の解決、出力の公開をまとめて行います。

```
# Standard project build
guidedog build html docs
```

```
guidedog build pdf docs

# Sphinx-build compatibility syntax
guidedog build -b html docs _build/html
guidedog build -M html docs _build
```

対応するビルダー:

- `html`: 検索とナビゲーションを備えた HTML サイトを生成します。
- `dirhtml`: `dir/index.html` のようなディレクトリ形式の URL を生成します。
- `singlehtml`: すべての文書を一つの HTML ページにまとめます。
- `pdf`: Typst で PDF の本を組版します。別名は `latex` と `latexpdf` です。
- `text`: 文書ごとにプレーンテキストを生成します。
- `gettext`: 翻訳対象の文字列を GNU gettext の POT テンプレートに抽出します。
- `dummy`: 出力ファイルを作らずに文書ツリーを解析し、参照を解決します。構文確認に使います。

ビルドのオプション:

- `-a`: 更新時刻によらず、すべての出力を書き込みます。
- `-E`: キャッシュを使わず、環境を作り直します。
- `-W`: 警告があればビルドを失敗とします。
- `--keep-going`: `-W` 使用時でも残りの文書进行处理し、終了前にすべてのエラーを報告します。
- `-n`: 未解決の相互参照をすべて警告します。
- `-j N / -j auto`: 文書解析の並列数を指定します。`auto` は CPU コア数に合わせます。
- `-D name=value`: この実行だけ `conf.toml` の設定を上書きします。
例: `-D language=de`。
- `-D table.key=value`: `html_context` などの表の設定のキーを一つだけ設定し、ほかのキーは残します。`sphinx-build` と同じです。
- `-t TAG: only` ディレクティブで使う条件タグを定義します。
- `-c DIR`: `conf.toml` があるディレクトリを指定します。
- `-C`: `conf.toml` を読み込まずにビルドします。
- `-v`: 詳細表示。補足情報と処理時間を表示します。
- `-q`: 警告とエラーだけを表示します。
- `--color=always|never|auto`: 端末の ANSI 色表示を指定します。
- `--diagnostics=json`: 機械処理用の JSON 診断を標準エラー出力に書き込みます。
- `--budget=MIB`: プロジェクトホストが保持するメモリの上限を指定します。初期値は 1024 MiB です。
- `--memory=ram|disk`: メモリ予算を使い切ったときの動作を指定します。
- `--untrusted`: 生の埋め込みマークアップを省き、ネットワーク取得を無効にします。機能を制限する方針であり、プロセスのサンドボックスではありません。

guidedog serve

ローカルの HTTP サーバーとファイル監視を起動します。ソース、テンプレート、素材の変更時に再ビルドし、ブラウザを更新します。

```
# Serve current project on default port (8000)
guidedog serve docs

# Serve on custom port and host
guidedog serve docs --port 9000 --host 0.0.0.0
```

オプション:

- `-p, --port PORT`: 待ち受けポート。初期値は `8000` です。
- `--host HOST`: バインドする IP アドレス。初期値は `127.0.0.1` です。

guidedog clean

ビルドディレクトリと環境のキャッシュを削除します。

```
guidedog clean docs
```

guidedog convert

プロジェクトや設定ファイルを使わず、ファイルまたは標準入力を変換します。

```
# Convert reStructuredText to HTML
guidedog convert guide.rst --output guide.html

# Convert Markdown to PDF
guidedog convert guide.md --to pdf --output guide.pdf

# Stream conversion from standard input to standard output
cat document.md | guidedog convert - --from commonmark --to html --output -
```

オプション:

- `-o, --output FILE`: 出力先のパス。`-` は標準出力です。
- `--to FORMAT`: 出力形式は `html`、`pdf`、`text`。ファイル名からも推定できます。
- `--from FORMAT`: 入力リーダーは `rst`、`commonmark`、`myst`、`typst`。拡張子からも推定できます。
- `--force`: 確認せずに既存の出力ファイルを上書きします。
- `--strict`: 警告があれば失敗とします。
- `--raw=allow|omit`: 埋め込みの生コードの扱い。初期値は `omit` です。
- `--emit-typst FILE`: 生成した中間 Typst ソースを別のファイルに保存します。

guidedog migrate

Sphinx の Python 設定ファイル `conf.py` を宣言的な `conf.toml` に変換します。

```
# Convert conf.py in-place
guidedog migrate docs/conf.py

# Output to a specific file
guidedog migrate docs/conf.py -o docs/conf.toml --force
```

オプション:

- `-o FILE`: 指定のパスに保存します。`-` は標準出力です。
- `--force`: 既存の `conf.toml` を上書きします。

guidedog intl

多言語文書の翻訳カタログを管理します。

```
# Step 1: Extract POT templates
guidedog build gettext docs

# Step 2: Create or update language PO catalogs
guidedog intl update -l de -l fr docs

# Check translation completion percentages
guidedog intl stat -l de docs
```

`intl update` のオプション:

- `-l, --language LANG`: 対象言語のコード。複数回指定できます。
- `-p, --pot-dir DIR`: 抽出済み POT テンプレートのパスを指定します。

guidedog formats

組み込まれたリーダー、レンダラー、構文ハイライト機能、仕様テストの結果を表示します。

```
guidedog formats
```

guidedog gds

Guidedog Discussions のアーキテクチャ設計提案 (RFC) を管理します。

```
guidedog gds new "Feature Title" --author="Name"
guidedog gds list --state=prediscussion
guidedog gds show 0003
guidedog gds promote 0003 --dry-run
guidedog gds state 0003 --to=accepted --reason="Consensus reached"
guidedog gds index
guidedog gds check --render
guidedog gds recover --rollback
```

設定リファレンス

Guidedog の設定はプロジェクトルートの `conf.toml` に記述します。宣言的な TOML データとして読み込まれ、任意のコードは実行されません。

多くの設定キーは Sphinx の `conf.py` 変数に対応し、`guidedog migrate conf.py` で移行できます。

最小構成

基本情報だけでもプロジェクトを作れます。省略した設定には初期値が使われます。

```
project = "Field Notes"
author = "Your Name"
version = "1.0"
release = "1.0.0"
language = "en"
root_doc = "index"
source_suffix = [".rst", ".md"]
```

```
templates_path = ["_templates"]
html_static_path = ["_static"]
pdf_paper_size = "a4"
```

プロジェクト情報

表 8 メタデータの設定

キー	型	初期値	説明
<code>project</code>	文字列	<code>"Python"</code>	表示用のプロジェクト名。
<code>author</code>	文字列	<code>" "</code>	著者または組織名。
<code>copyright</code>	文字列	<code>" "</code>	フッターに表示する著作権表記。
<code>version</code>	文字列	<code>" "</code>	短いバージョン番号。例: <code>"1.0"</code> 。
<code>release</code>	文字列	<code>" "</code>	完全なバージョン番号。 alpha や beta の識別子も含みます。例: <code>"1.0.0b1"</code> 。
<code>language</code>	文字列	<code>"en"</code>	組版、ハイフネーション、翻訳に使う ISO 言語コード。 例: <code>"de"</code> 、 <code>"zh_CN"</code> 。
<code>today</code>	文字列	<code>" "</code>	日付の表示文字列。未指定なら現在の日付を <code>today_fmt</code> で整形します。
<code>today_fmt</code>	文字列	<code>" "</code>	文書の日付に使う書式文字列。

一般設定

表 9 検出と解析の設定

キー	型	初期値	説明
root_doc	文字列	"index"	目次のルートとなる文書。
source_suffix	配列	[".rst"]	文書ソースとして認識する拡張子。
exclude_patterns	配列	[]	ソース検出から除外するファイルやディレクトリの glob パターン。
include_roots	配列	[]	外部ファイルの取り込みを許可するルートディレクトリ。
templates_path	配列	[]	HTML と PDF のテンプレートディレクトリ。組み込みより先に検索します。
extensions	配列	[]	有効にする組み込みの Sphinx 拡張。例: <code>"sphinx.ext.autodoc"</code> 。
primary_domain	文字列	"py"	接頭辞のないディレクティブやロールに使う既定のドメイン。例: "py"、"c"、"odin"。
highlight_language	文字列	"default"	コードブロックで言語を省略したときの初期値。
pygments_style	文字列	""	明るいテーマで使う Pygments のハイライトスタイル。
pygments_dark_style	文字列	""	暗いテーマで使う構文ハイライトスタイル。
smartquotes	真偽値	true	直線の引用符やハイフンを組版用の句読点に変換します。
rst_prolog	文字列	""	各文書の先頭に追加する reStructuredText 断片。
rst_epilog	文字列	""	各文書の末尾に追加する reStructuredText 断片。

番号と数式

表 10 番号の設定

キー	型	初期値	説明
<code>numfig</code>	真偽値	<code>false</code>	図、表、コードに自動で番号を付けます。
<code>numfig_secnum_depth</code>	整数	<code>1</code>	図番号に含める見出しの深さ。 <code>1</code> なら Fig. 2.1 のようになります。
<code>numfig_format</code>	テーブル	下記参照	"figure"、"table"、"code-block" の番号書式。
<code>math_number_all</code>	真偽値	<code>false</code>	すべての独立行数式に自動で番号を付けます。
<code>math_eqref_format</code>	文字列	<code>"({number})"</code>	<code>:eq:</code> による数式参照の書式。
<code>mathjax_path</code>	文字列	URL	MathJax JavaScript の CDN URL またはローカルパス。

`numfig_format` の初期値:

```
[numfig_format]
figure = "Fig. %s"
table = "Table %s"
code-block = "Listing %s"
section = "Section %s"
```

診断と厳密な検査

表 11 診断の設定

キー	型	初期値	説明
<code>nitpicky</code>	真偽値	<code>false</code>	未解決の相互参照やリンク先を警告します。
<code>nitpick_ignore</code>	配列	<code>[]</code>	警告を除外する [type, target] の配列。
<code>suppress_warnings</code>	配列	<code>[]</code>	表示しない警告カテゴリのコード。
<code>keep_warnings</code>	真偽値	<code>false</code>	公開文書の本文に警告を含めます。

HTML 出力の設定

表 12 HTML の設定

キー	型	初期値	説明
html_theme	文字列	"guidedog"	HTML 生成に使うテーマ。
html_theme_options	テーブル	{}	HTML テーマに渡すキーと値。
html_title	文字列	自動生成	ブラウザーのタブに表示するタイトル。
html_short_title	文字列	自動生成	パンくずナビゲーションの短いタイトル。
html_logo	文字列	""	ソースディレクトリからのロゴ画像の相対パス。
html_favicon	文字列	""	ファビコンのパス。
html_static_path	配列	[]	出力の <code>_static/</code> にコピーするディレクトリ。
html_extra_path	配列	[]	変換せずに出力ルートにコピーするディレクトリ。
html_css_files	配列	[]	HTML ページが読み込む追加 CSS のファイル名。
html_js_files	配列	[]	HTML ページが読み込む追加 JavaScript のファイル名。
html_permaLinks	真偽値	true	段落と見出しに固定リンクを付けます。
html_permaLinks_icon	文字列	"¶"	固定リンクに使う記号や文字。
html_baseurl	文字列	""	サイトマップとメタデータの正規の基準 URL。
html_context	テーブル	{}	Jinja テンプレートに渡す追加変数の辞書。
html_additional_pages	テーブル	{}	追加ページ: <pre>{ "page_name" = 。 "template.html" }</pre>

PDF と Typst の設定

表 13 PDF の設定

キー	型	初期値	説明
<code>pdf_paper_size</code>	文字列	"a4"	用紙サイズ: "a4" または "us-letter"。
<code>pdf_logo</code>	文字列	" "	表紙に使うロゴ画像のパス。
<code>pdf_toplevel_sectioning</code>	文字列	"chapter"	本の最上位の区分: "chapter" または "part"。
<code>pdf_show_urls</code>	文字列	"no"	印刷版の URL 表示: "no"、"inline"、"footnote"。
<code>pdf_preamble</code>	文字列	" "	生成文書の先頭に挿入する Typst の生コード。
<code>pdf_font_paths</code>	配列	[]	追加の OTF/TTF フォントを探すディレクトリ。
<code>pdf_packages</code>	文字列	"download"	パッケージ解決の方針: "download" または "offline"。
<code>typst</code>	文字列	"typst"	システムの Typst CLI の名前または絶対パス。

PDF の本を定義する

`[[pdf_documents]]` テーブルで、このプロジェクトから作る本を一冊以上定義します。

```
[[pdf_documents]]
root = "index"
file = "guide.pdf"
title = "Field Notes Complete Guide"
author = "Author Name"

[[pdf_documents]]
root = "reference/index"
file = "reference.pdf"
title = "Field Notes Reference"
author = "Author Name"
```

MyST Markdown の設定

表 14 MyST Markdown の設定

キー	型	初期値	説明
myst_enable_extensions	配列	<code>["dollarmath"]</code>	構文拡張: "colon_fence"、"deflist"、 "dollarmath"、"fieldlist"、 "tasklist"、"substitution"。
myst_heading_anchors	整数	<code>0</code>	見出しからアンカーを作る深さ。 <code>0</code> で無効になります。
myst_substitutions	テーブル	<code>{}</code>	Markdown 文書で使う変数の置換。例: <code>{key = "value"}</code> 。
myst_url_schemes	配列	<code>["http", ...]</code>	外部リンクとして認識する URI スキーム。
myst_commonmark_only	真偽値	<code>false</code>	MyST 拡張を使わず、CommonMark の構文だけを解析します。

Intersphinx の設定

```

extensions = ["sphinx.ext.intersphinx"]
intersphinx_cache_limit = 5
intersphinx_timeout = 30

[intersphinx_mapping]
python = ["https://docs.python.org/3/", ""]
click = ["https://click.palletsprojects.com/", ["click.inv", ""]]

```

詳しい手順は [他のプロジェクトへのリンク](#) を参照してください。

Python autodoc の設定

表 15 Autodoc の設定

キー	型	初期値	説明
autodoc_source_paths	配列	<code>[]</code>	静的解析で Python モジュールを探すディレクトリ。
autoclass_content	文字列	<code>"class"</code>	クラスの docstring の取得元: "class"、"init"、"both"。
autodoc_member_order	文字列	<code>"alphabetical"</code>	メンバーの並び順: "alphabetical"、"bysource"、 "groupwise"。
autodoc_typehints	文字列	<code>"signature"</code>	型ヒントの表示位置: "signature"、"description"、 "none"。
autodoc_default_options	テーブル	<code>{}</code>	すべての <code>auto*</code> ディレクティブに適用する既定の設定。

検出規則の詳細は [autodoc による Python のドキュメント作成](#) を参照してください。

Odin API 文書の設定

表 16 Odin ドメインの設定

キー	型	初期値	説明
<code>odin_autoapi_dirs</code>	配列	<code>[]</code>	Odin パッケージを探すソースディレクトリ。
<code>odin_autoapi_root</code>	文字列	<code>"api"</code>	生成する Odin API 文書の出力サブディレクトリ。
<code>odin_autoapi_options</code>	配列	<code>["members", "undoc-members"]</code>	含めるメンバーのフィルター。
<code>odin_autoapi_member_order</code>	文字列	<code>"source"</code>	メンバーの並び順: <code>"source"</code> または <code>"alphabetical"</code> 。

Odin ドメインの詳しい使い方は [Odin の文書化](#) を参照してください。

国際化の設定

表 17 i18n の設定

キー	型	初期値	説明
<code>locale_dirs</code>	配列	<code>["locales"]</code>	gettext カタログを探すディレクトリ。
<code>gettext_compact</code>	真偽値	<code>true</code>	同じディレクトリの文書を一つのカタログにまとめます。
<code>gettext_uuid</code>	真偽値	<code>false</code>	POT メッセージに安定した UUID を付けます。
<code>gettext_auto_build</code>	真偽値	<code>true</code>	ビルド時に <code>.po</code> をバイナリの <code>.mo</code> にコンパイルします。
<code>figure_language_filename</code>	文字列	<code>"{root}.\n{language}\n{ext}"</code>	言語別の画像を選ぶファイル名テンプレート。

翻訳手順は [国際化](#) を参照してください。

リーダー、レンダラー、ビルダー

リーダーはソース言語を理解し、レンダラーは一つの文書木を書き、ビルダーはプロジェクトを調整して公開する。それぞれ役割が異なる。

表 18 ソースの処理経路

入力	リーダー	経路
プロジェクト内の RST	RST と組み込み Sphinx 層	プロジェクトグラフから HTML、Typst/PDF、テキスト、カタログを生成します。
プロジェクト内の Markdown	MyST 互換の Markdown 層	同じプロジェクトグラフとビルダーを使います。
<code>convert file.md</code>	既定は CommonMark	1 つの文書から HTML または Typst/PDF を生成します。
<code>convert file.typ</code>	ネイティブ Typst アダプター	Guidedoc の意味木の外で Typst をコンパイルします。

ビルダー

html

各文書の HTML ファイルと、索引・検索・インベントリを生成します。

dirhtml

ディレクトリ形式の文書 URL。

singlehtml

文書ごとの節とローカル参照を含む、1 つのメイン HTML ページです。

pdf

プロジェクトが定義する Typst の本を生成します。LaTeX ビルダーの別名もこの経路を選びます。

text

文書ごとのプレーンテキストファイル。

gettext

翻訳用のメッセージテンプレート。

dummy

成果物を公開せずにプロジェクトを読み、参照を解決します。

対応範囲は検証の証拠

`guidedog formats` は実行ファイルにあるアダプターを示す。互換性ページは固定テスト群と実プロジェクトの検証を記録する。CommonMark 合格は MyST 全対応を、RST 読み取りは全 Python 拡張の再現を意味しない。Sphinx との互換性を参照。

エラーメッセージ

診断は問題の名前、場所、役立つ次の操作を示すべきです。Guidedog はその情報を構造化データとして保持します。端末では色とソースの印を付けます。JSON は装飾なしでデータを保持します。

診断の読み方

```

UNDEFINED SUBSTITUTION                                rst.substitution.undefined

guide/page.rst:4: See |logo| here.
                   ^^^^^^

The substitution "|logo|" is not defined.
No output was written.

Add a definition, e.g. ".. |name| replace:: text".

```

診断は常に次の順序で、4つの部分からなります。

1. **タイトル行**。問題の種類を大文字で示します。警告と注記は `WARNING:` と `NOTE:` で始まります。右端の `rst.substitution.undefined` などは、検索に使える安定した診断コードです。
2. **場所**。ファイル、行番号、ソースの行と問題箇所の下印を示します。ディレクトリ不足やオプションなど、特定の行に関係しない問題ではパスだけ、または場所なしで表示します。
3. **何が起きたか**。問題を説明し、必要なら出力への影響も示します。この例では何も書き出していないことです。
4. **何をすべきか**。カラー端末では緑で示す、具体的な修正方法です。複数ある場合は、最も可能性の高い方法から列挙します。

色は文章で伝えた内容を強調するだけです。 `--color=never` で色を無効にし、 `--color=always` で端末以外の出力にも色を残せます。

次の操作を示すヒント

Guidedog 自身のメッセージにもこの規則を課す。ヒントは操作（「Add」「Pass」「Set」「Rename」「Run」）と、その対象を示す。例えば `--budget=MIB`、`conf.toml` の `root_doc`、ディレクティブのオプションだ。 `tests/hints_test.odin` は全メッセージを読み、空のヒント、本文の繰り返し、報告内容を示さない「report this」を拒否する。修正を直接示す提案（「Did you mean numfig?」）と、操作不要の注記「Nothing to do:」に理由を添える形式も認める。

Guidedog 自身の不具合なら、そのことを明示します。実行したコマンド、原因となった入力、 `guidedog --version` の出力を添えて報告するよう案内します。

プログラム向けの診断

`--diagnostics=json` は標準エラーに一行一個の JSON オブジェクトとして報告を書く。エディターや継続的インテグレーションから読める。

```

{"schema":1,"code":"rst.substitution.undefined","severity":"error",
 "title":"UNDEFINED SUBSTITUTION","message":"The substitution \"|logo|\" is not
 defined.",
 "hint":"Add a definition, e.g. \".. |name| replace:: text\".", "path":"guide/
 page.rst",
 "line":4,"column":5,"end_line":4,"end_column":11,"source":"See |logo| here.",
 "category":""}

```

ここでは折り返して示すが、実際は一オブジェクト一行。 `schema` はフィールドが変わったときだけ更新する形式の版で、他を読む前に確認する。列は1から始まる Unicode スカラー数で、バイト数や端末の桁数ではない。 `source`

は元の行全体。端末はエラー周辺だけを表示し、制御文字のエスケープと全角文字の幅を考慮して印を合わせる。category は suppress_warnings で使う警告分類、または空。

メモリが足りないとき

メモリ不足も診断として報告し、クラッシュしません。2 種類あります。

host.budget (メモリ予算の上限)

ビルドが保持可能なメモリ予算 (--budget=MIB、既定 1 GiB) に達した。報告は該当の工程を、テンプレートなら要求元の行も示す。端末では一度だけ、予算を増やすか、作業メモリをディスクに置いて続けるか、停止するかを尋ねる。端末以外では停止する。--budget または --memory=disk で事前に選べる。ヒントも予算を数値で示す。

```
Build again with --budget=256, or with --memory=disk to keep working memory
on
disk (slower); or split the work into smaller files.
```

提案値は、現在の保持量に拒否された工程の要求量を 4 倍して加えた量（工程は出力と書き込み用コピーも保持する）と旧予算の 2 倍の大きい方を、64 MiB 単位で切り上げる。工程が空きメモリに収まる場合はその空き量を超えない。収まらない場合はその旨を示し、まず --memory=disk を提案する。

```
Build again with --memory=disk to keep working memory on disk (slower):
the step
needs a budget of about 1408 MiB, more than the 900 MiB this machine has
free. Or
free that memory and build with --budget=1408, or split the document.
```

host.memory (メモリ不足)

予算内の要求をシステムが拒否した。命令はそのメモリを使わず停止する。ビルドは公開せず、GDS の変更は全部か無変更、convert は一ページ全部か無出力となる。メモリを解放するかスレッドを減らして (-j 1) 再実行する。公開を決定した後の切り替えはメモリを必要とせず、拒否で半公開にはならない。以前の未完了の公開を見つけても復旧計画のメモリがなければ、そのまま次のビルドに残す。

どちらもステータス 3 で終了します。

終了ステータス

各コマンドは次のいずれかのステータスで終了するため、スクリプトで失敗の種類を区別できます。

状態	意味
0	成功です。出力を書き込み、ビルドの場合は公開しました。
1	入力に問題があります。文書のエラー、方針による拒否 (--raw なしの raw コンテンツ)、読み取れない文書、または -W • --strict でエラー扱いになった警告です。

状態	意味
2	コマンドの使い方が不正です。不明なオプションやコマンド、値の不足・不正、 <code>conf.toml</code> や <code>-D</code> の不正な設定です。
3	制限に達しました。メモリ予算 (<code>--budget</code>)、入れ子の深さ (<code>--max-depth</code>)、ノード数 (<code>--max-nodes</code>)、スタック (<code>--stack-kib</code>)、大きすぎるソース、またはシステムが供給できないメモリです。
4	ファイルまたは外部プログラムの失敗です。読み書きできないパス、読み取り中に変わったファイル、Typst の起動・完了の失敗などです。
5	Guidedog 内部の失敗です。診断は不具合の報告を依頼します。
70	Guidedog がクラッシュすると、 <code>GUIDEDOG CRASHED (internal error)</code> と実行した命令、障害、 <code>guidedog --version</code> の結果を表示する。報告先はビルドが指定する課題管理、なければ提供者。ビルドは最後に公開するので、以前の出力は残る。
130	Ctrl+C で中断されました。公開済みのファイルは変わりません。

理由にかかわらずビルドが失敗すると、前の出力とビルド記録はそのまま残ります。問題を修正して再ビルドしてください。

Sphinx との互換性

作者の移行に役立つ部分では Sphinx に従い、機能ごとに互換性を検証します。Python の設定や任意の Python 拡張は実行しません。**ビルドの成功は、そのビルドが完了した証拠であり、完全な同等性の証明ではありません。**

プロジェクトと公開

表 20 プロジェクトの操作

操作	実装済みの動作
プロジェクト構成とビルドオプション 設定	Sphinx 形式のソース・出力ディレクトリとオプション： <code>-a -E -W -n -D -t -j -b -M -c -C</code> 。 Sphinx でおなじみの設定名を TOML データとして記述します。移行時はリテラルを変換し、計算が必要な値を報告します。
HTML ビルダー	<code>html</code> 、 <code>dirhtml</code> 、 <code>singlehtml</code> に対応します。単一ページのアンカーには文書名を含めます。
その他のビルダー	<code>text</code> 、 <code>gettext</code> 、 <code>dummy</code> に対応します。 <code>pdf</code> は Typst を使い、 <code>latex</code> と <code>latexpdf</code> も同じ経路を選びます。
テンプレート	Guidedog の Jinja 実装、型を保持する <code>html_context</code> 、レイアウトの継承、追加ページ。
差分ビルド	ソース、依存関係、テンプレート、設定、参照検索の変更で、関連する出力が無効になります。
公開	完全な世代を一度に公開します。ビルドに失敗した場合は前の公開済み世代を保持します。

`-D` の真偽値には `true` と `false`、または `1` と `0` を使えます。`index` がなく `contents` があるプロジェクトでは、警告付きで `contents` をルートにできません。追加テンプレートページは `dirhtml` でもサイトのルートに置きます。詳しい規約は [テンプレート](#) を参照してください。

変更のないページでも診断を再表示するため、`-w` は新規ビルドと変更のないビルドで同じ意味を持ちます。削除した文書のページは取り除きません。公開には復旧可能なジャーナルを使い、中断したコミットは次のビルドで処理します。不変条件とプラットフォーム上の制約は [完全な世代を公開する](#) を参照してください。

`epub`、`man`、`texinfo`、`linkcheck`、`doctest`、`coverage`、`changes` ビルダーはありません。指定すると修正方法を報告します。リポジトリの `tools/linkcheck` は独立した検証ツールです。

ソース形式

表 21 固定版による適合性の検証

リーダー	検証結果と制約
reStructuredText	98 個のコーパスソースで、比較対象の Docutils の木と識別子が一致します。ソース位置と一部の内部管理属性は比較から除外しています。
CommonMark 0.31.2	仕様の例 652 件がすべて通ります。
MyST 0.16.1 reference	リーダーのテストケース 218 件中 215 件、Sphinx のビルドケース 10 件中 8 件が一致します。残る差分は記録しています。

固定版でのテストであり、以後のすべての版との互換性を示すものではありません。`guidedog formats` で実行ファイルのリーダーとレンダラーの検証結果を確認できます。各リーダーのディレクトリにも適合性の注記があります。

`rst_prolog` は冒頭の書誌フィールドの後に、`rst_epilog` はソースの後に挿入します。Markdown にはどちらも挿入しません。`only` と `ifconfig` は条件を満たす節を残します。条件付きの節が周囲の階層を変える場合、配置が Sphinx と異なることがあります。MyST の `eval-rst` 内の節見出しはエラーです。

参照とドメイン

Guidedog は `toctree`、ラベル、用語集、節と図の番号付け、および `ref`、`doc`、`numref`、`term`、`download`、`any`、`keyword`、`option`、`envvar`、`token` 参照を実装しています。オブジェクトは目次ツリー、サイドバー、ページのアウトラインに表示できます。

標準、Python、C、C++、JavaScript、reStructuredText、数学のドメインを実装しています。Odin ドメインは Guidedog 独自の拡張で、パッケージ、宣言、シグネチャ、生成 API ページを記述します。ソースだけを使う探索の仕組みと制約は [Odin の文書化](#) を参照してください。

引数、戻り値、例外、変数のフィールドを構造化した説明にまとめます。正規名は参照とインベントリの別名になります。Python の既定値と注釈はソースの表記を保ちます。Sphinx は Python の unparser で再出力する場合があります。C++ 宣言は Sphinx 互換の識別子バージョンとシンボル一覧を公開します。入れ子の深さには制限があり、入れ子の括弧は線形時間で解析します。

`numref` は番号のプレースホルダーを 1 つ置換し、番号のない参照先を報告します。節は読む順に番号を付けます。各文書は一度だけ番号を付け、2 つ目の番号付き `toctree` は競合を報告します。書式の置換と識別子の詳細は参照テストを確認してください。

総合索引、モジュール索引、検索は組み込みです。検索は単語の前方一致を使い、Sphinx の英語ステミングは再現しません。

宣言的なオブジェクト型で、よく使われる Python 拡張登録を置き換えられます。`object_types`、`crossref_types`、`directive_aliases` がデータを記述します。独自の Python `parse_node` は名前・表示・プログラムの宣言規則に置き換えます。表せないコードは移行時に報告します。オブジェクト型の宣言を参照してください。

組み込み拡張の動作

表 22 拡張の範囲

動作	状態
<code>todo</code> , <code>ifconfig</code> , <code>extlinks</code> , <code>autosectionlabel</code>	組み込みです。直書きした <code>extlinks</code> のリンクにはロールへの置換を提案できます。
<code>graphviz</code>	リンクした Graphviz が静的な図を生成します。リソースの範囲制限を適用します。
<code>intersphinx</code>	ローカルと取得済みのインベントリに対応します。Windows は現在ローカルファイルのみ対応です。
<code>githubpages</code>	静的サイト用の公開ファイルを組み込みで生成します。
<code>mathjax</code> と <code>imgmath</code>	HTML の数式は MathJax を使います。PDF は対応する LaTeX の部分集合を Typst に変換します。
<code>myst_parser</code>	実装済みのプロジェクト用 Markdown 層です。
<code>doctest</code> ディレクティブ	内容は表示しますが、プロジェクトビルダーはテストを実行しません。
<code>autodoc</code>	パッケージをインポート・実行せず、Python ソースを読みます。

ソースベースの `autodoc` は、実行時に作られるすべてのオブジェクトを検出できません。コンパイル済みモジュール、外部デコレータ、計算された値、拡張のイベントフックには明確な制約があります。Python 3.13 と固定版の Sphinx 文書生成テストを基準としています。設定と例外の全体は `autodoc` による Python のドキュメント作成を参照してください。

`autosummary`、`napoleon`、`viewcode` と任意の Python 拡張はこの経路の対象外です。未対応の拡張を設定すると報告します。不明なディレクティブやロールはソース位置付きで報告します。**省かれた内容は公開前に必ず確認してください。**他のテーマ名も受け付けますが、Guidedog のテーマ実装を使います。

信頼とリソース制限

通常のビルドはプロジェクトの raw HTML・Typst、テンプレート、ネットワーク上のインベントリを許可します。ローカル読み取りはソースディレクトリと明示した共有ルート内に限ります。シンボリックリンクで範囲を広げることはできません。テンプレート、静的ファイル、include、画像、フォント、API ソースも同じ規則に従います。

Graphviz の画像リソースは文書を基準に解決し、読み取り範囲を制限します。対応する画像データは図に埋め込みます。`imagepath` と `fontpath` は範囲を広げません。Typst は読み取り可能なディレクトリを含む専用ルートで実行し、本のテンプレートとプリアンプルを宣言したリソース内に限定します。

`--untrusted` は読み取り範囲をソースディレクトリに限定します。raw コンテンツ、安全でない URL、外部リソース、プロジェクトの Typst テンプレートとプリアンプルを診断付きで省きます。インベントリや Typst パッケージは取得しません。リソースを読むグラフはコードとして表示し、警告します。このモードはネイティブコンパイラのメモリや実行時間を制限しません。

ホストの予算は既定で 1 GiB です。確保前に容量を予約します。ページの作業領域は使用後に解放しますが、保持するプロジェクトグラフとカタログは規模に応じてメモリを使います。読み取りワーカーが処理の受け入れを調整します。要求を拒否した場合は、失敗した操作と修正方法を報告します。

`--memory=ram` は管理予算に達すると停止します。`--memory=disk` は管理対象の作業ストレージにメモリマップした一時ファイルを許可します。保持するホストストレージは RAM 予算に含まれます。`--disk-budget` はマップ領域を制限し、ディスクの余裕も残します。非対話コマンドは回答を待ちません。既知の空き容量は `GUIDEDOG_AVAILABLE_MIB` で指定できます。

`--max-depth`、`--max-nodes`、`--stack-kib` は文書処理を制限します。値を変更すると読み取りと出力のキャッシュを無効にします。深さとスタック容量は整合させてください。ネイティブ Graphviz、tree-sitter、Typst の確保はホスト予算の対象外です。厳密な上限には OS の制限を使います。**エラーメッセージとメモリと所有権**を参照してください。

検証と実際のプロジェクト

`tools/sphinxdiff` は固定版の Sphinx テストルート 42 件で、ページ、識別子、本文リンク、インベントリ、番号付け、診断を比較します。保存した結果と README に意図的な差分を説明しています。正確なソース位置、安定した読み順の番号、対象形式専用の raw コンテンツを省いた際の明示的な報告などです。

2026 年 10 月 1 日のリモート検証では CPython、Django、Flask の新規ビルドと変更なしの HTML・PDF ビルドを実施しました。12 件すべてが成功し、診断とリンク結果は基準と一致しました。Linux テスト 1,039 件と対象を絞った AddressSanitizer テスト 94 件も通りました。

これらのプロジェクトには未対応の Python 拡張構文があり、その診断も検証結果の一部です。終了ステータスがゼロでも、独自のディレクティブがすべて再現されたとは限りません。診断を出さずに公開する必要がある場合は `-w` を使ってください。

過去の測定値とソースのリビジョンは `docs/manual/evidence/manuals.md` にあります。最新のリモートレビューは `build/review-remote-20261001/report.txt` です。測定値は記録した負荷を表しており、一般的な速度やメモリの上限ではありません。

内部の仕組み

この各章は実装の根拠と現在の契約を説明する。GDS は別のプロジェクトで判断とレビュー状態を残す。

Guidedoc エンジン

Guidedoc は変換のコアで、呼び出し側の記憶領域を借用する。ソースを読み、意味木を作り、パスを実行し、出力を測ってバイト列を生成する。ファイルを開いたり公開したりしない。

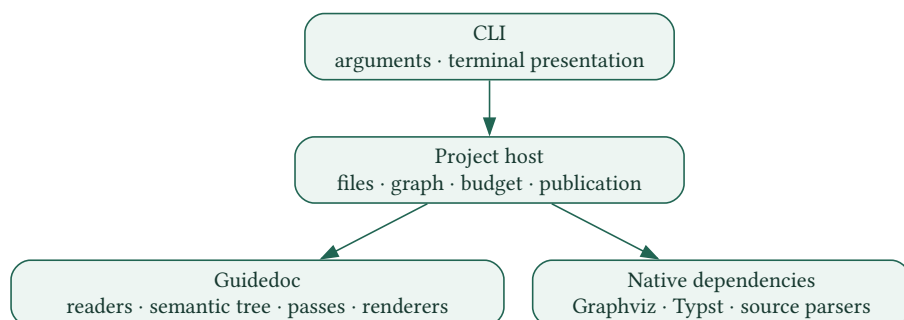


図 4 外部への作用はホストが担い、文書の意味は Guidedoc が担う。

境界

コア変換は割り当てを行わない。ノード列、テキスト、診断、出力の領域を呼び出し側が用意する。ホストは準備時に割り当ててもよく、第三者のコンパイラーには独自の規則がある。保証の範囲は変換境界であってプロセス全体ではない。

木構造

ノードは型付き ID を持つ。頻繁に使う列には種類と部分木の範囲を置き、それ以外の表に各種ノードのペイロードを置く。平坦な先行順表現なら、ヒープのポインター列をたどらず部分木全体を飛ばせる。

ノード i の排他的な部分木終端を e_i とする。部分木は $[i, e_i)$ を占める。葉では $e_i = i + 1$ 。親は子孫の後で閉じる。区間の証明は [アルゴリズムより先に不変条件](#) を参照。

API

コア API は明示的だ。領域の初期化、変換、結果の確認、リセットを行う。レジストリーはリーダー、パス、レンダラーの記述子を含む値で、実行時プラグイン探索やグローバル登録の副作用はない。少数の操作に絞り、他の Odin プログラムから使いやすくする。

所有権が要となる。ワークスペースのビューはリセットや再利用で無効になる。ホストが保持する結果は、セッション後も必要なテキストを所有しなければならない。正確なシグネチャは [core](#) を参照。独立した docs/api と同じ Odin ソースから生成する。マニュアル PDF は内部リンク付きで同じ本に収める。設計記録は docs/gds の GDS 0003。

アルゴリズムより先に不変条件

役に立つ不変条件は、実行中に保たれることと、終了後に呼び出し側が頼れることを示す。ループを書く前にそれを書く。

先行順走査の部分木区間

木を先行順で出力すると仮定する。各ノードは子より先に出力され、部分木は連続する。 e_i をノード i と全ての子孫の直後の添字とする。

$$\text{subtree}(i) = [i, e_i), \quad i < e_i \leq N. \quad (3)$$

命題。 添字を e_i に進めれば、部分木を飛ばせる。

証明。 連続性により全ての子孫は e_i より前にある。排他的な終端の定義により、以後のノードは e_i 以降にある。そこへ進めば部分木だけを飛ばせる。レンダラーが頼る前に、バリデーターが仮定を検査する。

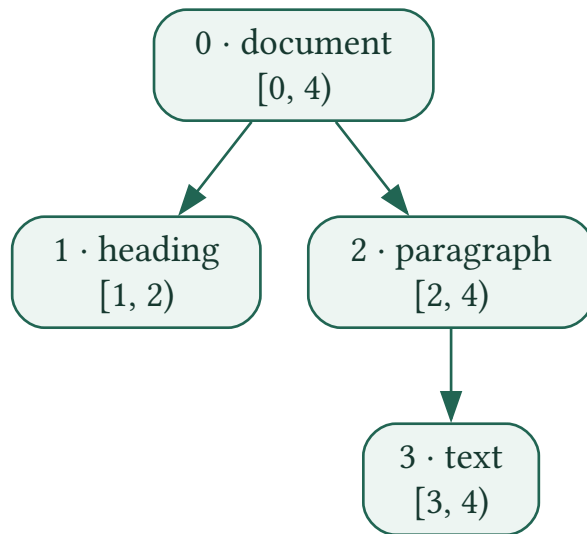


図 5 親の区間は子の区間を包含します。

全体走査は各ノードを一度訪れ、走査コストは $O(N)$ 。部分木のスキップは添字の更新一回で済むが、ペイロード処理には相応のコストがある。定数時間のスキップを定数時間の描画と混同しない。

出力の前に計測する

レンダラーはまずバイト数 m を測る。呼び出し側の出力容量を c とする。

$$m \leq c \implies \text{出力を開始できる。}$$

測定と出力の走査は同じ規則を使う。成功時は正確に m バイトを書く。失敗時に有効な部分成果物は返さない。公開はホストが担うため、不完全なバイト列が以前の出力を置き換えることはない。

キャッシュの有効性

キャッシュキーは結果を変え得る全データを含める。ソースだけでは足りない。インクルード、テンプレート、設定、参照検索、リソース、アダプターの版も出力に影響する。記録した依存関係が一致するときだけ再利用できる。キーが一致しても出力がなければ無効。

テストではコア境界に失敗するアロケータを置き、出力内容と比較する。インポートも監査する。割り当て数だけを数えて結果を見なければ、拒否で文書の一部を黙って捨てる不具合を見逃し得る。

完全な世代を公開する

読者に更新途中のサイトを見せてはならない。これは公開の性質であり、解析や描画とは別の問題だ。

不変条件

公開済みの出力木を P とする。準備中は P を変えず、次の木 Q を完成させてからコミットを決定する。ジャーナルは出力と対応する記録を指定する。

準備は失敗し得る。 ホストは未完成の世代を破棄する。**確定したコミットも中断し得る。** 次のビルドはジャーナルに基づいて完了または復元する。ステージング名から推測しない。

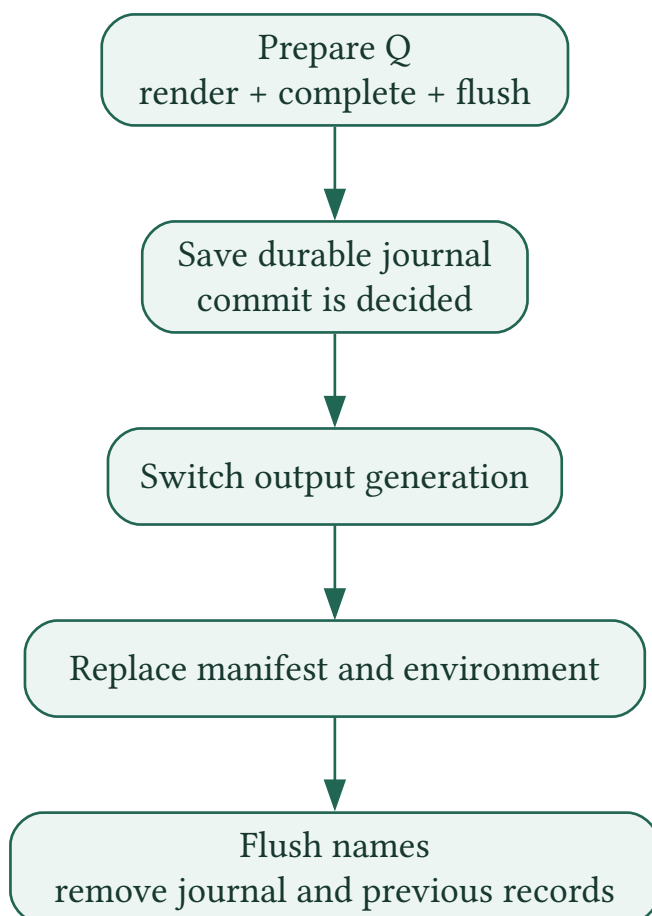


図6 永続化したジャーナルは、準備段階と確定したコミットを分ける。

キャッシュを出力先の外に置く理由

ディレクトリー交換は中の全てを動かす。ジャーナルを中に置くと、記録している操作自身で移動し、復旧時に位置が安定しない。Guidedog は公開木の外の安定した場所に記録を置く。そのため独自の出力には同階層のキャッシュを使う。

永続性と可視性

ファイル内容の永続化とディレクトリー名の永続化は別だ。ジャーナルを削除する前に両方を扱う。交換方法はプラットフォームで異

なり、復旧プロトコルはそれを考慮する。実装は `lib/project/commit.odin` と `lib/project/publish.odin`。

これはプロジェクトの公開を保護する。外部ホスティングとの分散トラザクションではない。成功したビルドをサーバーへコピーするのは、デプロイツールが担う別の操作だ。

メモリと所有権

予算は計上の契約である。計上の寿命と、それが賄う記憶領域の寿命が一致して初めて意味を持つ。

課金はストレージに伴う

割り当て前にバイト数を予約し、解放後に計上を解除する。バッファの置換中は新旧が共存し得るため、新しい割り当て全量を計上する。増分だけではピークを過小評価する。

置換時のピーク = その他の使用中メモリ + 旧バッファ + 新バッファ。

セッション所有の入力はセッションが回収する。呼び出し側所有の入力はその責任のまま。`read_owned` と `read_kept` はこの違いを明示する。短く読んでも既に確保した領域は小さくならない。

既知のゼロは不明ではない

メモリ調査では余裕がないと分かる場合と、余裕を調べられない場合がある。両者をゼロで表さず、既知かどうかのフラグで分ける。Linux では可視の祖先階層にある読み取り可能な cgroup 制限も適用する。ホスト全体の空きメモリだけではコンテナ内の実態を誤る。

診断は作業領域より長く残る

セッション後に示す報告は、見出し、ソース、修正案を所有し、解放済みのバッファを借用してはならない。内部の位置はバイトオフセット、利用者向けの列は Unicode スカラー数、端末の印は別途表示幅を考慮する。

残る境界

Guidedog は自身の管理する作業領域を計測する。Typst とネイティブ依存には独自の割り当てがあり、大きな PDF はホスト予算を大幅に超え得る。リモートの CPython 検証では Typst 込みのピーク RSS は約 14.5 GiB。これはその負荷の実測で、普遍的な上限ではない。予算到達時の選択は [エラーメッセージ](#) を参照。

このマニュアルについて

このマニュアルでは、実装済みの機能と意図的な制約を説明します。提案、決定、レビュー履歴は、別の GDS プロジェクトに記録します。組版を改善しても、提案の状態は変わりません。

実際の使い方に沿って構成しています。チュートリアルで学び、トピックガイドで理解し、手順ガイドで作業し、リファレンスで詳細を確認します。内部実装の章では、不変条件と証明を説明します。

小さな例を使い、前提を明示し、要点を直接述べます。読者が自分で確かめられる説明を目指しています。控えめな配色、明確な見出し、読みやすいセリフ体、使いやすいナビゲーションを採用しています。特定の著者の文章やレイアウトをまねるものではありません。

このプロジェクトは `guidedog quickstart` で作成し、Guidedog でビルドしています。本文は RST、図は Graphviz、数式は LaTeX の数学記法で記述します。Typst を使うのは PDF の組版テンプレートだけです。

ビルドの検証記録

2026-10-01 のリモート検証では、CPython、Django、Flask について HTML と PDF のクリーンビルドおよび増分ビルドを実行しました。12 個のコマンドがすべて成功し、診断とリンク検査の結果は既存の基準と一致しました。Linux のテスト 1,039 件と、対象を絞った sanitizer テスト 94 件が通りました。これは検証した版についての結果であり、Sphinx との完全互換性を示すものではありません。残る制約は Sphinx との互換性を参照してください。

ドキュメントの保守に使うコマンドは [この文書の保守](#) にまとめています。

Guidedog

[English](#) · [简体中文](#) · [日本語](#) · [한국어](#) · [हिन्दी](#) · [Español](#) · [Deutsch](#)

一度書く。明快地伝える。Web と紙に届ける。

Guidedog は reStructuredText と Markdown から HTML 文書サイトと PDF の本を作ります。Odin で実装され、変換エンジンは **Guidedoc** と呼ばれます。PDF は Typst で組版します。Python 環境や LaTeX は不要です。

執筆の流れは Sphinx を参考にしています。ただし Guidedog は独立したシステムです。設定は宣言的な TOML、API 抽出は静的解析で行い、Python の設定や拡張を実行しません。Sphinx との完全互換は目標ではありません。

プロジェクトを始める

```
guidedog quickstart docs
guidedog build html docs
guidedog build pdf docs
guidedog serve docs
```

プロジェクトには `conf.toml`、`index.rst`、`_templates/layout.html`、`_templates/book.typ`、`_static/` があります。テンプレートは普通の編集可能なファイルです。HTML にはナビゲーション、検索、コードの強調、図、数式があり、PDF には章、相互参照、図、索引、言語に応じた組版があります。

既定は明るいテーマで、暗いテーマやシステム設定も選べます。マニュアルの入口とリポジトリ概要ではブラウザの言語に合わせます。保存した選択が優先され、特定の版や章への直接リンクは尊重されます。

```
guidedog convert notes.md --to pdf --output notes.pdf
guidedog clean docs
guidedog migrate conf.py
```

インストール

各リリースには Linux (x86-64 と ARM64)、macOS (Apple シリコンと Intel)、Windows (x86-64) 向けのアーカイブがあります。どのアーカイブにも Typst、Graphviz、tree-sitter が入っているので、サイトや本のビルドにほかのもの

は要りません。Linux には glibc 2.35 以降と libcurl、macOS にはバージョン 12 以降が必要です。お使いのシステム向けのアーカイブと SHA256SUMS をダウンロードし、確認してからインストールします。

```
sha256sum -c --ignore-missing SHA256SUMS      # on macOS: shasum -a 256 -c --
ignore-missing SHA256SUMS
tar -xzf guidedog-linux-amd64.tar.gz
sudo install guidedog-linux-amd64/guidedog /usr/local/bin/
```

macOS では、公式の [Guidedog Homebrew tap](#) からインストールします。Homebrew が Apple シリコンまたは Intel 用のアーカイブを選びます。この tap は Linux にも対応しています。

```
brew tap insanai/guidedog
brew install insanai/guidedog/guidedog
guidedog --version
```

Windows ではアーカイブを展開し、そのフォルダーを PATH に加えます。Graphviz の DLL と config8 は guidedog.exe と同じ場所に置いたままにします。プログラムにはコード署名がありません。macOS でブラウザからダウンロードした場合は、`xattr -d com.apple.quarantine guidedog` で隔離属性を外してください。各プラットフォームの制限はリリースノートに書いてあります。

ソースからビルドする

Odin dev-2026-10 (tools/install-toolchain.sh odin DIR で取得と検証ができます)、C コンパイラー、Linux では libcurl、expat、zlib の開発パッケージを用意します。ネイティブライブラリーをビルドしてから Guidedog をビルドします。

```
native/tree-sitter/build.sh
native/graphviz/build.sh
tools/release/build.sh
```

できあがった build/guidedog は、PDF の生成にインストール済みの Typst 0.15.1 を使います。リリース版と同じように Typst コンパイラーを組み込むには、Rust と Cargo を用意して次を実行します。

```
native/typst_bridge/build.sh
tools/release/build.sh --release --typst-bridge
```

Windows では Git Bash と、Visual Studio の C++ ツール、LLVM を使います。Graphviz には Windows 向けの静的ビルドがないため、build.sh の代わりに native/graphviz/windows.sh を実行します。これは公式の Graphviz 16.1.0 の DLL をインストールし、tools/release/build.sh がそれを build/guidedog.exe の隣にコピーします。tools/release/suites.sh でテストスイートを実行できます。リリースの手順は [RELEASING.md](#) にあります。

Guidedog で翻訳する

```
mkdir -p docs/manual/_editions
```

```
build/guidedog build -b gettext docs/manual docs/manual/locales/templates -W
build/guidedog intl update -p docs/manual/locales/templates \
-l zh_CN -l ja -l ko -l hi -l es -l de docs/manual
build/guidedog intl stat -l zh_CN -l ja -l ko -l hi -l es -l de docs/manual
build/guidedog build -b html docs/manual docs/manual/_editions/de \
-D language=de -D 'html_extra_path=[]' -W
```

PO ファイルの `msgstr` を翻訳し、内容を確認します。Guidedog は POT 抽出、カタログのマージ、進捗表示、各版のビルドを行います。翻訳文を自動で作りません。コマンド、パス、コード、API 名、参照先は変えません。翻訳するのはマニュアルとリポジトリ概要のみで、API と GDS は英語のままです。各版と PDF のまとめ方は [ビルド手順](#) を参照してください。

GitHub では上のリンクから README の言語を選びます。言語の自動選択は生成した文書サイトで行います。

読む・組み込む

- ・ [マニュアル](#): チュートリアル、解説、作業手順、リファレンス、内部構造。
- ・ [API リファレンス](#): Odin ソースから生成し、マニュアル PDF の付録にも含めます。
- ・ [ライブラリガイド](#): 呼び出し元が所有するワークスペース、メモリ、変換例。
- ・ [テンプレート](#): Jinja で HTML、Typst で本を調整します。
- ・ [GDS](#): 番号付きの設計記録と実装上の判断。
- ・ [ベータレビュー](#): 修正、検証、対応範囲、公開前の残課題。

Guidedoc は呼び出し元のストレージを使い、構造化された診断を返します。ホストがファイル、予算、キャッシュ、公開、ネイティブコンパイラーを管理します。--untrusted は機能を制限しますが、プロセスのサンドボックスでも、すべてのネイティブ資源の上限でもありません。ベータ版の対象は macOS と Linux の信頼できる文書プロジェクトです。

謝辞とライセンス

Vikrant Rathore が制作し、Ronak Rathore と Kanak Rathore が開発を支援しました。Issue や Pull Request による貢献を歓迎します。文書の著者表記には名前やハンドルを使い、個人のメールアドレスは掲載しません。Git の識別情報は別に管理します。

Sphinx と Georg Brandl、Docutils と David Goodger、Typst、Tree-sitter と Max Brunsfeld、Graphviz、Ginger Bill と Odin コミュニティに感謝します。

Guidedog は [GNU Affero General Public License v3.0 以降](#) (AGPL-3.0-or-later) で公開しています。別のライセンス条件をご希望の場合は、作者にご連絡ください。連絡方法と、同梱フォントなど独自のライセンスが適用される第三者コンポーネントの一覧は [NOTICE](#) にあります。

追加の許可により、Guidedog を Graphviz とリンクして配布できます。詳しくは [NOTICE](#) をご覧ください。

API Reference

- `autodoc` – Package autodoc generates the reStructuredText that Sphinx’s sphinx.ext.autodoc generates (Sphinx 9.1), from Python source analysed by lib/pyscan instead of imported modules: the same directives (`⌈` ..
- `cli` – Package cli is the guidedog command: run parses a command line, does what it asks, and returns the exit status.
- `core` – Generated by tools/casefold from tools/data/CaseFolding-17.0.0.txt ; do not edit.
- `defaults` – Package defaults assembles the built-in readers and renderers into an explicit, immutable registry.
- `fetch`
- `gds` – Package gds is the GDS process of docs/gds (GDS 0001) as a library: it loads the discussion records of a directory into a Catalog, plans changes to them (a new draft, a promotion, a state or label change, the regenerated index) as a Plan, applies a plan as one recoverable transaction under a lock.
- `gettext` – Package gettext reads, writes, merges, and looks up GNU gettext message catalogs: PO and POT text files, and compiled MO files.
- `graphdog` – Package graphdog turns Graphviz DOT source into SVG (or PNG, or laid-out DOT) by calling the Graphviz C library in process.
- `highlight` – Generated by tools/pygments_styles from pygments-2.19.2’s style modules and sphinx-9.1.0’s pygments_styles.py; do not edit.
- `highlight/guidedoc` – Package highlight_guidedoc connects the highlight library to Guidedoc’s renderers.
- `host` – Package host composes the allocation-free core with memory, files, and retries.
- `i18n` – Package i18n translates Guidedoc documents the way Sphinx does with gettext: it extracts every translatable message of a document into a POT template (Sphinx’s gettext builder), and it replaces each message with its translation from a catalog (Sphinx’s Locale transform), re-parsing the translation as inline markup of the document’s own language.
- `inventory`
- `jinja`
- `odindoc` – Package odindoc reads Odin packages by static analysis, with Odin’s own parser (core:odin/parser), and writes the reStructuredText of the Guidedoc odin domain that documents them, as sphinx.ext.autodoc writes the Python domain’s: packages, procedures, procedure groups, types and

their fields or enumerators, constants, variables, and foreign imports, each with its doc comment turned into reST.

- `project` – Package `project` builds a documentation project the way Sphinx does: a source directory with `conf.toml`, builders writing `_build/<builder>`, and incremental builds.
- `pyscan` – Package `pyscan` analyses Python source statically, through tree-sitter: modules and their docstrings, `__all__`, and imports; classes with their bases, decorators, and members in source order; functions with their parameters as written; and the attribute documentation Sphinx’s `ModuleAnalyzer` finds (`#:` comments and docstrings after assignments).
- `readers/commonmark` – Package `commonmark` reads Markdown into a Guidedoc document without calling an allocator, as two readers with distinct ids: `「commonmark」` (`reader`, `read`, `read_inline`) is pure CommonMark 0.31.2 and takes no settings; `「myst」` (`myst_reader`, `read_myst`, `read_myst_inline`) is the MyST Markdown that Sphinx projects write (`myst-parser` 0.16.1): directives, roles, targets, comments, front matter, footnotes, GFM tables, and the optional syntax that `Settings.enable_extensions` turns on.
- `readers/rst` – Package `rst` reads reStructuredText as the Docutils markup specification defines it, implemented independently.
- `readers/rst/inline` – Package `rst_inline` parses reStructuredText inline markup and resolves references.
- `readers/sphinx` – Package `sphinx` reads reStructuredText with the Sphinx layer: Sphinx’s directives, roles, and domains (GDS `「Guidedoc: everything Sphinx does, the Odin way」`), implemented independently on the base reader’s extension interface.
- `render/html` – Package `html` renders a Guidedoc document as HTML without calling an allocator.
- `render/text` – Package `text_render` renders a Guidedoc document as plain text without calling an allocator, following the conventions of Sphinx’s text builder: paragraphs wrapped at 70 columns, titles underlined with `= - ~` and the other section characters, `「*」` and `「1」`.
- `render/typst` – Package `typst_render` encodes a validated Guidedoc document as one Typst source file.
- `tree-sitter` – Package `tree-sitter` is a thin binding of the tree-sitter C library and its Python grammar: parsers, trees, nodes, and cursors, and nothing more.
- `typst` – Package `typst` adapts the Typst toolchain for the host.

autodoc

Package `autodoc` generates the reStructuredText that Sphinx’s `sphinx.ext.autodoc` generates (Sphinx 9.1), from Python source analysed by `lib/pyscan` instead of imported modules: the same directives (`「.. py:class:: Name(signature)」` with `:module:`, `:canonical:`, `「Bases: ...」`), docstrings prepared as `prepare_docstring` prepares them, and members chosen, filtered, and ordered as `autodoc`’s options say. What static analysis cannot know (compiled modules, objects made at run time, members of classes outside the source roots) is reported as a note naming the object, never guessed silently.

It is a host library: it reads files through pyscan and allocates with the context allocator, freeing none of it, so a host runs it in an arena, the same as pyscan's: a Session and every Result live until the arena is freed.

Types

autodoc.Config :: struct

Config holds the conf.toml settings autodoc reads, with Sphinx's names and defaults (see `default_config`).

autoclass_content: string

「class」, 「init」, 「both」.

autodoc_class_signature: string

「mixed」, 「separated」.

autodoc_default_options: []Default_Option

autodoc_docstring_signature: bool

autodoc_inherit_docstrings: bool

autodoc_member_order: string

「alphabetical」, 「bysource」, 「groupwise」.

autodoc_mock_imports: []string

autodoc_preserve_defaults: bool

autodoc_typehints: string

「signature」, 「description」, 「none」, 「both」.

autodoc_typehints_description_target: string

「all」, 「documented」, 「documented_params」.

autodoc_typehints_format: string

「short」, 「fully-qualified」.

autodoc_type_aliases: []Type_Alias

autodoc_use_type_comments: bool

python_display_short_literal_types: bool

strip_signature_backslash: bool

tab_width: int

typehint_fields: bool

typehint_fields writes the fields of autodoc_typehints = 「description」 into the text (true by default); Sphinx adds them to the parsed field lists instead, so comparisons with its generated text turn it off.

autodoc.Default_Option :: struct

Default_Option is one autodoc_default_options entry: a value, or true for a flag.

name: string

value: string

flag: bool

the setting is true: the option is given without a value.

autodoc.Message :: struct

Message is a problem to report where the directive stands, Elm-style.

severity: Severity

code: string

title: string

text: string

hint: string

```
autodoc.Option :: struct
```

Option is a directive option as written; value is 「」 for a flag.

```
name: string
```

```
value: string
```

```
autodoc.Request :: struct
```

Request is one autodoc directive: its name (autoclass, automodule, ...), argument, options, content, and the reader's context (the current py:module and py:class).

```
directive: string
```

```
argument: string
```

```
options: []Option
```

```
content: []string
```

```
module: string
```

```
class: string
```

```
autodoc.Result :: struct
```

Result is the generated reST (lines without line breaks; empty when there is nothing to add) and the messages about it.

```
lines: [dynamic]string
```

```
messages: [dynamic]Message
```

```
autodoc.Session :: struct
```

Session is autodoc's state across the directives of one build: the analysed project, and the modules whose analyser annotations autodoc has merged into their classes (as `_ensure_annotations_from_type_comments` does the first time it documents a data or attribute from them), which changes the members found afterwards.

```
project: ^py.Project
```

```
merged: map[string]bool
```

```
autodoc.Severity :: enum u8
```

```
Note
```

```
Warning
```

```
Error
```

```
autodoc.Type_Alias :: struct
```

```
name: string
```

```
target: string
```

Procedures

```
autodoc.default_config :: proc() -> Config
```

default_config is Sphinx 9.1's defaults for the settings autodoc reads.

```
autodoc.document :: proc(session: ^Session, config: ^Config,  
request: Request) -> Result
```

document generates the reST of one autodoc directive, as AutodocDirective.run does before parsing it: the lines, and messages about them (an unknown directive or a bad option is an Error and no lines; what analysis cannot know is a note naming the object; never a panic). Modules are analysed through the session's project as needed; the result lives in the context arena.

```
autodoc.session_init :: proc(s: ^Session, project:  
^py.Project)
```

session_init starts a session over an analysed project, which it borrows for the session's life; its map lives in the context arena.

cli

Package cli is the guidedog command: run parses a command line, does what it asks, and returns the exit status. It is the command's whole surface; everything

else here is private. It allocates with `context allocator` and never frees, so the command runs it in an arena (`cmd/guidedog`), which frees everything at once.

Procedures

```
cli.run :: proc(args: []string) -> int
```

`run` runs the command line args (without the program name) and returns the exit status: 0 on success, and on failure the status the problem names, after reporting it on standard error. It allocates with `context allocator`, an arena the caller frees, through a `host.Guard`: when that allocator refuses memory, the command changes no record and ends with `host.memory` (exit 3), reported without allocating.

```
cli.version_text :: proc(buffer: []u8) -> string
```

`version_text` writes into `buffer` what `guidedog --version` prints before the Typst compiler it finds, the versions built into the program, and returns it. It allocates nothing, so the crash report is prepared with it at start (`cmd/guidedog`).

Constants

```
cli.FULL_VERSION :: VERSION + VERSION_SUFFIX
```

`FULL_VERSION` is what the command prints as its version: `VERSION` and `VERSION_SUFFIX`.

```
cli.LICENSE_NOTICE
```

`LICENSE_NOTICE` is what `guidedog --licenses` prints before the notices of the libraries linked in.

```
cli.VERSION :: "0.2.0"
```

`VERSION` is the Guidedog release this program belongs to, the one version ``guidedog --version``, `guidedog --help`, and the crash report name. The file `VERSION` at the repository root holds the same text for release tooling, and `version_test.odin` fails when the two differ. The Guidedoc engine's own version (`core.VERSION`) and document schema are reported beside it; they are separate because programs embed the engine without this command.

```
cli.VERSION_SUFFIX :: #config(GUIDEDOG_VERSION_SUFFIX, "")
```

`VERSION_SUFFIX` is build metadata appended to `VERSION`, for development builds such as `-define:GUIDEDOG_VERSION_SUFFIX=+3fe1d8a` (Semantic Versioning's 「+」 form). A release build has none.

core

Generated by tools/casfold from tools/data/CaseFolding-17.0.0.txt ; do not edit.

Types

```
core.Admonition_Kind :: enum i32
```

Generic

Attention

Caution

Danger

Error

Hint

Important

Note

Tip

Warning

Seealso

Todo

```
core.Align :: enum u8
```

None

Left

Center

Right

```
core.Anchor_Extra :: struct
```

Anchor_Extra is the id of an element a renderer generates for a node:

- an entry of a table of contents: owner is the Contents, item the Section;
- a footnote of target-notes: owner is the directive, item the first target (or anonymous reference) with the URI it lists, number its footnote number;
- the footnote reference target-notes adds after a reference: owner is the reference, item the directive, note the item of its footnote, number the same.

```
owner: Node_Id
```

```
item: Node_Id
```

```
note: Node_Id
```

```
span: u32
```

```
number: i32
```

```
core.Anchor_Ids :: struct
```

```
span: u32
```

```
core.Anchors :: struct
```

Anchors gives nodes stable, unique identifiers: HTML ids and Typst labels. Every renderer and the project host build the same table, so internal links agree across targets and the labels a project records are the ids its pages publish. The ids are the ones Docutils gives a document read as Sphinx reads it (auto_id_prefix 「id」), so a project that moves from Sphinx keeps its links and bookmarks:

- An explicit Id attribute is used verbatim (terms, equations, domain objects).
- Every element Docutils names gets one id when Docutils』 parser or transforms name it, in the same order (anchors_order.odin): the id made from its name by make_id, or, when the name gives none or only an id already used, the next 「idN」 . Elements without a name (footnote references, anonymous targets) always take the next 「idN」 .
- Targets and names Docutils does not write (external and indirect targets, names of embedded URIs) still take their id, so later ids are numbered as Docutils numbers them, but no node carries it.
- A label (an internal block Target) gives its id to the element it names, after the element's own ids.

- Elements a renderer generates have ids too (anchor_extras): the entries of a table of contents and the footnotes of target-notes with their references.

A node may have several ids: anchor_of returns the primary one, and anchor_ids iterates all of them for renderers that emit the others as aliases.

index: []u32

per node: its first span, or NONE.

spans: []Text

ranges of bytes.

next: []u32

per span: the node's next span, or NONE.

count: int

bytes: Buffer

prefix: string

prefix_length: int

the bytes the prefix takes before an id.

names: Symbol_Table

every id: the node that carries it, or NONE.

counter: int

labels: []u32

per node: the first id labels give it, until they follow its own.

own: []u8

room for the longest name a node carries by itself.

extras: []Anchor_Extra

sorted by owner, then in the order the ids were given.

extra_count: int

```
core.Arg_Kind :: enum u8
```

None

Int

Span

Static

```
core.Attribute :: struct
```

key: Attribute_Key

name: Text

the key text when key is Custom.

value: Text

```
core.Attribute_Key :: enum u8
```

Custom

Class

Id

Name

Language

Alt

Width

Height

Scale

Align**Format****Title****Target****Start****Widths****Source****core.Attribute_Range** :: *struct***first**: u32**count**: u32**core.Attributes** :: *struct***next**: u32**end**: u32**core.Buffer** :: *struct*

Buffer is a bounded byte accumulator over caller-provided bytes. The first write that does not fit sets overflow, and every later write is dropped, so the bytes are always a prefix of what was written; a caller that sees overflow must report it. wanted counts every byte asked for, stored or not: writing into a zero Buffer measures, and scratch_buffer then takes exactly that much.

bytes: []u8**used**: int**wanted**: int**overflow**: bool


```
core.Builder :: struct
```

Builder appends nodes in preorder. Open nodes form a bounded stack; closing a node patches its subtree end. The first failure poisons the builder: later calls do nothing, so an ignored capacity failure cannot produce a truncated document. The engine gives a reader or pass a builder over one bank for one stage: text is copied into the bank's pool, ids and Text values index that bank, and nothing a builder returns outlives the stage unless the stage succeeds. Adapters change nodes through these procedures; b.snap is visible so a reader can decode text straight into the pool after text_used, adding what it wrote to text_used before text_close.

```
snap: ^Snapshot
```

```
open: Stack(Node_Id)
```

```
reports: ^Reports
```

```
limits: Limits
```

```
status: Status
```

```
need: Capacity_Need
```

```
work: u64
```

```
cancel: Cancel_Probe
```

```
core.Cancel_Probe :: struct
```

```
procedure: proc(user: rawptr) -> bool
```

```
user: rawptr
```

```
core.Capacity_Need :: struct
```

```
region: Region
```

```
minimum: u64
```

```
exact: bool
```

```
core.Cell_Info :: struct
```

```
row_span: u16
```

```
col_span: u16
```

```
align: Align
```

```
core.Children :: struct
```

```
next: u32
```

```
end: u32
```

```
core.Conformance :: enum u8
```

```
Planned
```

```
Partial
```

```
Conformant
```

```
core.Content :: enum u8
```

```
Nothing
```

```
Blocks
```

```
Inlines
```

```
Structure
```

children are restricted to specific kinds; see `child_allowed`.

```
core.Content_Policy :: enum u8
```

```
Refuse
```

an error that names the construct and the policy.

```
Omit
```

a warning; the content is left out.

Allow

trusted content is emitted.

```
core.Diagnostic :: struct
```

```
using msg: Message
```

```
span: Source_Span
```

```
args: [2]Diagnostic_Arg
```

```
core.Diagnostic_Arg :: struct
```

Diagnostic_Arg is an integer, a quoted source span, or static text.

```
kind: Arg_Kind
```

```
int: i64
```

```
span: Source_Span
```

```
static: string
```

```
core.Document :: struct
```

Document is a checked handle to one build of a snapshot bank of a workspace. It borrows the workspace and is valid until the next conversion, reset, or rebuild of its bank, whichever comes first; document_valid tells.

```
ws: ^Workspace
```

```
generation: u32
```

```
revision: u32
```

the bank's revision when the document was built.

```
bank: u8
```

```
core.Document_Reading :: enum i32
```

Document_Reading is a Document's `value`: who read it, which decides the ids the transforms after reading give (anchors_order.odin). The Sphinx layer marks what it reads, as Sphinx adds transforms of its own to Docutils. .

Docutils

Sphinx

```
core.Emit_Mode :: enum u8
```

A renderer has one encoding procedure. The engine runs it in Measure mode, with reports enabled, to check support and count bytes; then in Write mode to store them.

Measure

Write

```
core.Emitter :: struct
```

Emitter is where a renderer writes. It never fails: Measure mode counts bytes, and Write mode stores them in the caller's output, which the engine sized by measuring.

mode: Emit_Mode

out: []u8

written: int

source_map: []Map_Entry

mapped: int

```
core.Highlighter :: struct
```

Highlighter writes highlighted HTML for a code block. It belongs to the caller's trust boundary and may allocate, but it must write the same bytes for the same input every time, because the renderer measures before it writes. It returns false to let the renderer write plain escaped code instead.

```
procedure: proc(user: rawptr, language, code: string, e: ^Emitter) -> bool
```

```
user: rawptr
```

```
core.INVALID_NODE :: Node_Id(max(u32))
```

```
core.Label :: enum u8
```

Label names a word or phrase a renderer writes by itself, not taken from the document, or that a reader writes into it (the group names of Sphinx’s info fields). A host translates them (Sphinx ships them in sphinx.po) and passes them in Render_Options.labels, and to a reader in its own options. Labels holding 「%s」 take a value there (a version number).

```
Attention
```

```
Caution
```

```
Danger
```

```
Error
```

```
Hint
```

```
Important
```

```
Note
```

```
Tip
```

```
Warning
```

```
See_Also
```

```
Todo
```

```
Contents
```

```
Version_Added
```

Version_Changed

Version_Deprecated

Version_Removed

Section_Author

Module_Author

Code_Author

Link_Heading

Link_Definition

Link_Image

Link_Table

Link_Code

Link_Equation

Link_Term

Parameters

Variables

Raises

Returns

Return_Type

Return_Values

Template_Parameters**Throws****Arguments****Availability**

```
core.Labels :: [Label]string
```

```
core.Limits :: struct
```

```
max_depth: int
```

nested blocks and inline markup.

```
max_nodes: int
```

```
max_work: u64
```

reader and resolution steps.

```
max_include_depth: int
```

```
max_included_bytes: int
```

```
max_expansion: int
```

substitution nesting.

```
stack_bytes: int
```

stack_bytes bounds the machine stack a recursive reader or renderer uses; o means DEFAULT_STACK_BYTES. The conversion thread needs this much more stack (stack.odin).

```
core.Line_View :: struct
```

Line_View is one line of a source as readers hand it to each other: its text without the line break, and the source offset of text[o]. The reST and Markdown readers share it, so nested content passes between them as it is, without converting or casting.

```
text: string
```

offset: u32

core.Link_Info :: struct

destination: Text

title: Text

name: Text

normalized reference name, when named.

resolved: Node_Id

meaningful only when kind is Internal.

kind: Link_Kind

core.Link_Kind :: enum u8

External

destination is a URL or path.

Email

destination is an address; HTML adds mailto:.

Internal

resolved is a node in this document.

Unresolved

name awaits resolution; an error, or a warning and problematic text

Project

under Policy.keep_documents_with_errors.

destination is a project anchor key in another document (books).

core.List_Info :: struct

style: List_Style


```
delimiter: u8
```

『.』 , 『)』 , or 『(『 for 『(1)』 ; bullet character for bullets.

```
start: i32
```

```
core.List_Style :: enum u8
```

```
Bullet
```

```
Arabic
```

```
Lower_Alpha
```

```
Upper_Alpha
```

```
Lower_Roman
```

```
Upper_Roman
```

```
core.Map_Entry :: struct
```

Map_Entry records that generated output from offset on came from span.

```
offset: u32
```

```
span: Source_Span
```

```
core.Math_Style :: enum u8
```

Math_Style is how HTML writes math for a script (MathJax, KaTeX) to typeset.

```
Plain
```

class 「math」 , the text in \(\ \) or \[\], as Docutils writes it.

```
Sphinx
```

as Sphinx's mathjax extension: its classes, and displayed math in parts

```
core.Media_Kind :: enum u8
```

Media_Kind is what an image node shows. As in Docutils, an image directive naming a video file plays it; Guidedoc treats audio files the same way. Renderers decide from the path.

Image

Video

Audio

```
core.Message :: struct
```

Message is a static description of one problem. Packages declare them as constants, so a report never formats or copies text.

code: string

severity: Severity

status: Status

title: string

message: string

「{0}」 and 「{1}」 name the arguments.

hint: string

```
core.Node :: struct
```

kind: Node_Kind

flags: Node_Flags

subtree_end: u32

value: i32

text: Text

payload: u32

span: Source_Span

attributes: Attribute_Range

core.Node_Flag :: *enum* u8

Tight

List: items render without paragraph spacing.

Anonymous

Reference or Target: anonymous hyperlink.

Auto

Footnote or reference: auto-numbered.

Symbol

Footnote or reference: auto-symbol.

Local

Contents: only the enclosing section.

Header

Row or Cell: belongs to a header.

Referenced

Target or Footnote: something refers to it.

Generated

Created by resolution rather than markup.

core.Node_Flags :: *bit_set*[Node_Flag; u8]

core.Node_Id :: *distinct* u32

```
core.Node_Kind :: enum u8
```

Document

Title

Block structure.

Subtitle

Section

Heading

Paragraph

Block_Quote

Attribution

List

List_Item

Definition_List

Definition_Item

Term

Classifier

Definition

Field_List

Field

Field_Name

Field_Body**Option_List****Option_Item****Option_Group****Option****Option_Argument****Option_Description****Line_Block****Line****Code_Block****Preformatted****Doctest_Block****Math_Block****Raw_Block****Thematic_Break****Table****Table_Head****Table_Body****Row**

Cell**Figure****Caption****Legend****Admonition****Topic****Sidebar****Rubric****Container****Compound****Footnote****Citation****Target****Comment****Substitution_Definition****Contents****Directive****Toctree**

Sphinx layer (GDS 「Guidedog: everything Sphinx does, the Odin way」).

Toctree_Entry

Only**Glossary****Version_Modified****Desc****Desc_Signature****Desc_Content****Hlist****Production_List****Production****Index****Text**

Inline content.

Emphasis**Strong****Code****Link****Reference****Image****Footnote_Reference****Citation_Reference**

Substitution_Reference

Inline_Target

Math

Raw_Inline

Soft_Break

Hard_Break

Subscript

Superscript

Title_Reference

Abbreviation

Role

Xref

Sphinx layer inlines.

Desc_Name

Desc_Addname

Desc_Annotation

Desc_Parameterlist

Desc_Parameter

Desc_Optional

Desc_Returns

```
core.Pass :: struct
```

A pass reads an immutable document and may rebuild it into the destination builder.

```
name: string
```

```
procedure: proc(user: rawptr, doc: Document, b: ^Builder)
-> Pass_Outcome
```

```
user: rawptr
```

```
core.Pass_Outcome :: enum u8
```

```
Unchanged
```

```
Rebuilt
```

```
Failed
```

```
core.Policy :: struct
```

```
raw: Content_Policy
```

raw HTML, raw Typst, raw LaTeX.

```
unsafe_urls: Content_Policy
```

javascript:, vbscript:, and non-image data: URLs.

```
math: Content_Policy
```

math the target cannot translate.

```
includes: bool
```

whether include directives may request resources.

```
strict: bool
```

warnings fail the conversion.

```
keep_documents_with_errors: bool
```

keep_documents_with_errors keeps a document whose content has errors, as Docutils does below its halt level: an unknown directive or role,

an unresolved reference, or a bad option is still reported as an error, but the construct is left out (or rendered as problematic text) and the status is Ok. Renderers report such problems as warnings. Errors that leave no usable tree still fail: capacity, limits, cancellation, invalid storage, and validation. See `content_error`.

```
core.ROOT :: Node_Id(0)
```

```
core.Read_Context :: struct
```

```
ws: ^Workspace
```

```
b: ^Builder
```

```
source: Source_Id
```

```
options: Read_Options
```

```
policy: Policy
```

```
limits: Limits
```

```
resources: Resource_Provider
```

```
reports: ^Reports
```

```
scratch: ^Scratch
```

```
included: int
```

bytes of included sources, charged against `max_included_bytes`.

```
core.Read_Extension :: struct
```

`Read_Extension` carries a reader's own settings (`rst.Settings`, `sphinx.Config`, `commonmark.Settings`, or a third-party reader's type): a pointer the caller owns and the type it points to. `read_extension` makes one, and a reader takes it back with `reader_settings`, which refuses settings of another type, so settings meant for one reader are never read as another's. The zero value means the reader's defaults.

```
data: rawptr
```

id: typeid

```
core.Read_Inline_Proc :: #type proc(ctx: ^Read_Context, span:
Source_Span) -> Status
```

Read_Inline_Proc parses a span of ctx.source as inline content of the builder's open node, with the reader's settings (ctx.options.extension), as a translated message is parsed. It neither resolves references nor reports unresolved ones: the caller owns the surrounding document. span.source must be ctx.source.

```
core.Read_Options :: struct
```

promote_title: bool

RST: a lone top-level section becomes the title.

tab_width: int

RST tab stops; zero means 8.

today: string

RST date substitutions, e.g. 2026-09-29.

extension: Read_Extension

the reader's own settings; zero: its defaults.

```
core.Read_Proc :: #type proc(ctx: ^Read_Context) -> Status
```

Read_Proc reads ctx.source into ctx.b and returns its status; see Reader.

```
core.Read_Result :: struct
```

using result: Result

document: Document

```
core.Reader :: struct
```

id: string

name: string

version: string

conformance: Conformance

read: Read_Proc

read_inline: Read_Inline_Proc

optional; see Read_Inline_Proc.

<code>core.Region :: enum u8</code>

None

Nodes

Text

Attributes

Lists

Links

Cells

Sources

Scratch

Reports

Output

Source_Map

Second_Bank

the workspace has no bank B, which passes that rebuild write.

<code>core.Registry :: struct</code>

readers: []Reader

```
renderers: []Renderer
```

```
core.Render_Context :: struct
```

```
view: View
```

```
doc: Document
```

```
out: ^Emitter
```

```
reports: ^Reports
```

```
options: Render_Options
```

```
policy: Policy
```

```
limits: Limits
```

```
scratch: ^Scratch
```

```
core.Render_Options :: struct
```

```
standalone: bool
```

a complete document rather than a body fragment.

```
reference_profile: bool
```

CommonMark reference HTML, for conformance tests.

```
title: string
```

overrides the document's own title.

```
language: string
```

BCP 47 tag; 「en」 when empty.

```
stylesheet: string
```

standalone HTML: CSS emitted in a style element.

```
section_numbers: bool
```

anchor_prefix: string

prepended to every anchor, e.g. a document ID in a book.

permalinks: bool

HTML: a link to each heading, caption, and signature.

permalink_icon: string

「¶」 when empty.

secnumber_suffix: string

after section numbers; 「.」 when empty.

highlighter: Highlighter

labels: ^Labels

the words renderers write, translated; nil: English.

math: Math_Style

math_inline: [2]string

HTML math delimiters for the script that typesets it, with Math_Style.Sphinx; empty: $(\)$ inline and $[\]$ displayed, as Sphinx's mathjax_inline and mathjax_display default to.

math_display: [2]string

```
core.Render_Proc :: #type proc(ctx: ^Render_Context) ->
Status
```

Render_Proc writes ctx.view with ctx.out and returns its status; see Renderer.

```
core.Renderer :: struct
```

id: string

name: string

version: string

media_type: string

conformance: Conformance

render: Render_Proc

core.Reports :: struct

items: []Diagnostic

used: int

suppressed: int

truncated: bool

muted: bool

write mode repeats the checked encoding without reporting.

primary: Diagnostic

has_primary: bool

errors: int

warnings: int

fatal: Status

the first error that is not a content error; see content_error.

core.Request :: struct

source: Source

reader: string

renderer: string

read: Read_Options

render: Render_Options

policy: Policy**limits:** Limits**passes:** []Pass**resources:** Resource_Provider**cancel:** Cancel_Probe**source_map:** []Map_Entry

optional; filled by renderers that support it.

```
core.Resource_Provider :: struct
```

Resource_Provider belongs to the caller's trust boundary and may allocate. The core passes the requesting source and the name as written; the provider returns borrowed UTF-8 text valid for the whole conversion, and a canonical name for cycle checks.

```
procedure: proc(user: rawptr, from: Source, name:
string) -> (text: string, canonical: string, status:
Resource_Status)
```

user: rawptr

```
core.Resource_Status :: enum u8
```

Ok**Missing****Refused****Limit**

the host's memory budget cannot hold it.

```
core.Result :: struct
```

Result is the outcome of a stage. reports borrows the workspace's reports region, valid until the next conversion or reset; primary is the first error by value, so it survives both. On Capacity, need names the region to enlarge.


```
status: Status
```

```
written: int
```

```
reports: []Diagnostic
```

```
primary: Diagnostic
```

```
need: Capacity_Need
```

```
mapped: int
```

source_map entries written.

```
suppressed: int
```

```
truncated: bool
```

```
core.Scratch :: struct
```

```
bytes: []u8
```

```
used: int
```

```
high: int
```

high-water mark, for sizing evidence.

```
core.Scratch_Mark :: distinct int
```

Scratch is stage-local storage carved from the caller's byte slice. Taking scratch advances a used-count; it is not an allocator call. A stage records a mark and releases to it when it finishes.

```
core.Severity :: enum u8
```

```
Note
```

```
Warning
```

```
Error
```

```
core.Slab_Plan :: struct
```

```
nodes: int
```

```
attributes: int
```

```
lists: int
```

```
links: int
```

```
cells: int
```

```
text: int
```

```
scratch: int
```

```
sources: int
```

```
reports: int
```

```
output: int
```

```
second_bank: bool
```

```
core.Snapshot :: struct
```

```
nodes: Table(Node)
```

```
attributes: Table(Attribute)
```

```
lists: Table(List_Info)
```

```
links: Table(Link_Info)
```

```
cells: Table(Cell_Info)
```

```
text: []u8
```

```
text_used: int
```

```
core.Snapshot_Storage :: struct
```

```
nodes: []Node
```

```
attributes: []Attribute
```

```
lists: []List_Info
```

```
links: []Link_Info
```

```
cells: []Cell_Info
```

```
text: []u8
```

```
core.Source :: struct
```

```
name: string
```

```
text: string
```

```
core.Source_Entry :: struct
```

```
using source: Source
```

```
parent: Source_Id
```

the including source; itself for the main source.

```
included_at: Source_Span
```

the include directive, when included.

```
use: Source_Use
```

how an included source is read.

```
core.Source_Id :: distinct u16
```

```
core.Source_Span :: struct
```

Source_Span is a half-open byte range in a source of the workspace source table.

source: Source_Id

start: u32

end: u32

core. Source_Use :: <i>enum</i> u8

Source_Use tells how the reader uses an included source: parsed as markup (include), or shown as it is (literalinclude, a graph file). Hosts can tell a document's parts from files it only quotes, as Sphinx's env.included does.

Parsed

Literal

core. Stack :: <i>struct</i> (\$T: <i>typeid</i>)

Stack is a bounded LIFO over caller-provided items.

items: []T

used: int

core. Stack_Guard :: <i>struct</i>

base: uintptr

limit: uintptr

core. Status :: <i>enum</i> u8

Ok

Invalid_Input

Unsupported

Policy

Capacity

Limit**Cancelled****Internal****core.Storage** :: struct

Storage lists every region a workspace borrows. All lengths are capacities.

banks: [2]Snapshot_Storage**scratch:** []u8**sources:** []Source_Entry**reports:** []Diagnostic**core.Symbol** :: struct

Symbol_Table is open addressing over caller-provided slots. It never grows. The load factor stays at or below one half, and probes are bounded, so hostile keys cost a bounded amount of work and end in a limit report rather than a slow scan.

hash: u64**key:** string

borrowed; must outlive the table.

value: u32**used:** bool**core.Symbol_Status** :: enum u8**Ok****Duplicate****Full**

Probe_Limit**core.Symbol_Table** :: struct**slots**: []Symbol**count**: int**max_probe**: int**core.Table** :: struct(\$T: typeid)

Table is a caller-owned SoA slice with a used-count. Appending never allocates.

rows: []T**used**: int**core.Text** :: struct

Text is a byte range in the active snapshot's text pool.

start: u32**length**: u32**core.View** :: struct

View is the read-only form renderers and passes traverse. Obtaining a view checks the handle once; accessors are then plain bounds-checked indexing. Strings they return borrow the snapshot's text pool or the sources, valid while the document is valid. An id out of range is a programming error (a bounds-check panic), not a data error: ids come from the view itself.

snap: ^Snapshot**sources**: []Source_Entry**core.Workspace** :: struct

Workspace is one conversion's borrowed storage. A document handle names a generation (one conversion, from read to render) and a bank's revision (one

build of that bank); both only grow, so a handle to a cleared or rebuilt bank never becomes valid again.

banks: [2]Snapshot

revisions: [2]u32

raised whenever a bank is cleared to be built again.

active: int

generation: u32

scratch: Scratch

sources: []Source_Entry

sources_used: int

reports: Reports

keep_errors: bool

the read's Policy.keep_documents_with_errors; passes honour it.

output: []u8

optional; set by init_workspace_from_bytes for convert_text.

ready: bool

Procedures

```
core.add_attribute :: proc(b: ^Builder, id: Node_Id, key:
Attribute_Key, value: Text, name := Text{})
```

add_attribute requires that id is the most recently begun node, so each node's attributes stay contiguous.

```
core.add_attribute_string :: proc(b: ^Builder, id: Node_Id,
key: Attribute_Key, value: string, name := "")
```

add_attribute_string copies value (and name, for a Custom key) into the text pool and adds the attribute as add_attribute does.

```
core.add_flags :: proc(b: ^Builder, id: Node_Id, f:
Node_Flags)
```

add_flags adds flags to node id; nothing happens once the builder has failed.

```
core.add_leaf :: proc(b: ^Builder, kind: Node_Kind, span:
Source_Span, text := Text{}) -> (Node_Id, bool) #optional_ok
```

add_leaf adds a closed node of kind with text (borrowed from the bank's pool) and no children; it fails as begin_node does.

```
core.add_source :: proc(ctx: ^Read_Context, name, text:
string) -> (Source_Id, bool)
```

add_source registers text that is not a file, such as a configured prologue, so nodes parsed from it carry spans into it. The text must outlive the conversion.

```
core.add_text :: proc(b: ^Builder, s: string) -> Text
```

add_text copies s into the text pool.

```
core.admonition_label :: proc(o: Render_Options, k:
Admonition_Kind) -> string
```

admonition_label is the title of an admonition kind; 「」 for a generic admonition.

```
core.admonition_name :: proc(k: Admonition_Kind) -> string
```

admonition_name returns the static class name of an admonition kind (「note」, 「tip」).

```
core.anchor_extras :: proc(a: ^Anchors, owner: Node_Id) -
> []Anchor_Extra
```

anchor_extras returns the ids generated for a node's sake (see Anchor_Extra), in the order they were given.

```
core.anchor_ids :: proc(a: ^Anchors, id: Node_Id) ->
Anchor_Ids
```

anchor_ids iterates every id of a node, the primary one first.

```
core.anchor_of :: proc(a: ^Anchors, id: Node_Id) -> (string,
bool)
```

anchor_of returns the primary id of a node, borrowed from the anchor table, or false when the node has none.


```
core.arg_int :: proc(v: i64) -> Diagnostic_Arg
```

arg_int makes an integer diagnostic argument.

```
core.arg_span :: proc(s: Source_Span) -> Diagnostic_Arg
```

arg_span makes a diagnostic argument that quotes the source text of s.

```
core.arg_static :: proc(s: string) -> Diagnostic_Arg
```

arg_static makes a diagnostic argument of static text, which must outlive the reports.

```
core.attribute :: proc(v: View, id: Node_Id, key: Attribute_Key) -> (string, bool)
```

attribute returns the value of node id's first attribute with key, borrowed from the view, or false when it has none.

```
core.attributes :: proc(v: View, id: Node_Id) -> Attributes
```

attributes starts an iteration over every attribute of node id, for next_attribute.

```
core.begin_node :: proc(b: ^Builder, kind: Node_Kind, span: Source_Span) -> (Node_Id, bool) #optional_ok
```

begin_node opens a node of kind as the last child of the current node, to be closed with end_node. It fails, poisoning the builder and returning INVALID_NODE, when the nesting is illegal (child_allowed), a limit is reached, or the nodes region is full.

```
core.buffer_string :: proc(b: ^Buffer) -> string
```

buffer_string returns the bytes written, borrowed from the buffer's storage.

```
core.buffer_write :: proc(b: ^Buffer, s: string)
```

buffer_write appends s, or sets overflow and drops it, and every later write, when it does not fit; wanted counts it either way.

```
core.buffer_write_byte :: proc(b: ^Buffer, c: u8)
```

buffer_write_byte appends one byte, as buffer_write does.

```
core.build_anchors :: proc(ctx: ^Render_Context, a: ^Anchors, headings: bool) -> Status
```

build_anchors takes its storage from the render context's scratch. Identifiers are made from text in the snapshot's pool, and folding never more than

doubles its length, so twice the pool plus a fixed overhead per anchor always suffices. A node's own name is gathered in room measured for the longest one.

```
core.builder_capacity :: proc(b: ^Builder, region: Region,
minimum: u64)
```

`builder_capacity` poisons the builder with a Capacity status whose need names region and the minimum it lacks, reporting CORE_CAPACITY; nothing happens if it already failed.

```
core.builder_fail :: proc(b: ^Builder, d: Diagnostic)
```

`builder_fail` reports `d` and poisons the builder with its status (the first failure wins); later builder calls do nothing.

```
core.builder_finish :: proc(b: ^Builder) -> Status
```

`builder_finish` closes the root. Every other node must already be closed.

```
core.builder_init :: proc(b: ^Builder, snap: ^Snapshot,
scratch: ^Scratch, reports: ^Reports, limits: Limits,
root_span: Source_Span) -> Status
```

`builder_init` starts a builder over `snap` with its open-node stack taken from `scratch`, and opens the root Document node at `root_span`. The engine calls it for every read and pass; tests use it to drive a reader's pieces. It returns Capacity, with `b.need` set, when `scratch` cannot hold the stack. The builder borrows every argument for one stage.

```
core.cell_info :: proc(v: View, id: Node_Id) -> Cell_Info
```

`cell_info` returns the cell payload of node `id`, or a cell spanning one row and one column when it has none.

```
core.check_render :: proc(ws: ^Workspace, doc: Document,
renderer: Renderer, request: Request) -> Result
```

`check_render` runs the renderer in measure mode and reports its exact output size, as need, and every support problem, without writing. `doc` must be a valid document of `ws`.

```
core.child_count :: proc(v: View, id: Node_Id) -> int
```

`child_count` counts the children of node `id`, in time linear in their number.

```
core.children :: proc(v: View, id: Node_Id) -> Children
```

`children` starts an iteration over the children of node `id`, for `next_child`.

```
core.code_caption :: proc(v: View, id: Node_Id) -> Node_Id
```

code_caption is a code block's Caption child (its parsed Custom caption), or INVALID_NODE when it has none.

```
core.content_error :: proc(status: Status) -> bool
```

content_error tells whether an error's status describes the content (an unknown directive or role, a bad option, an unresolved reference, content refused by policy) rather than a tree that cannot be used: capacity, limits, cancellation, or a bug. Policy.keep_documents_with_errors keeps documents whose errors are all content errors.

```
core.content_model :: proc(k: Node_Kind) -> Content
```

content_model tells what children nodes of kind k may have; see child_allowed. The switch is exhaustive, so a new kind does not compile until it says.

```
core.contents_scope :: proc(v: View, contents: Node_Id) ->
(first, end: Node_Id, base: int)
```

contents_scope is the range of nodes a table of contents lists and the number of sections around that range.

```
core.convert :: proc(ws: ^Workspace, registry: Registry,
request: Request, output: [u8]) -> Result
```

convert reads, optionally transforms, and renders one source into output, which the caller owns; on success output[:written] holds it, and on failure written is 0. It starts a new workspace generation, which invalidates every earlier document, even on failure. It checks the registry first (validate_registry). The result borrows the workspace: its reports are valid until the next conversion or reset. It never allocates; every failure is a status with a primary diagnostic.

```
core.convert_text :: proc(ws: ^Workspace, registry: Registry,
reader, renderer, text: string, name := "input") -> (string,
Result)
```

convert_text converts text with default_request and the workspace's own output region (set by the slab initializers; convert with an output slice otherwise). The returned string borrows that region and is valid until the next conversion or reset of the workspace; on failure it is 「」 and the result says why.

```
core.copy_payload :: proc(b: ^Builder, v: View, from, to: Node_Id)
```

copy_payload copies a node's list, cell, or link row, with its texts, to the node to of b's document; passes that rebuild a document share it.

```
core.copy_subtree :: proc(b: ^Builder, v: View, id: Node_Id)
-> Node_Id
```

copy_subtree copies a node and its descendants from a document view into a builder, including text, attributes, and payload rows, and returns the copy of id. Copies are appended in order, one per node, so the copy of a descendant d is copy + (d - id): a pass can adjust copies with set_value, set_text, add_flags, and set_kind. Internal link targets keep their numbering only when the pass copies the whole document in order. On failure the builder is poisoned and the result is INVALID_NODE.

```
core.current :: proc(b: ^Builder) -> Node_Id
```

current returns the innermost open node, or INVALID_NODE when none is open.

```
core.current_kind :: proc(b: ^Builder) -> Node_Kind
```

current_kind returns the kind of the innermost open node; Document when none is open.

```
core.custom_attribute :: proc(v: View, id: Node_Id, name: string) -> (string, bool)
```

custom_attribute returns the value of node id's Custom attribute named name, borrowed from the view, or false when it has none.

```
core.decode_last_rune :: proc(s: string, end: int) -> (rune, int)
```

decode_last_rune decodes the UTF-8 character ending at byte end of s, as decode_rune.

```
core.decode_rune :: proc(s: string, i: int) -> (rune, int)
```

decode_rune decodes the UTF-8 character starting at byte i of s and returns it with its width; an invalid byte decodes as the replacement character, width 1.

```
core.default_limits :: proc() -> Limits
```

default_limits returns the documented limits by value; zero limits never mean unlimited.

```
core.default_policy :: proc() -> Policy
```

default_policy returns the safe policy by value: raw content, unsafe URLs, and math the target cannot translate are refused; includes may be requested.

```
core.default_read_options :: proc() -> Read_Options
```

default_read_options returns the read options by value: RST title promotion on, tab width 8, and no reader settings.

```
core.default_request :: proc(name, text, reader, renderer:
string) -> Request
```

default_request assembles a complete Request with the documented defaults: default_read_options, a standalone render, default_policy, and default_limits, with no passes, resource provider, or cancellation probe. It borrows name and text. Change fields afterwards; lib/defaults has typed read options for the built-in readers.

```
core.diagnostic :: proc(m: Message, span: Source_Span, a :=
Diagnostic_Arg{}, b := Diagnostic_Arg{}) -> Diagnostic
```

diagnostic builds a Diagnostic by value from a static message, a span, and up to two arguments; it copies nothing.

```
core.document_valid :: proc(doc: Document) -> bool
```

document_valid reports whether doc still names the snapshot it was built as: the same generation, the active bank, and the bank's revision at that time.

```
core.dump_tree :: proc(v: View, e: ^Emitter)
```

dump_tree writes a readable, deterministic outline of a document. Tests use it as a renderer-independent golden format:

```
document
  section value=1
    heading value=1
      text "Title"
```

```
core.emit :: proc(e: ^Emitter, s: string)
```

emit writes s: Measure mode counts it, Write mode also stores it in the output.

```
core.emit_byte :: proc(e: ^Emitter, c: u8)
```

emit_byte writes one byte, as emit does.

```
core.emit_int :: proc(e: ^Emitter, value: i64)
```

emit_int writes value in decimal, as emit does.

```
core.emit_mark :: proc(e: ^Emitter, span: Source_Span)
```

emit_mark records that output from here on originates at span.

```
core.emit_rune :: proc(e: ^Emitter, r: rune)
```

emit_rune writes r UTF-8 encoded, as emit does.

```
core.encode_rune :: proc(r: rune) -> ([4]u8, int)
```

encode_rune returns r encoded as UTF-8 and the number of bytes used.

```
core.end_node :: proc(b: ^Builder, id: Node_Id)
```

end_node closes id, which must be the innermost open node, and records its subtree end; closing out of order poisons the builder.

```
core.extension_as :: proc(e: Read_Extension, $T: typeid) ->
(settings: ^T, ok: bool)
```

extension_as returns the settings e carries as ^T: nil and true when e is empty, nil and false when it carries another type. It neither reports nor allocates.

```
core.extra_anchor :: proc(a: ^Anchors, owner, item: Node_Id)
-> (string, bool)
```

extra_anchor returns the id of an element generated for owner and item.

```
core.extra_text :: proc(a: ^Anchors, e: Anchor_Extra) ->
string
```

extra_text is the id of a generated element.

```
core.find_reader :: proc(r: Registry, id: string) -> (Reader,
bool)
```

find_reader returns the reader of r with id, by value, or false.

```
core.find_renderer :: proc(r: Registry, id: string) ->
(Renderer, bool)
```

find_renderer returns the renderer of r with id, by value, or false.

```
core.first_child :: proc(v: View, id: Node_Id) -> Node_Id
```

first_child returns the first child of node id, or INVALID_NODE when it has none.

```
core.first_problem :: proc(r: ^Reports) -> Diagnostic
```

first_problem returns the primary error, or the first warning when there is none.

```
core.has_children :: proc(v: View, id: Node_Id) -> bool
```

has_children tells whether node id has any child.

```
core.has_class :: proc(v: View, id: Node_Id, class: string) -> bool
```

has_class looks through every class attribute; a value may hold several classes.

```
core.id_fold :: proc(r: rune) -> string
```

id_fold is the ASCII transliteration make_id gives a lowercase rune (「é」 gives 「e」, 「²」 gives 「2」), from Unicode's decompositions, or 「」 when it has none. The string is static.

```
core.include_source :: proc(ctx: ^Read_Context, from: Source_Id, name: string, span: Source_Span, use := Source_Use.Parsed) -> (Source_Id, bool)
```

include_source asks the host for a resource named by a directive in source from, records it in the source table, and returns its ID. Every refusal is reported at span. use says whether the reader parses the text or shows it as it is.

```
core.init_workspace :: proc(ws: ^Workspace, storage: Storage) -> Result
```

init_workspace borrows storage. It checks alignment-free invariants: every region is disjoint, and bank A can hold at least a root node. Initializing a workspace again invalidates every document of its earlier storage.

```
core.init_workspace_from_bytes :: proc(ws: ^Workspace, slab: []u8) -> Result
```

init_workspace_from_bytes carves the regions of slab_plan(estimate) from slab, one bank, for the largest input estimate whose plan fits, and initializes the workspace. The workspace borrows slab, which must outlive it; a slab too small for any plan is a capacity result. For passes that rebuild, use init_workspace_with_plan.

```
core.init_workspace_with_plan :: proc(ws: ^Workspace, slab: []u8, p: Slab_Plan) -> Result
```

init_workspace_with_plan carves exactly the regions of p from slab, and the rest of the slab becomes scratch. It is the authoritative path:

`init_workspace_from_bytes` only chooses a plan. A plan that cannot be sized is `Invalid_Input`; a slab smaller than `plan_bytes(p)` is a capacity result.

```
core.is_ascii_alnum :: proc(c: u8) -> bool
```

`is_ascii_alnum` tells whether `c` is an ASCII letter or digit.

```
core.is_ascii_digit :: proc(c: u8) -> bool
```

`is_ascii_digit` tells whether `c` is an ASCII digit.

```
core.is_ascii_punct :: proc(c: u8) -> bool
```

`is_ascii_punct` tells whether `c` is ASCII punctuation, as `CommonMark` defines it.

```
core.is_ascii_space :: proc(c: u8) -> bool
```

`is_ascii_space` tells whether `c` is ASCII white space: space, tab, LF, CR, FF, or VT.

```
core.is_unicode_punctuation :: proc(r: rune) -> bool
```

`is_unicode_punctuation` follows `CommonMark 0.31`: general categories P and S.

```
core.is_unicode_whitespace :: proc(r: rune) -> bool
```

`is_unicode_whitespace` follows `CommonMark`: Zs characters, tab, LF, FF, and CR.

```
core.kept_message :: proc(policy: Policy, m: Message) -> Message
```

`kept_message` is `m` as a renderer reports it: a content error becomes a warning when the policy keeps documents with errors, so the page is still written.

```
core.kind :: proc(v: View, id: Node_Id) -> Node_Kind
```

`kind` returns the kind of node `id`.

```
core.kind_name :: proc(k: Node_Kind) -> string
```

`kind_name` returns the static name of a node kind, as dumps and diagnostics write it.

```
core.label :: proc(o: Render_Options, l: Label) -> string
```

`label` returns a label in the render's language, or in English when it has none.


```
core.line_bounds :: proc(text: string, offset: int) ->
(start, end: int)
```

line_bounds returns the byte range of the line containing offset, without the break.

```
core.line_column :: proc(text: string, offset: int) -> (line,
column: int)
```

line_column converts a byte offset into a 1-based line and byte column.

```
core.link_info :: proc(v: View, id: Node_Id) -> (Link_Info,
bool)
```

link_info returns the link payload of node id, or false and an unresolved link when it has none.

```
core.list_info :: proc(v: View, id: Node_Id) -> List_Info
```

list_info returns the list payload of node id, or a bullet list starting at 1 when it has none.

```
core.load_object :: proc(ws: ^Workspace, bytes: []u8, limits:
Limits) -> Read_Result
```

load_object rebuilds a document from bytes written by write_object into bank A of a new workspace generation. bytes must outlive the document: source names and texts are views into them.

```
core.make_id :: proc(out: ^Buffer, name: string)
```

make_id turns a name into an identifier matching `[a-z](-?[a-z0-9]+)*`: letters are lowercased and transliterated to ASCII (ID_FOLDS, derived from Unicode data), every other run of characters becomes one hyphen, and leading digits and hyphens are dropped, as published Sphinx URLs expect. The result may be empty.

```
core.media_kind :: proc(path: string) -> Media_Kind
```

media_kind reads a path's extension, ignoring case, a query, and a fragment.

```
core.next_anchor_id :: proc(a: ^Anchors, it: ^Anchor_Ids) ->
(string, bool)
```

next_anchor_id returns the next id of an anchor_ids iteration, borrowed from the anchor table; false ends it.

```
core.next_attribute :: proc(v: View, it: ^Attributes) ->
(Attribute, bool)
```

next_attribute returns the next attribute of an attributes iteration, whose Text values index the view's pool; false ends it.

```
core.next_child :: proc(v: View, it: ^Children) -> (Node_Id,
bool)
```

next_child returns the next child of a children iteration; false ends it.

```
core.nfd_first :: proc(s: string) -> rune
```

nfd_first returns the first character of s in NFD, as Sphinx takes an index entry's letter: unicodedata.normalize(「NFD」, s[o])[o].

```
core.no_reader_settings :: proc(ctx: ^Read_Context) -> bool
```

no_reader_settings is reader_settings for a reader that takes no settings: any extension is reported (CORE_SETTINGS_NONE) and the result is false.

```
core.node :: proc(v: View, id: Node_Id) -> Node
```

node returns node id by value.

```
core.node_count :: proc(v: View) -> int
```

node_count returns the number of nodes in the document, the root included.

```
core.node_text :: proc(v: View, id: Node_Id) -> string
```

node_text returns the text of node id, borrowed from the view.

```
core.normalize_label :: proc(out: ^Buffer, label: string)
```

normalize_label writes the case-folded label with runs of white space collapsed to one space and the ends trimmed. It is used for CommonMark labels and RST names.

```
core.normalize_nfd_into :: proc(dst: []u8, s: string) -> (n:
int, ok: bool) #optional_ok
```

normalize_nfd_into writes s in Normalization Form D into dst, the caller's storage, and returns the length of the whole normal form. It never allocates and never borrows s: the result is always dst[:n]. When dst is too small, ok is false, n is the size dst needs, and dst holds only a prefix; a nil dst measures.

```
core.offset_in :: proc "contextless" (outer, inner: string) -> (int, bool)
```

offset_in returns where inner starts in outer when inner is a slice of outer's bytes, and false otherwise (an empty inner included). It compares addresses only and reads neither string, so it is safe for any two strings.

```
core.parent_of :: proc(v: View, id: Node_Id) -> Node_Id
```

parent_of scans backwards; it is linear and meant for diagnostics, not traversal.

```
core.plain_text :: proc(v: View, id: Node_Id, out: ^Buffer)
```

plain_text writes the concatenated text of a subtree, as used for alt text and ids.

```
core.plan_bytes :: proc(p: Slab_Plan) -> (bytes: int, ok: bool) #optional_ok
```

plan_bytes returns the slab size a plan needs. ok is false when a capacity is negative or the size overflows, which init_workspace_with_plan reports.

```
core.pool_string :: proc(b: ^Builder, t: Text) -> string
```

pool_string returns the text t of the builder's pool; it borrows the bank for the stage.

```
core.pop :: proc(s: ^Stack($T)) -> (v: T, ok: bool)
```

pop removes and returns the top item; false when the stack is empty.

```
core.push :: proc(s: ^Stack($T), v: T) -> bool
```

push appends v; false, changing nothing, when the stack is full.

```
core.read_document :: proc(ws: ^Workspace, reader: Reader, request: Request) -> Read_Result
```

read_document parses the request's source into bank A of a new generation and validates it. The document borrows the workspace and the request's source text, which must outlive it; it is valid until the next conversion, reset, or rebuild of bank A, and is returned only on success.

```
core.read_extension :: proc(settings: ^$T) -> Read_Extension
```

read_extension wraps a pointer to a reader's settings with their type. It borrows settings, which must outlive the read.

```
core.reader_settings :: proc(ctx: ^Read_Context, $T: typeid)
-> (settings: ^T, ok: bool)
```

reader_settings is extension_as at a reader's entry: settings of another type are reported (CORE_SETTINGS_TYPE, naming both types' packages) and ok is false; the reader then returns Invalid_Input before reading anything. nil and true mean the defaults.

```
core.render_document :: proc(ws: ^Workspace, doc: Document,
renderer: Renderer, request: Request, output: []u8) -> Result
```

render_document checks and measures, then writes into the caller's output only if it fits; otherwise the result is Capacity with the exact size needed, and output is untouched. doc must be a valid document of ws, and stays valid.

```
core.report :: proc(r: ^Reports, d: Diagnostic)
```

report records a diagnostic. The first error is kept by value in primary even when storage is full, so capacity failures remain explainable.

```
core.report_message :: proc(r: ^Reports, m: Message, span:
Source_Span, a := Diagnostic_Arg{}, b := Diagnostic_Arg{})
```

report_message reports the message m at span with its arguments, as report does.

```
core.reset_workspace :: proc(ws: ^Workspace)
```

reset_workspace invalidates every document and clears all used-counts.

```
core.resolve_link :: proc(b: ^Builder, row: u32, target:
Node_Id, kind: Link_Kind, destination := Text{})
```

resolve_link records the resolution of a link row during a reader's resolve stage.

```
core.scratch_buffer :: proc(s: ^Scratch, n: int) -> (b:
Buffer, missing: u64, ok: bool)
```

scratch_buffer takes a Buffer of n bytes from scratch, usually the wanted count of a measuring pass into a zero Buffer; ok is false, with the bytes missing, when scratch cannot hold it. The caller releases the scratch.

```
core.scratch_mark :: proc(s: ^Scratch) -> Scratch_Mark
```

scratch_mark records how much scratch is taken, for scratch_release.

```
core.scratch_release :: proc(s: ^Scratch, m: Scratch_Mark)
```

scratch_release gives back every scratch item taken since m; slices taken after m must not be used afterwards.

```
core.scratch_take :: proc(s: ^Scratch, $T: typeid, n: int) ->
(items: []T, missing: u64, ok: bool)
```

scratch_take returns n zeroed elements of T, or ok = false with the bytes missing.

```
core.set_cell :: proc(b: ^Builder, id: Node_Id, info:
Cell_Info)
```

set_cell stores info as node id's cell payload; a full cells region poisons the builder with Capacity.

```
core.set_kind :: proc(b: ^Builder, id: Node_Id, k: Node_Kind)
```

set_kind changes the kind of node id, as a pass does to a copy; validation checks the result. Nothing happens once the builder has failed.

```
core.set_link :: proc(b: ^Builder, id: Node_Id, info:
Link_Info) -> u32
```

set_link stores info as node id's link payload and returns its row, for resolve_link, or NONE when the builder has failed or the links region is full (Capacity).

```
core.set_list :: proc(b: ^Builder, id: Node_Id, info:
List_Info)
```

set_list stores info as node id's list payload; a full lists region poisons the builder with Capacity.

```
core.set_span_end :: proc(b: ^Builder, id: Node_Id, end: u32)
```

set_span_end moves the end of node id's source span, as a reader learns where a construct ends; nothing happens once the builder has failed.

```
core.set_text :: proc(b: ^Builder, id: Node_Id, t: Text)
```

set_text sets the text of node id; nothing happens once the builder has failed.

```
core.set_value :: proc(b: ^Builder, id: Node_Id, v: i32)
```

set_value sets the value of node id (a heading level, a list count, and so on); nothing happens once the builder has failed.

```
core.slab_plan :: proc(input_bytes: int, second_bank :=
false) -> (p: Slab_Plan, ok: bool) #optional_ok
```

slab_plan estimates capacities for an input of the given size. The estimate is generous for prose and typical technical documents; tables and heavy markup may need more, which a capacity result reports. ok is false when a capacity would overflow.

```
core.source_entry :: proc(v: View, id: Source_Id) ->
Source_Entry
```

source_entry returns the source-table entry of id (the main source is o), borrowed from the view, or a zero entry when id is out of range.

```
core.source_of :: proc(ctx: ^Read_Context, id: Source_Id) -
> string
```

source_of returns the text of source id, borrowed from the workspace's source table for the read.

```
core.source_text :: proc(v: View, span: Source_Span) ->
string
```

source_text returns the source bytes a span covers, or 「」 for an invalid span.

```
core.spend :: proc(b: ^Builder, steps: u64 = 1) -> bool
```

spend charges work against limits.max_work and polls cancellation periodically.

```
core.split_label :: proc(s: string) -> (before, after:
string)
```

split_label splits a label at its 「%s」: the text before and after the value.

```
core.stack_diagnostic :: proc(g: Stack_Guard, span:
Source_Span) -> Diagnostic
```

stack_diagnostic is the Limit a guard's recursion reports.

```
@(require_results) core.stack_exhausted :: proc "contextless"
(g: Stack_Guard) -> bool
```

stack_exhausted reports whether the next frame would take the stack past the budget. A zero guard (never started) is never exhausted.

```
@(require_results, no_sanitize_address) core.stack_guard ::  
proc "contextless" (limits: Limits) -> Stack_Guard
```

stack_guard starts measuring at the caller's frame, with the limits budget.

```
core.stack_slice :: proc(s: ^Stack($T)) -> []T
```

stack_slice returns the items in the stack, bottom first, borrowed from its storage.

```
core.subtree_end :: proc(v: View, id: Node_Id) -> Node_Id
```

subtree_end returns the id after the last descendant of node id.

```
core.symbol_get :: proc(t: ^Symbol_Table, key: string) ->  
(value: u32, found: bool)
```

symbol_get returns the value stored for key, or NONE and false when it is absent.

```
core.symbol_put :: proc(t: ^Symbol_Table, key: string, value:  
u32) -> (existing: u32, status: Symbol_Status)
```

symbol_put inserts key. For a duplicate it leaves the table unchanged and returns the existing value.

```
core.symbol_set :: proc(t: ^Symbol_Table, key: string, value:  
u32) -> bool
```

symbol_set replaces the value of an existing key.

```
core.symbol_table_init :: proc(t: ^Symbol_Table, s: ^Scratch,  
capacity: int) -> (missing: u64, ok: bool)
```

symbol_table_init takes $2 * \text{capacity}$ slots (rounded to a power of two) from scratch.

```
core.text :: proc(v: View, t: Text) -> string
```

text returns the string t names in the snapshot's text pool, borrowed from the view.

```
core.text_close :: proc(b: ^Builder, start: u32) -> Text
```

text_close returns the Text pushed since start (text_open), or an empty Text when the builder has failed.

```
core.text_open :: proc(b: ^Builder) -> u32
```

text_open starts a run of pushed text: the offset text_close measures from.

```
core.text_push :: proc(b: ^Builder, s: string)
```

text_push copies s into the text pool; a full pool poisons the builder with Capacity.

```
core.text_push_byte :: proc(b: ^Builder, c: u8)
```

text_push_byte copies one byte into the text pool, as text_push does.

```
core.text_push_rune :: proc(b: ^Builder, r: rune)
```

text_push_rune copies r, UTF-8 encoded, into the text pool, as text_push does.

```
core.text_rewind :: proc(b: ^Builder, start: u32)
```

text_rewind drops text pushed since start; used when a tentative parse is abandoned.

```
core.top :: proc(s: ^Stack($T)) -> ^T
```

top returns a pointer to the top item, valid until the next push or pop, or nil when the stack is empty.

```
core.transform_document :: proc(ws: ^Workspace, doc: Document, passes: []Pass, limits := Limits{}) -> Read_Result
```

transform_document runs passes in order. A rebuilding pass writes the inactive bank, which is validated before it becomes active; on failure the old bank stays valid and is returned as the document. A workspace without a second bank fails before a pass runs, with Capacity and need.region Second_Bank. doc must be a valid document of ws. Zero limits mean default_limits.

```
core.trusted_policy :: proc() -> Policy
```

trusted_policy suits conformance tests and content the caller wrote.

```
core.unwrap_node :: proc(b: ^Builder, id: Node_Id)
```

unwrap_node removes a closed node and keeps its descendants in place, as when an RST section becomes the document title. Links that resolved to the removed node resolve to the node that takes its index. Validation still runs after the reader finishes.


```
core.validate :: proc(snap: ^Snapshot, sources:
[]Source_Entry, scratch: ^Scratch, limits: Limits) ->
(Status, Diagnostic)
```

validate checks every structural invariant of a finished snapshot. It needs a stack of open subtree ends, taken from scratch and released before returning.

```
core.validate_registry :: proc(r: Registry) -> Result
```

validate_registry checks that every adapter has an ID, a version, and a procedure, and that no ID is registered twice. convert checks its registry the same way before using it, so a bad registry fails with Invalid_Input instead of the first match silently winning; a host may call this once where it builds a registry.

```
core.view :: proc(doc: Document) -> (View, bool)
```

view checks doc once (document_valid) and returns its read-only View, which borrows the workspace until doc becomes invalid; false for a stale or foreign handle.

```
core.workspace_size :: proc(input_bytes: int) -> (bytes: int,
ok: bool) #optional_ok
```

workspace_size returns the slab size, in bytes, of slab_plan(input_bytes): one bank, for conversions without rebuilding passes. ok is false when the size cannot be represented. Size a workspace for passes with a plan (see above).

```
core.write_id :: proc(out: ^Buffer, name: string)
```

write_id lowercases letters, keeps letters and digits, and joins every other run of characters with one hyphen. It names documents and chapters, whose names may start with digits; make_id names anchors.

```
core.write_object :: proc(v: View, e: ^Emitter)
```

write_object serializes the document in view v. Use Measure mode first for its size.

Constants

```
core.BUILDER_OPEN :: max(u32) - 1
```

```
core.CORE_CAPACITY
```

```
core.CORE_DEPTH
```

```
core.CORE_NODES
```

```
core.CORE_STACK
```

```
core.CORE_UNHANDLED_KIND
```

```
core.CORE_WORK
```

```
core.DEFAULT_STACK_BYTES :: (512 << 10) * ASAN_FRAME_SCALE
```

DEFAULT_STACK_BYTES is the stack a reader or renderer may use when Limits.stack_bytes is 0. It holds the default max_depth of every recursive path with room to spare, and with STACK_RESERVE it fits a 1 MiB thread, the smallest default stack of the supported platforms¹ threads (Windows; Odin's threads on Unix take RLIMIT_STACK, usually 8 MiB).

```
core.INCLUDE_MISSING
```

```
core.NONE :: max(u32)
```

```
core.SCHEMA_VERSION :: 1
```

```
core.STACK_RESERVE :: (128 << 10) * ASAN_FRAME_SCALE
```

STACK_RESERVE is the stack a conversion needs below its deepest guard.

```
core.VERSION :: "0.2.0"
```

VERSION is the engine's version. It is released with Guidedog and so far follows its version (cli.VERSION), but is kept apart because programs embed the engine without the command. SCHEMA_VERSION counts the incompatible changes to documents saved as objects.

Variables

```
@(rodata) core.ENGLISH_LABELS
```

ENGLISH_LABELS are the labels in English, Sphinx's msgids; renderers fall back to them.

```
@(rodata) core.LIST_STYLE_NAMES
```

defaults

Package defaults assembles the built-in readers and renderers into an explicit, immutable registry. Embedders may build their own registry from any subset.

Procedures

```
defaults.myst_options :: proc(settings: ^commonmark.Settings,
base := gd.Read_Options{}) -> gd.Read_Options
```

myst_options reads with the 「myst」 reader's settings, from commonmark.default_settings or sphinx.markdown_settings. It borrows settings, as rst_options does.

```
defaults.registry :: proc() -> gd.Registry
```

registry returns the built-in registry by value; it borrows static tables, valid for the whole program. Readers: 「rst」, 「sphinx」, 「commonmark」 (pure CommonMark), and 「myst」 (MyST Markdown). Renderers: 「html」, 「typst」, and 「text」.

```
defaults.rst_options :: proc(settings: ^rst.Settings, base :=
gd.Read_Options{}) -> gd.Read_Options
```

rst_options reads with the 「rst」 reader's settings (rst.default_settings() to start). It borrows settings; base, when not zero, supplies the other read options.

```
defaults.sphinx_options :: proc(config: ^sphinx.Config,
base := gd.Read_Options{}) -> gd.Read_Options
```

sphinx_options reads with the 「sphinx」 reader's configuration (sphinx.default_config() to start). It borrows config, as rst_options does.

fetch

Types

```
fetch.Error :: enum u8
```

None

Unsupported

not http, https, or a file; or no libcurl on this platform.

Missing

no such file.

Network

the connection or transfer failed.

Timeout**Status**

the server answered with a status other than 2xx.

Too_Large

`fetch.Options` :: *struct*

Options bound one get. The zero value is the defaults.

timeout_ms: int

the whole transfer; 0: 30 seconds; below 0: none.

max_bytes: int

0: 64 MiB.

user_agent: string

`fetch.Result` :: *struct*

Result is what get read, or why it could not. Its data, url, and message are allocated with the allocator given to get, and the caller owns them: destroy frees them.

data: []u8

url: string

where the data came from, after redirects.

status: int

the HTTP status, or 0 for files.

error: Error

message: string

what went wrong, for a report.

Procedures

```
fetch.destroy :: proc(r: ^Result, allocator :=
context.allocator)
```

destroy frees what get allocated for r, with the same allocator, and clears r.

```
fetch.get :: proc(location: string, options: Options,
allocator := context.allocator) -> Result
```

get reads a URL or path, allocating the result with allocator; the caller owns it and frees it with destroy. It fails with an Error in the result, never a panic: a file that is missing, a transfer that fails, times out, answers with a status other than 2xx, or grows beyond max_bytes. It blocks until the transfer ends; threads may call it at once.

Foreign imports

```
foreign import fetch.curl "system:curl"
```

対応環境: Darwin, Linux, FreeBSD, OpenBSD, NetBSD, WASI, JS, Orca, Freestanding.

gds

Package gds is the GDS process of docs/gds (GDS 0001) as a library: it loads the discussion records of a directory into a Catalog, plans changes to them (a new draft, a promotion, a state or label change, the regenerated index) as a Plan, applies a plan as one recoverable transaction under a lock. RST publication uses the project engine; Typst typesets PDF. Failure is a Problem that carries the exit status, never a panic. It allocates with context.allocator and frees nothing, so a caller runs it in an arena, as `guidedog gds` does, and frees the arena when done.

Types

```
gds.Catalog :: struct
```

```
records: [dynamic]Record
```

```
reserved: int
```

```
gds.Change :: struct
```

```
path: string
```

```
before: string
```

after: string

existed: bool

present: bool

gds.Field *:: struct*

name: string

literal: string

gds.Options *:: struct*

dir: string

command: string

identifier: string

author: string

slug: string

labels: string

state: string

label: string

to: string

reason: string

evidence: string

discussion: string

record: string

target: string

number: int

dry_run: bool

check_only: bool

historical: bool

render: bool

rollback: bool

gds.Plan :: struct

changes: [dynamic]Change

summary: string

gds.Problem :: struct

code: string

title: string

path: string

message: string

hint: string

exit: int

gds.Record :: struct

number: string

slug: string

title: string**state:** string**kind:** string**created:** string**updated:** string**discussion:** string**authors:** []string**labels:** []string**path:** string**text:** string**historical:** bool**fields:** [dynamic]Field

Procedures

gds.apply :: *proc*(*dir*: string, *plan*: Plan) -> Problem

apply writes a plan's changes as one transaction under *dir*: each file must still have the contents the plan was made from, a journal records the originals first, and every file is replaced atomically, so an interrupted apply can be rolled back by recover. Hold lock while planning and applying.

gds.build :: *proc*(*o*: Options, *c*: Catalog, *engine*: string) -> (string, Problem)

build publishes *o.target* 「pdf」, 「html」, or 「both」 (the default). An RST catalog uses the project engine: HTML is a complete site and *o.record* selects separate PDF books. A legacy Typst catalog compiles records directly into docs/build. It returns the output paths, comma-separated. Typst is needed only for RST PDF or legacy publication; *engine* names its program, unless the compiler is embedded. Publication is staged so a compiler failure cannot replace a previously good output.


```
gds.check_index :: proc(dir: string, c: Catalog) -> Problem
```

check_index is a Problem when the registry or bundle of dir differs from what index_plan would write.

```
gds.check_sources :: proc(c: Catalog) -> Problem
```

check_sources checks that every record other than a historical one has the sections the discussion template requires; the first missing one is a Problem.

```
gds.edit_plan :: proc(o: Options, c: Catalog) -> (plan: Plan, p: Problem)
```

edit_plan plans the change o.command asks of the record o.identifier names: promote (a draft takes the next number), state (a move the process allows, with its reason and evidence), or label; with the index refreshed. It changes nothing; apply writes the plan. A historical record, or a move the process forbids, is a Problem.

```
gds.index_plan :: proc(dir: string, c: Catalog) -> (plan: Plan, p: Problem)
```

index_plan plans the regenerated registry.typ and bundle.typ of dir for the catalog; the plan has no changes when they are current.

```
gds.load :: proc(dir: string) -> (catalog: Catalog, p: Problem)
```

load reads the discussion directory dir (records/*.rst or legacy *.typ), and the numbers it reserves, into a catalog allocated in the context arena. It fails with a Problem, and changes nothing, when an update awaits recovery, a record is unreadable, too large, or malformed, or two records claim one number or slug.

```
gds.lock :: proc(dir: string) -> (^os.File, Problem)
```

lock takes the lock of the discussion directory dir and returns its file, which the caller holds while it plans and applies, and closes to release; another command holding it is a Problem. A kernel lock is released even if the process crashes. The lock file is kept: unlinking it would allow two processes to lock different inodes at the same path.

```
gds.new_plan :: proc(o: Options, c: Catalog) -> (plan: Plan, p: Problem)
```

new_plan plans a new draft record for o.identifier: its file, from the template, and the refreshed index. It changes nothing; apply writes the plan. A bad or taken slug, or no author (–author, or git’s user.name), is a Problem.

```
gds.next_number :: proc(o: Options, c: Catalog) -> (int, Problem)
```

`next_number` is the number a promoted draft takes: `o.number` when it is free, else one past the highest that a record, the registry's reservation, or a `gds/` branch of the repository in `o.dir` holds. A number already reserved, or above 9999, is a Problem.

```
gds.now :: proc() -> datetime.DateTime
```

`now` is the moment a build is stamped with. As in Sphinx and other reproducible builds, `SOURCE_DATE_EPOCH` (seconds since 1970, in UTC) sets it; otherwise it is the local time, in the local time zone, which stays loaded for the life of the process.

```
gds.parse_record :: proc(path, source: string) -> (r: Record, p: Problem)
```

`parse_record` reads RST metadata or legacy `#let gds-...` lines from `source`; the record borrows `path` and `source`. A missing, repeated, unknown, or invalid field, a bad slug, or dates out of order is a Problem.

```
gds.problem :: proc(code, title, path, message, hint: string, exit := 1) -> Problem
```

`problem` makes a Problem: its code (「`gds.slug`」), title, the path it concerns, what went wrong, a hint, and the exit status the command returns for it.

```
gds.recover :: proc(dir: string, dry_run: bool) -> Problem
```

`recover` finishes an interrupted apply: it restores every file the journal names to its original contents and removes the journal; `dry_run` only checks that it can. No journal is no work.

```
gds.replace_field :: proc(source, key, literal: string) -> string
```

`replace_field` returns `source` with the value of the metadata field key replaced by the Typst literal (quoted, as 「\ accepted\」), allocated in the context arena.

```
gds.resolve :: proc(c: Catalog, id: string) -> (Record, Problem)
```

`resolve` finds the one record an identifier names: its number (7 or 「0007」), slug, title, or file name. The record borrows the catalog; no match, or more than one, is a Problem that lists the candidates.

```
gds.today :: proc() -> string
```

today is now's date, as YYYY-MM-DD.

Constants

```
gds.LABELS
```

```
gds.STATES :: []string{"prediscussion", "discussion",
"accepted", "published", "committed", "abandoned"}
```

gettext

Package gettext reads, writes, merges, and looks up GNU gettext message catalogs: PO and POT text files, and compiled MO files. Its lookups follow the C library's gettext, ngettext, and pgettext; state is explicit, a Catalog or a Translator, instead of the C library's process-wide domain table. It imports only Odin's core packages.

Types

```
gettext.Bound_Catalog :: struct
```

Bound_Catalog is a catalog a translator serves for a domain.

```
domain: string
```

```
catalog: Catalog
```

```
gettext.Catalog :: struct
```

Catalog is one message catalog. Everything it holds lives in its own arena, so destroy frees it at once. Lookups use an index built on first use.

```
name: string
```

file name, or the name given to parse_po.

```
header: Message
```

the msgid 「」 entry; its strs[o] holds the header fields.

```
has_header: bool
```

```
messages: [dynamic]Message
```

plural: Plural_Rule

allow_fuzzy: bool

look fuzzy translations up (gettext_allow_fuzzy_translations).

index: Index

arena: ^Catalog_Arena

nil when even it could not be allocated.

allocator: mem.Allocator

the arena's allocator; everything above lives in it.

owner: mem.Allocator

allocated the arena itself.

gettext.**Catalog_Arena** :: struct

Catalog_Arena is what a catalog owns: a growing arena on the allocator make_catalog was given. Once refused memory it refuses everything (refused), so a catalog is never built from a mix of what was and was not allocated; catalog_refused tells.

using arena: runtime.Arena

refused: bool

gettext.**Error** :: struct

Error explains a problem in the style of Elm's compiler: where, what, and what to do. kind == .None means success. Strings are allocated with the allocator the failing procedure was given; destroy_error frees them.

kind: Error_Kind

file: string

line: int

1-based; 0 when the problem has no line.

column: int

1-based byte column of the problem in excerpt; 0 for none.

excerpt: string

the offending line.

message: string

hint: string

gettext.**Error_Kind** :: enum u8

None

Syntax

the PO text breaks the format.

Duplicate

a message is defined twice.

Charset

the catalog is not UTF-8.

Encoding

bytes that are not valid UTF-8.

Plural

a Plural-Forms header that cannot be used.

Mo_Format

a truncated or inconsistent MO file.

Io

the file could not be read or written.

Memory

there was not enough memory to hold the catalog.

```
gettext.Header_Field :: struct
```

Header_Field is one 「Name: value」 line of the header entry.

```
name: string
```

```
value: string
```

```
gettext.Index :: struct
```

Index is a catalog's lookup table, built once from its messages. The translations it serves are copied into one pool, each form followed by 「\n」, so every string a lookup returns is a slice of pool: a reader can parse them as one source (see pool_of).

```
built: bool
```

```
fuzzy: bool
```

built with allow_fuzzy.

```
slots: []i32
```

open addressing over entries; -1 is empty.

```
entries: []Index_Entry
```

```
forms: []string
```

slices of pool.

```
pool: string
```

```
gettext.Index_Entry :: struct
```

```
message: i32
```

in messages, or -1 for the header.

```
first: i32
```

first form in forms.

```
count: i32
```

```
gettext.Merge_Options :: struct
```

fuzzy_matching: bool

look for similar messages (msgmerge without -N).

keep_obsolete: bool

keep translated messages that left the template as 「#~」.

stats: ^Merge_Stats

when set, what the fuzzy search did (fuzzy.odin).

```
gettext.Merge_Stats :: struct
```

Merge_Stats counts what a merge's fuzzy search did; set Merge_Options.stats to read it.

searches: int

msgids searched for (each distinct msgid once).

bounds: int

candidates whose byte counts were compared (similarity_bound).

comparisons: int

candidates compared in full (similarity).

work: int

bytes those comparisons read.

cut_short: int

searches the work bound stopped before they had looked everywhere.

```
gettext.Message :: struct
```

Message is one catalog entry. strs holds the msgstr, or msgstr[o..n) of a plural message; an empty string is an untranslated form.

ctx: string

has_ctx: bool

msgctxt 「」 is a context too, different from none.

id: string

id_plural: string

has_plural: bool

strs: [dynamic]string

comments: [dynamic]string

「#」 translator comments.

notes: [dynamic]string

「#。」 comments for translators, from the extractor.

references: [dynamic]string

「#:」 locations, file:line each.

flags: [dynamic]string

「#,,」 flags other than fuzzy, e.g. 「c-format」 .

fuzzy: bool

obsolete: bool

「#~」 : kept for reuse, never looked up.

previous: Previous

has_previous: bool

line: int

where the entry starts in its file; 0 when built in code.

gettext.**Plural_Node** :: struct

op: Plural_Op

a: i16

b: i16

c: i16

value: u64

gettext. Plural_Op :: <i>enum</i> u8

Number

N

Not

Mul

Div

Mod

Add

Sub

Less

Greater

Less_Equal

Greater_Equal

Equal

Not_Equal

And

Or

Choose

$$a ? b : c$$
gettext.Plural_Problem :: *struct*

Plural_Problem is where and why an expression was refused; column is 1-based.

message: string

static text.

column: int

gettext.Plural_Rule :: *struct*

Plural_Rule is a compiled Plural-Forms header: nplurals and the plural expression.

nplurals: int

nodes: [MAX_PLURAL_NODES]Plural_Node

count: int

root: int

gettext.Previous :: *struct*

Previous holds a fuzzy message's earlier msgctxt, msgid, and msgid_plural (the 「#|」 comments msgmerge writes), so a translator sees what changed.

ctx: string

has_ctx: bool

id: string

id_plural: string

has_plural: bool

```
gettext.Reference_Style :: enum u8
```

Wrapped

「#: a.rst:1 b.rst:2」, wrapped at the width, as msgmerge writes.

One_Per_Line

one 「#:」 line per location, as Sphinx's gettext builder writes.

None

no locations (gettext_location = False).

```
gettext.Stats :: struct
```

translated: int

fuzzy: int

untranslated: int

obsolete: int

```
gettext.Translator :: struct
```

Translator serves several text domains, each from a chain of catalogs: the first bound catalog that translates a message wins, as Sphinx searches locale_dirs in order before its own catalogs. It stands in for the C library's global state (bindtextdomain, textdomain, setlocale).

language: string

domain: string

the current domain (textdomain); 「messages」 by default.

bound: [dynamic]Bound_Catalog

allocator: mem.Allocator

```
gettext.Write_Options :: struct
```

```
width: int
```

lines are wrapped to this many characters; 0 or less: no wrapping.

```
references: Reference_Style
```

```
previous: bool
```

write 「#|」 previous msgids of fuzzy messages.

```
obsolete: bool
```

write 「#~」 obsolete messages.

Procedures

```
gettext.add_catalog :: proc(tr: ^Translator, domain: string,  
cat: Catalog)
```

add_catalog binds a catalog to a domain after those already bound; the translator owns it from now on.

```
gettext.add_message :: proc(cat: ^Catalog, m: Message) ->  
^Message
```

add_message appends a copy of m, cloning its strings into the catalog's arena, and returns the copy, which the catalog owns, or nil when the arena refused the memory. The pointer is valid until the next message is added (the list may then move). The index is rebuilt on the next lookup.

```
gettext.bind_text_domain :: proc(tr: ^Translator, domain,  
dir: string, allow_fuzzy := false) -> (found: bool, err:  
Error)
```

bind_text_domain looks for dir/<language>/LC_MESSAGES/<domain>.po, then .mo, trying the language, then its base language (pt_BR, then pt), and binds the first catalog found, which the translator then owns. found is false, with no error, when there is none; an error is allocated with the translator's allocator (destroy_error).

```
gettext.build_index :: proc(cat: ^Catalog)
```

build_index (re)builds the lookup table, in the catalog's arena. Lookups call it when the catalog changed or allow_fuzzy was toggled; call it yourself to pay the cost up front.

```
gettext.catalog_bytes :: proc(cat: ^Catalog) -> int
```

catalog_bytes is what cat's arena holds, in bytes: what a host charges for keeping it.

```
gettext.catalog_refused :: proc(cat: ^Catalog) -> bool
```

catalog_refused tells whether cat's arena was refused memory, so that what it holds may be incomplete. The readers then fail with an error of kind Memory.

```
gettext.charset :: proc(cat: ^Catalog) -> string
```

charset returns the charset the header's Content-Type names, or 「」 without one.

```
gettext.clone_message :: proc(cat: ^Catalog, m: Message) -> Message
```

clone_message copies m and its strings into the catalog's arena.

```
gettext.compile_plural :: proc(expr: string, nplurals: int) -> (Plural_Rule, Plural_Problem)
```

compile_plural compiles an expression for nplurals forms.

```
gettext.default_merge_options :: proc() -> Merge_Options
```

default_merge_options is msgmerge's default: fuzzy matching on, obsolete messages kept.

```
gettext.default_write_options :: proc() -> Write_Options
```

default_write_options wraps at 76 columns, as sphinx-intl (through Babel) does.

```
gettext.destroy :: proc(cat: ^Catalog)
```

destroy frees a catalog and everything it holds.

```
gettext.destroy_error :: proc(err: Error, allocator := context.allocator)
```

destroy_error frees the strings of an error; an error of kind Memory has static text, which it leaves alone.

```
gettext.destroy_translator :: proc(tr: ^Translator)
```

destroy_translator frees the translator and every catalog bound to it.

```
gettext.domain_plural :: proc(tr: ^Translator, domain, id,
id_plural: string, n: int) -> string
```

domain_plural is dngettext.

```
gettext.domain_plural_with_context :: proc(tr: ^Translator,
domain: string, ctx: Maybe(string), id, id_plural: string, n:
int) -> string
```

domain_plural_with_context is dnpgettext.

```
gettext.domain_text :: proc(tr: ^Translator, domain, id:
string) -> string
```

domain_text is dgettext.

```
gettext.domain_with_context :: proc(tr: ^Translator, domain,
ctx, id: string) -> string
```

domain_with_context is dpgettext.

```
gettext.escape_string :: proc(s: string, allocator :=
context.allocator) -> string
```

escape_string escapes a value for a PO string: backslash, quote, and the control characters C names. The result is allocated with allocator; the caller owns it.

```
gettext.find :: proc(cat: ^Catalog, id: string, ctx:
Maybe(string) = nil) -> ([]string, bool)
```

find returns the translated forms of a message, slices of the index pool. ctx nil means no context. A fuzzy message is found only when allow_fuzzy is set.

```
gettext.find_catalog_file :: proc(dir, language, domain:
string, allocator := context.allocator) -> (string, bool)
```

find_catalog_file returns the path of a domain's catalog for a language, if any, allocated with allocator; the caller owns it.

```
gettext.format_error :: proc(err: Error, allocator :=
context.allocator) -> string
```

format_error renders an error as a friendly report, in a string allocated with allocator that the caller owns (「」 for no error):

```
-- PO SYNTAX ERROR ----- de/index.po:12
This string has no closing quote.
12| msgstr "Hallo
   |      ^
```

Hint: End the string with " on the same line; continue on the next line.

```
gettext.format_stats :: proc(s: Stats, allocator :=
context.allocator) -> string
```

format_stats writes sphinx-intl's summary line, allocated with allocator; the caller owns it.

```
gettext.header_field :: proc(cat: ^Catalog, name: string) ->
(string, bool)
```

header_field returns the value of a header field, compared case-insensitively.

```
gettext.language_candidates :: proc(language: string,
allocator := context.allocator) -> []string
```

language_candidates lists the catalog directory names to try for a language, most specific first: 「pt-BR」 gives pt-BR, pt_BR, and pt; 「sr@latin」 gives it, then sr. The list and each of its strings are allocated with allocator, apart from language and from each other; the caller owns them. A candidate whose copy is refused is left out.

```
gettext.load_mo :: proc(path: string, allocator :=
context.allocator) -> (Catalog, Error)
```

load_mo reads a compiled catalog. Its messages have no comments or references. The catalog owns its arena, made with allocator (destroy); an error is allocated with allocator too (destroy_error).

```
gettext.load_po :: proc(path: string, allow_fuzzy := false,
allocator := context.allocator) -> (Catalog, Error)
```

load_po reads a PO or POT file. Errors are allocated with allocator.

```
gettext.make_catalog :: proc(name := "", allocator :=
context.allocator) -> Catalog
```

make_catalog returns an empty catalog with a germanic plural rule (n != 1). The catalog owns an arena, made with allocator, that holds everything added to it; destroy frees it whole.

```
gettext.make_translator :: proc(language: string,
allocator := context.allocator) -> Translator
```

make_translator returns a translator for language with no catalogs bound, in the 「messages」 domain. It allocates with allocator; destroy_translator frees it and the catalogs bound to it.

```
gettext.merge :: proc(def, ref: ^Catalog, options :=
Merge_Options{fuzzy_matching = true, keep_obsolete = true},
allocator := context.allocator) -> Catalog
```

merge updates a translation (def) to a new template (ref), as msgmerge and sphinx-intl update do. The result has the template's messages in the template's order, with its locations, notes, and flags:

- a message found in def keeps its translation, translator comments, and fuzzy flag;
- a new message similar to an old translated one gets that translation, marked fuzzy, with the old msgid kept as its previous msgid (「#|」);
- old translated messages the template lost become obsolete (「#~」).

The header is def's, with the template's POT-Creation-Date. The result is a new catalog that owns its arena, made with allocator (destroy); def and ref are borrowed.

```
gettext.new_message :: proc(cat: ^Catalog) -> Message
```

new_message returns an empty message whose lists use the catalog's arena.

```
gettext.parse_mo :: proc(data: []u8, name := "<mo>",
allocator := context.allocator) -> (Catalog, Error)
```

parse_mo decodes MO bytes into a catalog that copies what it keeps into its own arena, made with allocator (destroy), so data may be freed afterwards. Every offset is checked, so a damaged file gives an error, allocated with allocator (destroy_error).

```
gettext.parse_plural_forms :: proc(value: string,
allocator := context.allocator) -> (Plural_Rule, Error)
```

parse_plural_forms compiles a Plural-Forms header value such as 「nplurals=2; plural=(n != 1);」. The rule is a value; an error, allocated with allocator (destroy_error), carries the value as its excerpt.

```
gettext.parse_po :: proc(text: string, name := "<string>",
allow_fuzzy := false, allocator := context.allocator) ->
(Catalog, Error)
```

parse_po parses PO or POT text. name labels errors and the catalog. The catalog copies what it keeps, so text may be freed afterwards.

```
gettext.plural_forms_for :: proc(language: string) -> string
```

plural_forms_for returns a Plural-Forms header value for a language, for a new catalog: the rules the GNU gettext manual lists for common languages, and the germanic rule (n != 1) for others. 「pt_BR」 and 「pt-BR」 find

「pt_BR」, then 「pt」. The value is static; the candidates are tried in the temp allocator.

```
gettext.plural_index :: proc(rule: ^Plural_Rule, n: u64) -> int
```

plural_index returns the msgstr index for n; out-of-range results select form 0, as the C library does.

```
gettext.pool_of :: proc(cat: ^Catalog) -> string
```

pool_of returns the text every translation of the catalog is a slice of, so a caller can register it once as a source and give parsed translations exact spans.

```
gettext.save_po :: proc(cat: ^Catalog, path: string,
options := Write_Options{width = 76, previous = true,
obsolete = true}, allocator := context.allocator) -> Error
```

save_po writes a catalog to a file, formatting it in the temp allocator; an error is allocated with allocator (destroy_error).

```
gettext.set_header_field :: proc(cat: ^Catalog, name, value:
string)
```

set_header_field replaces a header field, or appends it, keeping the others in order; the catalog's arena holds the copy.

```
gettext.similarity :: proc(a, b: string) -> f64
```

similarity is $2 \cdot M / (|a| + |b|)$, where M is the length of the longest common subsequence of bytes: 1 for equal strings, 0 for strings with nothing in common. It is the measure msgmerge's fstrcmp approximates. Long strings are compared by their first FUZZY_PREFIX bytes.

```
gettext.sort_by_reference :: proc(cat: ^Catalog)
```

sort_by_reference orders messages by their first location: file name, then line number as a number, then msgid; messages without locations go last. Obsolete messages keep their place after the live ones. Each message's own locations are sorted the same way, as Sphinx sorts them.

```
gettext.stats :: proc(cat: ^Catalog) -> Stats
```

stats counts messages as sphinx-intl stat does; the header does not count.

```
gettext.text_domain :: proc(tr: ^Translator, domain: string)
```

text_domain sets the current domain, as textdomain does; the translator owns its copy.

```
gettext.translated :: proc(m: Message) -> bool
```

translated reports whether every form of *m* has a translation.

```
gettext.translation :: proc(cat: ^Catalog, id: string, ctx:
Maybe(string) = nil) -> (string, bool)
```

translation returns a message's singular translation when there is one.

```
gettext.write_mo :: proc(cat: ^Catalog, include_fuzzy :=
false, allocator := context.allocator) -> []u8
```

write_mo compiles a catalog as msgfmt does: the header and every translated message, sorted by key, with a hash table. Fuzzy messages are left out unless include_fuzzy. The bytes are allocated with allocator; the caller owns them. They are nil when the memory was refused: a compiled catalog is never shorter than its 28-byte header.

```
gettext.write_po :: proc(cat: ^Catalog, options :=
Write_Options{width = 76, previous = true, obsolete = true},
allocator := context.allocator) -> string
```

write_po writes a catalog as PO text: the header, then the messages in order, then the obsolete ones, each entry laid out as msgmerge lays it out. The output depends on nothing but the catalog and the options, so files diff cleanly. The text is allocated with allocator; the caller owns it.

Procedure groups

```
gettext.plural :: proc{catalog_plural, translator_plural}
```

plural is ngettext.

```
gettext.plural_with_context ::
proc{catalog_plural_with_context,
translator_plural_with_context}
```

plural_with_context is npgettext.

```
gettext.text :: proc{catalog_text, translator_text}
```

text is gettext, on a catalog or on a translator's current domain.

```
gettext.with_context :: proc{catalog_with_context,
translator_with_context}
```

with_context is pgettext.

Constants

```
gettext.FUZZY_MIN_WORK :: 1 << 26
```

```
gettext.FUZZY_THRESHOLD :: 0.6
```

FUZZY_THRESHOLD is msgmerge's: a changed message inherits the translation of the most similar old message when their similarity is at least this.

```
gettext.FUZZY_WORK_PER_BYTE :: 256
```

Variables

```
@(rodata) gettext.ERROR_TITLES
```

graphdog

Package graphdog turns Graphviz DOT source into SVG (or PNG, or laid-out DOT) by calling the Graphviz C library in process. It depends only on Odin's core packages and Graphviz.

Example:

```
import "core:fmt"
import "lib/graphdog"
source := "digraph { a -> b }"
ctx := graphdog.init()
defer graphdog.close(&ctx)
svg, err := graphdog.render(&ctx, source, {engine = .Dot}) // svg: the
caller's
if err.kind != .None do fmt.eprintln(graphdog.format_error(err, source,
"g.dot"))
defer delete(svg)
```

Threads. Graphviz keeps process-wide state (the parser, the error handler, plugin tables), so it is not thread-safe even across separate contexts. graphdog therefore holds one process-wide mutex around every call into Graphviz: render from as many threads as you like, but renders run one at a time.

Types

```
graphdog.Attribute :: struct
```

One -G/-N/-E default. Like dot, it overrides the graph's own defaults (node [shape=box]) but not attributes written on a single node or edge.

```
kind: Attribute_Kind
```

```
name: string
```

value: string

graphdog.**Attribute_Kind** :: *enum* u8

What a default attribute applies to: -G, -N and -E on the dot command line.

Graph

Node

Edge

graphdog.**Context** :: *struct*

A Graphviz context: loaded plugins and nothing else. Make one with init and keep it; loading plugins is the expensive part.

gvc: ^GVC

graphdog.**Engine** :: *enum* u8

Layout engines. Each one is a Graphviz layout plugin: .Dot comes from dot_layout, the rest from neato_layout. The zero value is .Dot, like the dot command.

Dot

hierarchical, for directed graphs

Neato

spring model (stress majorization)

Fdp

spring model (force directed)

Sfdp

multiscale force directed, for large graphs

Circo

circular

Twopi

radial

Osage

clustered array packing

Patchwork

squarified treemap

graphdog.**Error** :: *struct*

Error says what went wrong and how to fix it. line and column are 1-based positions in the source (column counts characters), or 0 when Graphviz does not say. message is one sentence, in Graphviz's own words when it gave any; hint is advice for the author. render allocates near, message and hint with its allocator; delete_error frees them.

kind: Error_Kind

line: int

column: int

near: string

the text Graphviz stopped at, for .Syntax and .Html_Label

message: string

hint: string

graphdog.**Error_Kind** :: *enum* u8

None

Syntax

the DOT source does not parse

Html_Label

an HTML-like label (label=<...>) is malformed

Empty_Input

the source holds no graph

Missing_Engine

this Graphviz has no plugin for the layout engine

Missing_Format

this Graphviz has no plugin for the output format

Layout

the layout engine reported an error

Render

the renderer reported an error

Out_Of_Memory**File_Refused**

the caller refused a file the graph names (Options.files)

Not_Initialized

render was called with a context that init did not make

```
graphdog.File_Access :: struct
```

File_Access decides each file name a graph gives through attribute (「src」 for an): it returns the path Graphviz may open, 「」 to leave the attribute out, or ok false to refuse the graph, which render reports as a .File_Refused error.

```
procedure: proc(user: rawptr, attribute, name: string) ->
(path: string, ok: bool)
```

```
user: rawptr
```

```
graphdog.Format :: enum u8
```

Output formats. The zero value is .Svg, which the core plugin always provides.

Svg

standalone SVG 1.1 document, UTF-8

Png

needs the cairo (pango) or gd plugin; the static build has neither

Dot

the input with layout positions attached, like `dot -Tdot`

```
graphdog.Options :: struct
```

```
engine: Engine
```

```
format: Format
```

```
attributes: []Attribute
```

```
warnings: ^[dynamic]string
```

When set, each Graphviz warning (「using box for unknown shape boxx」) is appended, allocated with the allocator given to render.

```
files: File_Access
```

When set, decides every file the graph names before Graphviz may open it (see `files.odin`); otherwise Graphviz opens what the graph names, as dot does.

```
reserve: Reserve
```

When set, counts the output before render allocates it.

```
graphdog.Reserve :: struct
```

Reserve lets the caller count the output before render allocates it: false refuses the bytes, and render reports `.Out_Of_Memory`.

```
procedure: proc(user: rawptr, bytes: int) -> bool
```

```
user: rawptr
```

Procedures

```
graphdog.close :: proc(ctx: ^Context)
```

close frees the context. Output and errors returned by render stay valid.

```
graphdog.delete_error :: proc(err: Error, allocator := context.allocator)
```

delete_error frees the strings of an error that render returned, with the allocator given to render.

```
graphdog.engine_name :: proc(engine: Engine) -> string
```

engine_name is the Graphviz name of an engine (「dot」, 「neato」, ...).

```
graphdog.error_title :: proc(kind: Error_Kind) -> string
```

error_title is a short upper-case title for the kind of error, such as 「DOT SYNTAX ERROR」.

```
graphdog.format_error :: proc(err: Error, source, name: string, allocator := context.allocator) -> string
```

format_error writes err the way Elm writes compiler errors: a title rule naming the file, the source line with a caret under the problem, what went wrong, and a hint. name labels the source (「diagram.dot」, 「<stdin>」); source is the text given to render. The text is allocated with allocator, and the caller owns it.

```
-- DOT SYNTAX ERROR -----
diagram.dot

3 | b -> -> c
   ^ ^
Graphviz stopped at `->`.
Hint: An edge needs a node on each side, like `a -> b`. ...
```

```
graphdog.format_message :: proc(title, name, message, hint: string, allocator := context.allocator) -> string
```

format_message writes an error that has no source to point at, such as a bad command line option, in the same shape as format_error. The text is allocated with allocator, and the caller owns it.

```
graphdog.init :: proc() -> (ctx: Context, ok: bool)
#optional_ok
```

init creates a Graphviz context and loads its plugins. ok is false only when Graphviz cannot allocate one.


```
graphdog.parse_engine :: proc(name: string) -> (Engine, bool)
```

parse_engine maps a Graphviz engine name to an Engine.

```
graphdog.parse_format :: proc(name: string) -> (Format, bool)
```

parse_format maps a -T format name to a Format.

```
graphdog.render :: proc(ctx: ^Context, source: string,
options := Options{}, allocator := context.allocator) ->
(output: []u8, err: Error)
```

render lays out one graph from DOT source and renders it. On success output holds the rendered bytes, allocated with allocator (string(output) for SVG); Graphviz's own buffers are already freed. On failure output is nil and err says what went wrong and how to fix it; its strings are allocated with allocator too (see delete_error). Only the first graph in source is rendered, as with dot without -O.

```
graphdog.version :: proc(ctx: ^Context) -> string
```

version is the linked Graphviz version, such as 「16.1.0」. It is owned by Graphviz and valid until close.

Constants

```
graphdog.FILE_ATTRIBUTES :: [?]string{"image", "shapefile",
"imagepath", "fontpath"}
```

FILE_ATTRIBUTES are the attributes through which Graphviz reads files.

```
graphdog.STATIC :: #config(GRAPHDOG_STATIC, false)
```

The raw Graphviz C API that graphdog uses, and nothing more. Every signature was checked against the headers of Graphviz 16.1 (Homebrew) and 2.42 (Debian); where they differ, the note says how this binding stays correct for both.

Linking. By default graphdog links the shared libraries that a package manager installs:

```
macOS    brew install graphviz    Odin already passes -L/opt/homebrew/
lib and                                     -L/usr/local/lib, and Homebrew's
dylibs carry                                absolute install names, so no rpath
is needed.
Linux    apt install libgraphviz-dev the libraries are on the default
search path.
```

GRAPHDOG_LINK_FLAGS adds linker flags for any other prefix, for example

```
-define:GRAPHDOG_LINK_FLAGS="-L/opt/gv/lib -Wl,-rpath,/opt/gv/lib"
```

-define:GRAPHDOG_STATIC=true links the static libraries that native/graphviz/build.sh builds instead, and preloads the dot, neato and core plugins (see static.odin), so the program needs no Graphviz installation at run time.

Windows has no static build: native/graphviz/windows.sh installs the official Graphviz release for Windows (the same version, checked against its SHA-256) into native/graphviz/out, and graphdog links its import libraries there. The program then needs the DLLs and the plugin list (config8) from native/graphviz/out/bin beside it, which is how the Windows release archive ships them.

Foreign imports

```
@(extra_linker_flags = #config(GRAPHDOG_LINK_FLAGS, ""))
foreign import graphdog.graphviz {"../..../native/graphviz/
out/lib/libgvplugin_dot_layout.a", "../..../native/graphviz/
out/lib/libgvplugin_neato_layout.a", "../..../native/graphviz/
out/lib/libgvplugin_core.a", "../..../native/graphviz/out/lib/
libgvc.a", "../..../native/graphviz/out/lib/libpathplan.a",
"../..../native/graphviz/out/lib/libxdot.a", "../..../native/
graphviz/out/lib/libcgraph.a", "../..../native/graphviz/out/
lib/libcdt.a", "../..../native/graphviz/out/lib/libutil.a",
"../..../native/graphviz/out/lib/libvpasc.a", "system:expat",
"system:z", "system:c++"}
```

対応環境: Darwin.

The attribute takes only literals, so the vendored directory is spelled out.

```
@(extra_linker_flags = #config(GRAPHDOG_LINK_FLAGS, ""))
foreign import graphdog.graphviz {"../..../native/graphviz/
out/lib/libgvplugin_dot_layout.a", "../..../native/graphviz/
out/lib/libgvplugin_neato_layout.a", "../..../native/graphviz/
out/lib/libgvplugin_core.a", "../..../native/graphviz/out/lib/
libgvc.a", "../..../native/graphviz/out/lib/libpathplan.a",
"../..../native/graphviz/out/lib/libxdot.a", "../..../native/
graphviz/out/lib/libcgraph.a", "../..../native/graphviz/out/
lib/libcdt.a", "../..../native/graphviz/out/lib/libutil.a",
"../..../native/graphviz/out/lib/libvpasc.a", "../..../native/
graphviz/out/lib/libexpat.a", "../..../native/graphviz/out/lib/
libz.a", "system:stdc++", "system:m"}

```

対応環境: Windows, Linux, FreeBSD, OpenBSD, NetBSD, WASI, JS, Orca, Freestanding.

```
@(extra_linker_flags = #config(GRAPHDOG_LINK_FLAGS, ""))
foreign import graphdog.graphviz {"../..../native/graphviz/out/
lib/gvc.lib", "../..../native/graphviz/out/lib/cgraph.lib"}
```

対応環境: Windows.

```
@(extra_linker_flags = #config(GRAPHDOG_LINK_FLAGS, ""))
foreign import graphdog.graphviz {"system:gvc",
"system:cgraph"}
```

対応環境: Darwin, Linux, FreeBSD, OpenBSD, NetBSD, WASI, JS, Orca, Freestanding.

highlight

Generated by tools/pygments_styles from pygments-2.19.2's style modules and sphinx-9.1.0's pygments_styles.py; do not edit.

Types

```
highlight.Color :: struct
```

Color is a CSS colour as Pygments normalizes it: six hex digits, case kept.

```
hex: [6]u8
```

```
set: bool
```

```
highlight.Html_Lines :: struct
```

Html_Lines numbers and emphasizes lines as Pygments does for Sphinx's :linenos: and :emphasize-lines: (linenos=「 inline」, hl_lines): each line starts with 7, and an emphasized line, number included, is wrapped in ...\n.

```
first: int
```

the first line's number; 0 leaves lines unnumbered.

```
emphasized: proc(user: rawptr, line: int) -> bool
```

by 1-based line; nil for none.

```
user: rawptr
```

```
highlight.Lexer :: enum u8
```

Lexer names a language. .Default and .Guess are Sphinx's pseudo languages: tokenize resolves them against the code (see resolve).

```
Text
```

```
Python
```

Pycon**Pytb****C****Cpp****Rust****Go****Java****Kotlin****Csharp****Swift****Zig****Odin****Javascript****Typescript****Json****Yaml****Toml****Ini****Xml**

Html**Css****Bash****Console****Powershell****Sql****Diff****Makefile****Dockerfile****Rst****Markdown****Latex****Typst****Dot****Default****Guess**`highlight.Lexer_Info :: struct``name: string`

Pygments』 lexer name, e.g. 「Python」 .

aliases: []string

the names code blocks use; the first is the canonical one.

filenames: []string

glob patterns: 「*.py」, 「Makefile」.

mimetypes: []string

step: Step

spec: ^Code_Spec

the rule tables of lexers built on the code engine.

highlight.**Look** :: struct

Look is a token kind's resolved style.

color: Color

background: Color

border: Color

bold: bool

italic: bool

underline: bool

highlight.**Output** :: struct

Output is caller storage for the writers. They never allocate: bytes that fit go into buf, and written counts every byte the output needs, so a nil buf measures the output and a second call with a buffer of that size writes it. The same input always gives the same bytes.

buf: []u8

written: int

```
highlight.Problem :: struct
```

```
kind: Problem_Kind
```

```
name: string
```

the language or style as written.

```
lexer: Lexer
```

```
suggestion: string
```

a close name, for 「Did you mean」 .

```
line: int
```

Lexing_Error: 1-based line and column (in characters) of

```
column: int
```

the first Error token,

```
excerpt: string
```

the line it is on,

```
near: string
```

and its text.

```
highlight.Problem_Kind :: enum u8
```

```
None
```

```
Unknown_Language
```

Sphinx: 「Pygments lexer name 『x』 is not known」 .

```
Unknown_Style
```

pygments_style names no style.

```
Lexing_Error
```

Sphinx: 「Lexing literal_block ... resulted in an error」 .

```
highlight.Scanner :: struct
```

Scanner tokenizes one input. Create it with `scanner_init` and read tokens with `next_token`, or let `tokenize` and `highlight_html` drive it.

```
code: string
```

```
pos: int
```

```
limit: int
```

steps see `code[:limit]`; an inner lexer sees its region only.

```
state: State
```

```
inner: State
```

a nested lexer: Python after a pycon prompt, CSS in `<style>`.

```
region: int
```

the inner lexer runs while `pos < region`.

```
region_start: int
```

```
nested: bool
```

```
queue: [QUEUE_SIZE]Token
```

```
head: int
```

```
count: int
```

```
highlight.Style :: enum u8
```

Style is a `pygments_style`: every style Pygments ships, and Sphinx's sphinx and none.

```
Abap
```

```
Algol
```

```
Algol_Nu
```


Arduino**Autumn****Borland****Bw****Coffee****Colorful****Default****Dracula****Emacs****Friendly****Friendly_Grayscale****Fruity****Github_Dark****Gruvbox_Dark****Gruvbox_Light****Igor****Inkpot****Lightbulb****Lilypond**

Lovelace**Manni****Material****Monokai****Murphy****Native****None****Nord****Nord_Darker****One_Dark****Paraiso_Dark****Paraiso_Light****Pastie****Perldoc****Rainbow_Dash****Rrt****Sas****Solarized_Dark****Solarized_Light**

Sphinx**Staroffice****Stata_Dark****Stata_Light****Tango****Trac****Vim****Vs****Xcode****Zenburn****highlight.Style_Info** :: *struct***name:** string**dark:** bool

the background is dark.

background: string

the .highlight background.

highlight: string

emphasized lines (.hll).

line_number: string**line_number_background:** string**special_line_number:** string

special_background: string

base: string

the root token's definition, inherited by all.

rules: []Style_Rule

highlight. Style_Rule :: struct
--

Style_Rule is one entry of a style's dict; a style has at most one per kind.

kind: Token_Kind

definition: string

highlight. Token :: struct

Token is a run of the input: code[start:end] has kind. The tokens tokenize returns are contiguous, cover every byte exactly once, and never hold two neighbours of one kind.

start: u32

end: u32

kind: Token_Kind

highlight. Token_Kind :: enum u8

Token_Kind mirrors Pygments' token hierarchy. Each kind has Pygments' dotted name (kind_name), its short CSS class (kind_class) and its parent (kind_parent), so the stylesheets Sphinx themes already ship style Guidedog's output unchanged.

Text

Whitespace

Escape

Error

Other**Keyword****Keyword_Constant****Keyword_Declaration****Keyword_Namespace****Keyword_Pseudo****Keyword_Reserved****Keyword_Type****Name****Name_Attribute****Name_Builtin****Name_Builtin_Pseudo****Name_Class****Name_Constant****Name_Decorator****Name_Entity****Name_Exception****Name_Function****Name_Function_Magic**

Name_Property

Name_Label

Name_Namespace

Name_Other

Name_Tag

Name_Variable

Name_Variable_Class

Name_Variable_Global

Name_Variable_Instance

Name_Variable_Magic

Literal

Literal_Date

String

String_Affix

String_Backtick

String_Char

String_Delimiter

String_Doc

String_Double

String_Escape**String_Heredoc****String_Interpol****String_Other****String_Regex****String_Single****String_Symbol****Number****Number_Bin****Number_Float****Number_Hex****Number_Integer****Number_Integer_Long****Number_Oct****Operator****Operator_Word****Punctuation****Punctuation_Marker****Comment**

Comment_Hashbang

Comment_Multiline

Comment_Preproc

Comment_Preproc_File

Comment_Single

Comment_Special

Generic

Generic_Deleted

Generic_Emph

Generic_Emph_Strong

Generic_Error

Generic_Heading

Generic_Inserted

Generic_Output

Generic_Prompt

Generic_Strong

Generic_Subheading

Generic_Traceback

Punctuation_Indicator

Pygments』 YAML lexer uses two types outside its standard set; HTML gives them the classes of their ancestors too, as Pygments does.

Literal_Scalar_Plain**Generic_Underline**

Types only Pygments』 styles name: github-dark's Generic.Underline, gruvbox's Comment.PreProc (not Comment.Preproc), and the LilyPond lexer's, which lilypond styles. Token.String and Token.Number are not Literal.String and Literal.Number.

Comment_PreProc**Name_Lvalue****Name_Backslash_Reference****Name_Builtin_Articulation****Name_Builtin_Clef****Name_Builtin_Context****Name_Builtin_Context_Property****Name_Builtin_Dynamic****Name_Builtin_Grob****Name_Builtin_Grob_Property****Name_Builtin_Header_Variable****Name_Builtin_Markup_Command****Name_Builtin_Music_Command****Name_Builtin_Music_Function**

Name_Builtin_Paper_Variable**Name_Builtin_Repeat_Type****Name_Builtin_Scale****Name_Builtin_Scheme_Builtin****Name_Builtin_Scheme_Function****Name_Builtin_Translator****Token_String****Token_String_Escape****Token_String_Symbol****Token_Number****Pitch****Chord_Modifier**

Procedures

```
highlight.check_code :: proc(language, code: string) ->
Problem
```

`check_code` finds what Sphinx would warn about when it highlights code as language: an unknown language, or code that does not lex as the language it is marked as. Blocks in the 「default」 language never warn; they fall back to plain text.

```
highlight.check_style :: proc(name: string) -> Problem
```

`check_style` reports a `pygments_style` that names no style.

```
highlight.count_tokens :: proc(lexer: Lexer, code: string) -
> int
```

`count_tokens` is how many tokens tokenize needs for code.

```
highlight.find_lexer :: proc(name: string) -> (Lexer, bool)
```

`find_lexer` finds a lexer by alias (「py」, 「c++」), as Pygments』 `get_lexer_by_name` does, ignoring case; failing that by file name (「setup.py」, 「Makefile」) or MIME type.

```
highlight.find_lexer_for_filename :: proc(path: string) -> (Lexer, bool)
```

`find_lexer_for_filename` matches the base name of a path against the lexers』 patterns.

```
highlight.find_style :: proc(name: string) -> (Style, bool)
```

`find_style` finds a style by its Pygments name, ignoring case.

```
highlight.guess_lexer :: proc(code: string) -> Lexer
```

`guess_lexer` picks a lexer from the code alone, as Sphinx's 「guess」 language does. JSON that parses cleanly is JSON; otherwise the rule table decides.

```
highlight.has_errors :: proc(lexer: Lexer, code: string) -> bool
```

`has_errors` reports whether lexing code produces an Error token, which is when Sphinx warns that a block does not lex as its language.

```
highlight.highlight_html :: proc(lexer: Lexer, code: string, out: ^Output)
```

`highlight_html` lexes code and writes its HTML without a token buffer.

```
highlight.highlight_html_lines :: proc(lexer: Lexer, code: string, lines: Html_Lines, out: ^Output)
```

`highlight_html_lines` lexes code and writes it as `write_html_lines` does.

```
highlight.highlight_typst :: proc(lexer: Lexer, code: string, out: ^Output)
```

`highlight_typst` lexes code and writes its Typst array without a token buffer.

```
highlight.kind_class :: proc(kind: Token_Kind) -> string
```

`kind_class` is the CSS class Pygments』 `HtmlFormatter` writes: 「k」, 「s2」, 「c1」; 「」 for Text, which HTML writes without a span.

```
highlight.kind_is :: proc(kind, ancestor: Token_Kind) -> bool
```

`kind_is` reports whether `kind` is ancestor or one of its descendants, as Pygments』 `tttype` in `Token.String` does.

```
highlight.kind_name :: proc(kind: Token_Kind) -> string
```

`kind_name` is Pygments』 dotted name without the 「Token.」 root: 「Literal.String.Double」.

```
highlight.kind_parent :: proc(kind: Token_Kind) ->
(Token_Kind, bool)
```

`kind_parent` is the kind's parent; false for kinds directly below the root.

```
highlight.lexer_alias :: proc(lexer: Lexer) -> string
```

`lexer_alias` is the lexer's canonical alias: 「python」, 「cpp」.

```
highlight.lexer_info :: proc(lexer: Lexer) -> Lexer_Info
```

`lexer_info` is the lexer's static description: name, aliases, file patterns, and MIME types.

```
highlight.lexer_name :: proc(lexer: Lexer) -> string
```

`lexer_name` is Pygments』 name for the lexer: 「Python」, 「C++」.

```
highlight.next_token :: proc(s: ^Scanner) -> (Token, bool)
```

`next_token` returns the next token; adjacent tokens of one kind come back merged.

```
highlight.output_fits :: proc(out: ^Output) -> bool
```

`output_fits` reports whether everything written so far is in the buffer.

```
highlight.output_string :: proc(out: ^Output) -> string
```

`output_string` is what fits in the buffer; it is the whole output when `output_fits`.

```
highlight.put :: proc(out: ^Output, s: string)
```

`put` writes `s` to `out`: what fits into the buffer, and counts all of it.

```
highlight.put_byte :: proc(out: ^Output, c: u8)
```

`put_byte` writes one byte to `out`, as `put` does.

```
highlight.put_html :: proc(out: ^Output, text: string)
```

put_html writes text escaped as Pygments escapes it: & < > 「 and 〘.

```
highlight.resolve :: proc(lexer: Lexer, code: string) ->
Lexer
```

resolve makes Sphinx's choices for a code block. .Default is Python, or a Python console session when the code starts with 「>>>」, and falls back to .Text when that does not lex cleanly. Python blocks that start with 「>>>」 are console sessions too. .Guess guesses from the code. Every other lexer is itself.

```
highlight.resolve_style :: proc(style: Style) -> (looks:
[Token_Kind]Look, base: Look)
```

resolve_style resolves every kind's look: its parent's, changed by its own rules.

```
highlight.scanner_init :: proc(s: ^Scanner, lexer: Lexer,
code: string)
```

scanner_init prepares s to tokenize code. The pseudo lexers .Default and .Guess are resolved against the code first. Inputs are limited to 4 GiB.

```
highlight.style_name :: proc(style: Style) -> string
```

style_name is the style's pygments_style name, as find_style takes it: 「github-dark」.

```
highlight.suggest_lexer :: proc(name: string) -> string
```

suggest_lexer is the alias closest to a name that find_lexer does not know, for 「Did you mean」 hints; 「」 when nothing is close.

```
highlight.token_text :: proc(t: Token, code: string) ->
string
```

token_text is the input a token covers.

```
highlight.tokenize :: proc(lexer: Lexer, code: string,
buffer: [ ]Token) -> (tokens: [ ]Token, need: int)
```

tokenize fills buffer with the tokens of code and returns them. need is how many the whole input has; when it exceeds len(buffer), tokens holds only the first ones, and a buffer of need tokens holds them all. tokenize never allocates.

```
highlight.write_css :: proc(style: Style, selector: string,
out: ^Output)
```

`write_css` writes the stylesheet Pygments writes for the style with the selector, as Sphinx writes `_static/pygments.css` with `「.highlight」`. It ends without a newline.

```
highlight.write_html :: proc(tokens: []Token, code: string,
out: ^Output)
```

`write_html` writes tokens of code as Pygments' HTML.

```
highlight.write_html_lines :: proc(tokens: []Token, code:
string, lines: Html_Lines, out: ^Output)
```

`write_html_lines` writes tokens of code as Pygments' HTML with numbered or emphasized lines. The whole block is lexed at once, so a string or comment across lines keeps its colour on every line.

```
highlight.write_problem :: proc(p: Problem, location: string,
out: ^Output)
```

`write_problem` writes an Elm-style report; location, such as `「index.rst:12」`, heads it.

```
-- UNKNOWN LANGUAGE ----- index.rst:12

I do not know how to highlight "pyhton", so this block is shown as plain
text.

Hint: Did you mean "python"?
```

```
highlight.write_typst :: proc(tokens: []Token, code: string,
out: ^Output)
```

`write_typst` writes tokens of code as a Typst array of lines. A final newline does not start another line.

```
highlight.write_typst_theme :: proc(style: Style, out:
^Output)
```

`write_typst_theme` writes a style as a Typst dictionary: `「background」` is the block's fill, `「text」` the default text settings, and each class maps to its text settings (fill, weight, style), with `「underline」`, `「background」` and `「border」` where set.

Variables

```
highlight.STYLES
```

highlight/guidedoc

Package `highlight_guidedoc` connects the `highlight` library to Guidedoc's renderers. It is separate so that the `highlight` library itself depends on nothing but Odin's core.

Procedures

```
guidedoc.highlighter :: proc() -> gd.Highlighter
```

`highlighter` is the value for `Render_Options.highlighter`: code blocks become the HTML Pygments writes for Sphinx, with its short CSS classes.

```
guidedoc.typst_highlighter :: proc() -> gd.Highlighter
```

`typst_highlighter` is for a Typst renderer that draws code with Guidedoc's colours rather than Typst's: it writes code as `write_typst`'s array of lines, for the book template's `gd-highlight` function (see the README). It returns `false` for plain text and languages it does not know, so the renderer can keep Typst's own raw block for those.

host

Package `host` composes the allocation-free core with memory, files, and retries. Everything here may allocate; the core's zero-allocation contract ends at `gd` calls.

Ownership, unless a procedure says otherwise: what a procedure returns (a string, a slice, a report, a problem's text) is allocated in `context allocator` and belongs to the caller, which frees it with that allocator's arena rather than piece by piece; its arguments are borrowed for the call only. A failure is a `Problem` (`exit ! = o`) or a false result, never a panic. The surface is listed in `lib/host/README.md`.

Types

```
host.Arena :: struct
```

`Arena` hands out memory from blocks it takes from a backing allocator (a `Meter`, usually) and frees them all at once. It is `core:mem`'s `Dynamic_Arena` made safe for a backing that refuses: a block the backing refuses leaves the arena as it was, and the request fails with `.Out_Of_Memory`, where `Dynamic_Arena` would go on handing out memory at address `o`. The blocks are linked through a header at their start, so the arena allocates nothing besides them; each block is used through bounds-checked slices of itself. It is not safe for use by several threads at once.

```
backing: mem.Allocator
```

```
block: int
```

bytes of a block; `o`: `ARENA_BLOCK_BYTES`.

blocks: ^Arena_Block

the newest first; the first is the one being filled.

last: []byte

the latest allocation from the first block, which can grow.

bytes: int

the bytes of every block held.

host.**Artifact_Request** :: struct

Artifact_Request is one conversion of guidedog convert. Its strings and slices are borrowed for the call.

input: string

path, or '-' for standard input.

from: Format

to: Target

output: string

path, or '-' for standard output.

emit_typst: string

optional path for the intermediate Typst source.

force: bool

policy: gd.Policy

limits: gd.Limits

read: gd.Read_Options

render: gd.Render_Options

typst_program: string

Typst program name or path.

typst_options: `typst.Options`

packages and fonts.

roots: `[]string`

approved resource roots; the input's directory by default.

budget: `int`

allocator: `mem.Allocator`

session regions; unset: `context.allocator`. Must reclaim frees.

host.Artifact_Result *:: struct*

Artifact_Result is what `convert_artifact` did: `exit 0` and the published path (or the bytes for standard output), or `exit != 0` and the problem; reports either way. All of it is allocated in the caller's `context.allocator`.

exit: `int`

reports: `[]Report`

problem: `Problem`

written: `string`

the published path, or `␣` for standard output.

stdout: `[]u8`

bytes to print when output is `␣-␣`.

host.Budget *:: struct*

Budget bounds the memory sessions hold at once: one session's alone, or, shared, the total of a parallel build's threads. A session reserves what it will allocate before allocating it, and releases it when it frees it.

Units of work that run at once (the threads of a parallel read) join the budget while they run. A reservation the budget cannot hold now waits while another joined unit is still running, since that one will release what it holds when it finishes; when every other unit is waiting too, nothing would be released, and it is refused. So a reservation never waits for itself, and waiting units never wait for each other: a build whose units together need more than the budget refuses some of them.

A reservation the budget cannot hold, and no unit will make room for, goes to the budget's policy (`Budget_Policy`): it is refused, unless the policy raises the limit or spills working memory to disk.

limit: int

used: int

active: int

units joined: see `budget_join`.

waiting: int

joined units waiting in reserve.

peak: int

the most used at once.

mutex: `sync.Mutex`

freed: `sync.Cond`

policy: `Budget_Policy`

stopped: bool

the policy chose to stop: later refusals do not ask again.

refused_used: int

What the budget held, and was asked for, when it last refused (`budget_hint`).

refused_need: int

`host.Budget_Policy` :: struct

`Budget_Policy` is what a budget does when it cannot hold a reservation (`Budget`). With no `on_refusal`, and `spill` not set, the reservation is refused: the step that asked fails with `host.budget`. `on_refusal` can raise the limit, or turn on spilling; `spill` set from the start spills without asking. Spilling sends working memory past the limit (`sessions` storage, and what `Meters` allocate) to disk: slower, but bounded by the `Disk`'s own limit and free space. Other reservations, such as a file a build keeps in memory, are still refused.

```
on_refusal: On_Refusal
```

```
user: rawptr
```

```
disk: ^Disk
```

borrowed; the owner destroys it after the build is released.

```
spill: bool
```

```
locate: proc(user: rawptr) -> Report
```

locate says where the work that asked for memory is, for Refusal.place; it is called, with locate_user, from inside the allocator that asked.

```
locate_user: rawptr
```

```
host.Confinement :: enum u8
```

Confinement is where a path lies with respect to the roots a project reads from.

```
Inside
```

confined: below a root once every link is followed.

```
Outside
```

the path as written is below no root.

```
Through_Link
```

written below a root, but a link on the way leads out of the roots.

```
host.Disk :: struct
```

Disk is working memory on disk: memory-mapped temporary files, used when a build's budget is exhausted and the build may go on slowly rather than stop (Budget_Policy). Each file is written out in full before it is mapped, so the disk space is taken before the memory is handed out and writing to it never meets a full disk. On Linux, macOS, and the BSDs a file is removed as soon as it is mapped, so the system reclaims it however the process ends, a crash or Ctrl+C included; on Windows it is created to be deleted when closed.

A Disk hands out memory from its files, as an arena does from its blocks, and counts the allocations each file holds: a file is unmapped when its last allocation is freed, or by disk_destroy, and nothing reads it after. It is safe for

use by several threads at once. Its zero value, with `dir` set, is ready to use; the owner calls `disk_destroy` once nothing allocated from it is used any more.

dir: string

the folder of the temporary files; 「」 for the system's.

limit: int

the most bytes mapped at once; 0: the free space, less `DISK_MARGIN`.

used: int

bytes mapped now.

peak: int

the most bytes mapped at once.

refused: int

bytes a refused request asked for, or 0.

failure: `Disk_Failure`

why the latest request was refused.

chunks: `^Disk_Chunk`

the newest first; the first is the one being filled.

mutex: `sync.Mutex`

`host.Disk_Failure :: enum u8`

`Disk_Failure` is why a `Disk` refused a request.

None

Limit

the disk budget (`Disk.limit`) cannot hold it.

Space

the disk lacks the free space, less `DISK_MARGIN`.

System

the system could not create, write, or map the file.

host.Files :: *struct*

Files provides included resources from approved roots. It refuses Docutils『<standard>』 includes, names that leave the roots, and names that pass through a symbolic link. Links are checked on the path as written, component by component with lstat, before it is resolved; then the resolved path must still lie in a root. The rule is therefore the same on Windows, macOS, and Linux.

roots: []string

max_bytes: int

project: string

When set, a name starting with / is relative to this directory, as Sphinx reads /path in include and literalinclude; otherwise such a name is absolute.

canonical: []string

the roots, resolved; filled by files_provider.

ledger: ^Ledger

When set, each file's size is charged here before it is read (see read_file); a file the budget cannot hold is refused. The caller settles it when the files go.

host.Folder_Entry :: *struct*

Folder_Entry is a name in a folder, what it names (a symbolic link is .Symlink, whatever it leads to), and its inode, as the listing gives it: two names with the same inode on the same file system are the same file. inode is 0 where the system gives none.

name: string

type: os.File_Type

inode: u64

```
host.Format :: enum u8
```

Format is the markup convert_artifact reads.

```
Rst
```

```
Commonmark
```

```
Typst
```

```
host.Guard :: struct
```

Guard is an allocator that watches another for refusals. Guidedog's libraries do not all stop at a failed allocation: they go on with what they have, never touching memory they were refused (tests/allocation_failure_test.odin holds them to that), so their result may be incomplete. A caller that must know its work is whole allocates through a Guard and asks guard_problem when the work ends.

The first refusal trips the guard, and from then on it refuses every allocation itself: memory that ran out once is not relied on again, so the work that follows only shrinks, and nothing is built from a mix of what was and was not allocated. Frees still reach the backing allocator. A guard may be shared by threads.

```
backing: mem.Allocator
```

```
tripped: i32
```

atomic: 1 once an allocation was refused.

```
refused: int
```

atomic: bytes the first refused allocation asked for.

```
host.Identity :: struct
```

Identity is a file's or a folder's device and inode, which a rename keeps. The zero value is none: the path does not exist, or the system gives no inode (Windows may not).

```
device: u64
```

```
inode: u64
```

```
host.Ledger :: struct
```

Ledger counts what one unit of work (reading a document, rendering a page, composing a book) holds of a budget besides its sessions: the files it reads, the graphs it draws. Each charge is made before the memory is allocated, and settle releases them all when the unit frees its memory. A ledger without a budget counts nothing.

```
budget: ^Budget
```

```
held: int
```

```
refused: int
```

bytes a refused charge asked for, or 0.

```
host.Meter :: struct
```

Meter is an allocator that charges a ledger for each allocation before passing it to its backing allocator. It is for work whose memory is not known in advance, such as rendering a page's layout or parsing a saved environment. A charge the budget refuses is refused before anything is allocated: the allocation fails with `.Out_Of_Memory`, the ledger records the refused bytes (`ledger.refused`), and the step that asked stops with `metered_refusal`'s problem. Code behind a meter must therefore handle a failed allocation; see 「Memory」 in `lib/host/README.md` for what the build's steps do.

When the budget spills (`Budget_Policy`), an allocation it cannot hold comes from the policy's Disk instead, and is freed there. Freed memory from the backing allocator are not discharged one by one (an arena's allocator is told no sizes); `settle` returns the step's charges when it frees its memory.

`prepaid` holds bytes charged ahead (`meter_prepay`) that allocations use before charging anew.

```
backing: mem.Allocator
```

```
ledger: ^Ledger
```

```
prepaid: int
```

```
host.On_Refusal :: proc(user: rawptr, r: Refusal) ->
Refusal_Decision
```

`On_Refusal` decides what a budget does with a reservation it cannot hold. It is called at most once at a time, while the budget's other reservations wait; it must not reserve from the budget itself.

```
host.Problem :: struct
```

Problem is a host failure: files, the Typst backend, or command usage.

```
code: string
```

```
title: string
```

```
path: string
```

```
message: string
```

```
hint: string
```

```
exit: int
```

```
host.Refusal :: struct
```

Refusal is what a budget tells its policy when it cannot hold a reservation: the bytes asked for, what the budget holds, and its limit; and where the work that asked is, as the policy's locate says (a report's path, line, and excerpt; zero when unknown), so that a question names the place a report would.

```
need: int
```

```
used: int
```

```
limit: int
```

```
disk: bool
```

the policy has a Disk to go on with.

```
place: Report
```

```
host.Refusal_Choice :: enum u8
```

Refusal_Choice is what a build does when its budget cannot hold a reservation: stop with the budget's problem, raise the budget, or go on with working memory on disk.

```
Stop
```


Raise

to Refusal_Decision.limit.

Disk

host.**Refusal_Decision** :: struct

choice: Refusal_Choice

limit: int

Raise: the new limit in bytes, more than Refusal.limit.

host.**Report** :: struct

Report is a diagnostic with its template filled in and its position resolved.

schema: int

code: string

severity: string

title: string

message: string

hint: string

path: string

line: int

column: int

end_line: int

end_column: int

excerpt: string `json:"-`"

marker: string `json:"- "`

source: string

the complete source line, for JSON consumers; columns count runes.

category: string

The warning's category as 「type.subtype」, such as 「ref.term」, which a host may let users silence (Sphinx's `suppress_warnings`); 「」 for none.

host.Session :: struct

Session owns the storage of one conversion so reports stay valid while it lives. Its storage, output, and source map are reserved from shared when that is set (the threads of a parallel build share one), and otherwise from a budget of its own of limit bytes. allocator supplies its regions (context.allocator when unset); it must outlive the session and reclaim freed regions to bound actual memory across retries. A shared budget's policy may raise its limit; all reservations use that same limit.

allocator: mem.Allocator

ws: gd.Workspace

storage: gd.Storage

sizes: Sizes

output: []u8

result: gd.Result

source_map: []gd.Map_Entry

map_size: int

source map entries; 0 for none.

retries: int

ready: bool

limit: int

o: DEFAULT_BUDGET.

shared: ^Budget

nil: own.

own: Budget

held: int

bytes reserved for the storage, output, and source map.

inputs: ^Kept_File

owned input buffers; freed before releasing kept.

kept: int

bytes reserved for owned input buffers and their records.

refused: int

bytes a refused reservation asked for, or 0; max(int): too many.

denied: bool

the system refused an allocation of refused bytes the budget allowed.

disk: ^Disk

where the regions are when the budget spilled them; nil: memory.

allocated: mem.Allocator

the allocator that owns the current regions.

host.**Sizes** :: struct

Sizes are region capacities in elements (bytes for text and scratch).

nodes: int

attributes: int

lists: int

links: int

cells: int

text: int

scratch: int

sources: int

reports: int

output: int

second_bank: bool

needed only for passes.

host.Target :: *enum* u8

Target is the artifact convert_artifact publishes.

Html

Pdf

Procedures

host.arena_allocator :: *proc*(a: ^Arena) -> mem.Allocator

arena_allocator returns an allocator that allocates from a; it borrows a. Free does nothing; Free_All frees every block.

host.arena_destroy :: *proc*(a: ^Arena)

arena_destroy frees every block of a, which is empty after and may be used again. Nothing allocated from a may be used after it.

host.arena_init :: *proc*(a: ^Arena, backing: mem.Allocator, block := 0)

arena_init starts an empty arena over backing, which must outlive it, taking blocks of block bytes (0: ARENA_BLOCK_BYTES; tests use small ones to reach every allocation).

```
host.available_memory :: proc() -> (bytes: int, ok: bool)
```

available_memory is the memory the system could give the process now without swapping, as it reports it: MemAvailable on Linux, lowered to what the process's cgroup still allows, the available physical memory on Windows, and the free and inactive pages on macOS; or GUIDEDOG_AVAILABLE_MIB when it is set to a number. ok is false when it cannot say. It allocates nothing.

```
host.backend_problem :: proc(program: string) -> Problem
```

backend_problem is the problem of a Typst program that cannot be found; program is borrowed into it.

```
host.budget_advice :: proc(used, need, limit, available: int,
disk: bool, other: string, available_known := false) ->
string
```

budget_advice is the hint budget_hint gives, for a step that asked for need bytes while the budget held used of its limit, on a machine with available bytes free (0 when unknown unless available_known is true). It names the budget to build with (suggest_budget); when that is more than the free memory, it says so, and names keeping working memory on disk first, when disk is there to offer. It is allocated.

```
host.budget_hint :: proc(b: ^Budget, need: int, other:
string) -> string
```

budget_hint is the hint of a budget's problem (budget_advice): the budget the refused step fits in, from what b held and was asked for when it last refused (need, the bytes the step needed, when b refused nothing itself), capped by the memory the machine has free; keeping working memory on disk, when the caller gave b a Disk; and the other way out the caller names. It is allocated.

```
host.budget_join :: proc(b: ^Budget)
```

budget_join counts a unit of work that runs at once with others until budget_leave.

```
host.budget_leave :: proc(b: ^Budget)
```

budget_leave ends a unit budget_join counted, waking reservations that waited for it.

```
host.budget_limit :: proc(b: ^Budget) -> int
```

budget_limit is b's limit now: on_refusal may have raised it since it was set.

```
host.budget_problem :: proc(what, path: string, bytes: int,
b: ^Budget) -> Problem
```

budget_problem explains a charge the budget refused: what needed the bytes, and where. what and path are borrowed into it; the message is allocated.

```
host.budget_report :: proc(s: ^Session) -> (Report, bool)
```

budget_report explains a session that stopped at its budget, or at an allocation the system refused, or returns false. The message is allocated.

```
host.cancel_probe :: proc() -> gd.Cancel_Probe
```

cancel_probe returns a probe with which the core polls the Ctrl+C flag (catch_interrupts) during long conversions; it holds no state of its own.

```
host.canonical_roots :: proc(roots: []string) -> []string
```

canonical_roots resolves roots to absolute paths, as confined compares paths with them; a root that cannot be resolved is kept as written. The slice is allocated.

```
host.catch_interrupts :: proc()
```

catch_interrupts makes Ctrl+C (SIGINT) set a flag instead of ending the process, for the whole process: was_interrupted reads it, and cancel_probe gives it to the core. A program calls it once, at the start.

```
host.charge :: proc(l: ^Ledger, bytes: int) -> bool
```

charge reserves bytes for the unit, waiting as Budget describes, or returns false, reserving nothing and recording the refused bytes (l.refused). A nil ledger, or one without a budget, accepts every charge.

```
host.confined :: proc(roots: []string, path: string) -> bool
```

confined reports whether a file a project reads (a document, an image, a template, a static file) lies in one of the canonical roots once every link on its way is followed. Where the system resolves paths (POSIX) the resolved path decides; on Windows, where it does not, a path through a link is refused.

```
host.confinement :: proc(roots: []string, path: string) -
> Confinement
```

confinement tells whether confined accepts a path, and why not when it does not, so a refusal can say whether the path itself or a link on its way leads outside. A refused path has a link leading out when one of its folders is itself confined: the path is written below a readable folder, and something below that folder leads away.

```
host.containing_root :: proc(roots: []string, path: string) -> (string, bool)
```

containing_root returns the root of roots that path is written below, or false. It compares the paths as written (case-insensitively on Windows) and follows no link; confined is the check that does.

```
host.convert_artifact :: proc(registry: gd.Registry, req: Artifact_Request) -> Artifact_Result
```

convert_artifact is the host route from a file to a published HTML or PDF artifact: read the input, convert it in a session of req.budget bytes, run Typst for PDF, and publish the output atomically (publish), or return it in stdout for 「-」. Guidedoc conversion inside it is allocation-free; reading, Typst, and publishing are not. The result, its reports, and stdout are allocated in context.allocator and owned by the caller; the session is destroyed before it returns. A failure sets exit and problem and leaves any existing output unchanged.

```
host.crash_on_assertion :: proc(prefix, message: string, loc: runtime.Source_Code_Location) -> !
```

crash_on_assertion reports a panic or a failed assertion in Odin code, with its message and where it was raised, and exits with CRASH_EXIT.

```
host.default_stack_budget :: proc(stack: int) -> int
```

default_stack_budget is the budget to use when none is asked for: the default, or less on a thread too small for it.

```
host.describe :: proc(d: gd.Diagnostic, sources: []gd.Source_Entry) -> Report
```

describe fills a diagnostic's templates and resolves its span against sources into a Report: path, line, column, excerpt and marker. It borrows d and sources for the call; the report's strings are allocated and do not refer to the sources afterwards.

```
host.describe_all :: proc(s: ^Session) -> []Report
```

describe_all describes what the session's conversion reported, and the budget it reached if any (budget_report). The reports are allocated and do not borrow the session, so they outlive session_destroy.

```
host.destroy_storage :: proc(st: gd.Storage, allocator := context.allocator)
```

destroy_storage frees every region of storage make_storage allocated with allocator; a region left nil is skipped, so it also frees a partly made storage.

```
host.discharge :: proc(l: ^Ledger, bytes: int)
```

discharge returns bytes the unit has freed before it ends.

```
host.disk_allocator :: proc(d: ^Disk) -> mem.Allocator
```

disk_allocator returns an allocator that allocates from d's files; it borrows d. Free returns an allocation, and unmaps its file once the file holds none; Free_All is not supported (disk_destroy unmaps everything).

```
host.disk_destroy :: proc(d: ^Disk)
```

disk_destroy unmaps every file of d, freed or not; nothing allocated from d may be used after it. d may be used again.

```
host.disk_owns :: proc(d: ^Disk, p: rawptr) -> bool
```

disk_owns reports whether p lies in one of d's files.

```
host.display_path :: proc(name: string) -> string
```

display_path returns name relative to the working directory when that is shorter to read (not above it by three folders or more), and name itself otherwise or for 「」 and 「-」. The result may borrow name; a relative one is allocated.

```
host.dump :: proc(s: ^Session, doc: gd.Document) -> string
```

dump returns the tree outline of a successfully read document, allocated, or 「」 when the document is no longer valid; tests use it.

```
host.each_chunk :: proc(path: string, user: rawptr, each:
proc(user: rawptr, chunk: [u8] -> bool) -> bool
```

each_chunk reads a file through a CHUNK_BYTES buffer, giving each chunk to each until it returns false. It returns whether the whole file was read and accepted.

```
host.excerpt_at :: proc(r: ^Report, text: string, line,
column: int)
```

excerpt_at fills a report's excerpt and marker at a 1-based line and Unicode scalar column, for positions in sources Guidedoc generated, such as a composed book.

```
host.exchange :: proc(a, b: string) -> (swapped: bool, err:
os.Error)
```

対応環境: Windows, Darwin, Linux, FreeBSD, OpenBSD, NetBSD.

exchange: see `switch.odin`. macOS swaps with `renamex_np` and `RENAME_SWAP`. It allocates nothing (`fixed.odin`).

```
host.exit_code :: proc(status: gd.Status) -> int
```

`exit_code` maps a conversion status to the process exit status Guidedog's commands use: 0 ok, 1 invalid input or policy, 3 capacity or limits, 5 internal, 130 cancelled.

```
host.files_provider :: proc(f: ^Files) ->  
gd.Resource_Provider
```

`files_provider` resolves `f.roots` and returns a `gd.Resource_Provider` that serves includes through `f`, which it borrows: `f` and its roots must outlive every conversion the provider is given to. Each file it serves is allocated in the `context allocator` of the conversion's caller and charged to `f.ledger` when that is set.

```
host.fitting_budget :: proc(available: int) -> int
```

`fitting_budget` is a budget that fits in the free memory: nine tenths of it, in steps of 16 MiB; 0 when there is no room for even that minimum. Callers must not offer a zero budget to the CLI, where it means the default budget.

```
host.folders_of :: proc(paths: []string, top: string) ->  
[]string
```

`folders_of` lists, once each and in order, the folders that hold paths and the folders between them and top (top included): the folders whose names a publication adds or changes. A path not below top gives only its own folder.

```
host.grow_session :: proc(s: ^Session, need:  
gd.Capacity_Need) -> bool
```

`grow_session` enlarges the region need names, for the next `prepare_session`; false when the enlarged sizes cannot be represented, which counts as the budget reached.

```
host.guard_allocator :: proc(g: ^Guard) -> mem.Allocator
```

`guard_allocator` returns an allocator that allocates from `g.backing` until it refuses once, and then refuses everything (see `Guard`); it borrows `g`, which must outlive every allocation made through it.

```
host.guard_problem :: proc(g: ^Guard, what: string) ->  
(Problem, bool)
```

`guard_problem` is `host.memory` (exit 3) when an allocation through `g` was refused, with `what`, a static sentence naming the work, as its message; or false. Its text is static, since the memory to write more has run out.

```
host.guard_tripped :: proc(g: ^Guard) -> bool
```

guard_tripped tells whether an allocation through g was refused. A caller may poll it to stop long work early; guard_problem explains it.

```
host.identity :: proc(path: string) -> (id: Identity, err: os.Error)
```

identity is the identity of what is at path; {} with no error when nothing is. It allocates nothing.

```
host.install_crash_reporter :: proc(args: []string, version: string)
```

install_crash_reporter prepares the crash report for the command line args (the program name first) and version, what `guidedog --version` prints (one or more lines), and installs the handlers that print it: for faults and traps: signals on POSIX, an unhandled-exception filter on Windows. It allocates nothing. The caller also sets `context.assertion_failure_proc` to `crash_on_assertion`, so a panic or failed assertion is reported with its message.

```
host.interrupted_problem :: proc() -> Problem
```

interrupted_problem is the problem, with exit status 130, of a command stopped by Ctrl+C.

```
host.io_problem :: proc(path: string, err: os.Error) -> Problem
```

io_problem explains an operating-system error on path; path is borrowed into it.

```
host.is_stage_name :: proc(name: string) -> bool
```

is_stage_name is whether name is one stage_file gives a temporary file: STAGE_PREFIX, STAGE_DIGITS digits, then nothing or one extension. Any other name, however close, is not Guidedog's.

```
host.link_file :: proc(from, to: string, type: os.File_Type) -> (copied: bool, err: os.Error)
```

link_file makes to name the same file as from: a hard link, or, where the file system cannot link it (FAT, some network shares), a copy, which copied reports and the caller must flush. A symbolic link is made again, to the same target. An existing to is an error, never replaced. A regular file is linked or copied without allocating (fixed.odin).

```
host.list_folder :: proc(dir: string, allocator :=
context.allocator) -> (entries: []Folder_Entry, err:
os.Error)
```

対応環境: Windows, Darwin, Linux, FreeBSD, OpenBSD, NetBSD.

list_folder lists a folder's entries, 「.」 and 「..」 left out, in the order the system gives them, with each entry's type from the listing itself (d_type): unlike os.read_all_directory_by_path it opens no entry, which matters for a folder of thousands of files. An entry whose type the file system does not give is read with lstat. The names are allocated in allocator; an allocation that fails is an error.

```
host.main_stack_bytes :: proc() -> int
```

対応環境: Windows, Darwin, Linux, FreeBSD, OpenBSD, NetBSD.

main_stack_bytes is the stack of the main thread.

```
host.make_storage :: proc(s: Sizes, allocator :=
context.allocator) -> (st: gd.Storage, ok: bool) #optional_ok
```

make_storage allocates storage of the sizes with allocator; the caller owns it and frees it with destroy_storage. ok is false, and nothing is kept allocated, when any region cannot be allocated: a region is never left nil. It reserves nothing from a Budget; a Session does that before calling it.

```
host.meter_allocator :: proc(m: ^Meter) -> mem.Allocator
```

meter_allocator returns an allocator that charges m.ledger and allocates from m.backing; it borrows m, which must outlive every allocation made through it.

```
host.meter_prepay :: proc(m: ^Meter, bytes: int) -> bool
```

meter_prepay charges bytes ahead, for work whose memory can be bounded before it starts: the work is refused at once when the budget cannot hold the bound, before any of it is done. Allocations then use the prepaid bytes before charging more; meter_refund returns what they did not use.

```
host.meter_refund :: proc(m: ^Meter)
```

meter_refund returns the prepaid bytes no allocation used to the budget.

```
host.metered_refusal :: proc(l: ^Ledger, what, path: string)
-> (Problem, bool)
```

metered_refusal reports a charge a Meter's ledger could not make: what the step needed, as budget_problem explains it.

```
host.out_of_memory :: proc(what: string) -> Problem
```

out_of_memory is host.memory with what, a static sentence naming the work that could not get memory, as its message; nothing in it is allocated.

```
host.output_text :: proc(s: ^Session) -> string
```

output_text is what the session's last successful run rendered. It borrows the session's output buffer: it is valid until the next run, prepare_session, resize_output, or session_destroy.

```
host.path_type :: proc(path: string) -> (type: os.File_Type,
found: bool, err: os.Error)
```

path_type is what is at path, not following a symbolic link; found is false, with no error, when nothing is. It allocates nothing.

```
host.plan_session :: proc(s: ^Session, input_bytes: int,
second_bank: bool) -> bool
```

plan_session sizes the session for an input of input_bytes; false when the sizes cannot be represented, which counts as the budget reached.

```
host.prepare_session :: proc(s: ^Session) -> bool
```

prepare_session allocates storage of the session's current sizes, releasing any older storage first. It reserves the bytes from the session's budget before allocating, and returns false, allocating nothing, when the budget cannot hold them. run and read call it; callers that load objects call it directly.

```
host.publish :: proc(path: string, data: []u8, force: bool) -
> Problem
```

publish writes bytes through a sibling temporary file, flushes it, and renames it over the destination, making the folders on its way: path holds its old bytes or the new ones, never a part. An existing destination is refused (host.output.exists) unless force is set; any failure leaves the destination unchanged and returns its Problem.

```
host.read :: proc(s: ^Session, reader: gd.Reader, request:
gd.Request, budget := DEFAULT_BUDGET) -> gd.Read_Result
```

read parses without rendering, then runs request.passes (a translation, for example) over the document; storage grows and the whole read retries when a region fills. The document and reports borrow the session, as run's do, and the source text.

```
host.read_file :: proc(path: string, ledger: ^Ledger,
limit := MAX_SOURCE_BYTES, allocator := context.allocator) ->
([]u8, Problem)
```

read_file reads a regular file of at most limit bytes whole, charging its size to ledger before allocating; the caller discharges the bytes when it frees them. A file that grows while it is read is refused. If it shrinks, its original allocation stays charged until freed: fewer bytes read do not mean fewer bytes held.

```
host.read_kept :: proc(s: ^Session, path: string, limit: int)
-> ([]u8, Problem)
```

read_kept reads a file whose bytes the caller keeps while the session's document refers to them. The bytes belong to the caller, allocated in context.allocator; the caller frees them no later than session_destroy, which returns their budget charge. Use read_owned when the session should own and free the bytes itself.

```
host.read_owned :: proc(s: ^Session, path: string, limit:
int) -> ([]u8, Problem)
```

read_owned reads a file the session's document refers to while it lives, such as its cached object. The session owns the returned bytes until session_destroy, which frees them before releasing their charge. Its allocator supplies both the bytes and their ownership record; each allocation is charged before it is made.

```
host.read_source :: proc(path: string, limit :=
MAX_SOURCE_BYTES) -> (text, canonical: string, p: Problem)
```

read_source reads a named input, or standard input for 「-」, within limit bytes. It returns the text and the absolute path used as the source's canonical name.

```
host.remove_path :: proc(path: string) -> os.Error
```

対応環境: Windows, Darwin, Linux, FreeBSD, OpenBSD, NetBSD.

remove_path removes a file, a link, or an empty folder; see fixed.odin.

```
host.remove_stages :: proc(dir: string)
```

remove_stages removes, in dir and every folder below it, the temporary files that a process stopped (killed, or the machine lost power) left before renaming or removing them: files, never folders or links, whose names is_stage_name accepts. Every other name is left as it is. A folder it cannot list, or has no memory to name, is left as it is: a stage file is clutter that nothing reads. The caller makes sure no stage file below dir is still in use.

```
host.rename_path :: proc(from, to: string) -> os.Error
```

対応環境: Windows, Darwin, Linux, FreeBSD, OpenBSD, NetBSD.
 rename_path renames from to to, replacing a file at to; see fixed.odin.

```
host.render_json :: proc(r: Report) -> string
```

render_json is a report as one JSON object (schema DIAGNOSTICS_SCHEMA), without its terminal excerpt and marker; 「{}」 when it cannot be encoded. The text is allocated.

```
host.render_problem_json :: proc(p: Problem) -> string
```

render_problem_json is a host problem as a JSON report of severity 「error」.

```
host.render_problem_text :: proc(p: Problem, color: bool) -> string
```

render_problem_text lays out a host problem as render_text lays out an error report.

```
host.render_text :: proc(report: Report, consequence: string, color: bool) -> string
```

render_text lays out a report the Elm way: what happened, where, what it means for the result, and what to do next. Color supplements words; it never replaces them.

```
host.resize_output :: proc(s: ^Session, bytes: int) -> bool
```

resize_output replaces the session's output buffer with one of bytes, reserving the whole new buffer while the old one still lives. It returns false, keeping the old buffer, when the budget cannot hold the new one.

```
host.resize_source_map :: proc(s: ^Session, entries: int) -> bool
```

resize_source_map gives the session a source map of entries, reserved as resize_output reserves its buffer.

```
host.run :: proc(s: ^Session, registry: gd.Registry, request: gd.Request, budget := DEFAULT_BUDGET, map_source := false) -> gd.Result
```

run converts with storage sized for the input, growing the exhausted region and retrying until it fits or the budget (bytes; see Session) would be exceeded, which ends it with status Limit and a report that says so (budget_report). With map_source set, the session also collects a source map from the renderer. The session owns everything the result refers to: its reports and output_text are

valid until the next run or read on `s`, or `session_destroy`. request is borrowed for the call, and its source text for as long as the reports are read.

```
host.run_workers :: proc(count: int, data: ^$T, work:
proc(data: ^T))
```

`run_workers` runs `work(data)` on `count` threads at once and returns when every one has ended; work shares data between them, and takes its share of the work itself (an index taken atomically, for one). A thread that cannot start is left out; when none can, work runs once on the calling thread, so the work is always done. It allocates the threads with `context allocator`, which the workers may share (`read_pending` gives them the build's, through a mutex this procedure does not take): so every thread is made before any starts, and freed only once every one has ended, and nothing is allocated while a worker runs. A refusal starts fewer.

```
host.same_contents :: proc(path: string, data: []u8) -> bool
```

`same_contents` reports whether the file at `path` holds exactly data, reading it in chunks and stopping at the first difference.

```
host.session_destroy :: proc(s: ^Session)
```

`session_destroy` frees the session's storage, output, source map, and kept inputs, and releases what it reserved and kept from its budget. Reports, `output_text`, and documents the session returned are invalid after it. The session may be run again.

```
host.settle :: proc(l: ^Ledger)
```

`settle` releases everything `l` holds of its budget; the unit calls it when it frees the memory it charged. A nil ledger or one without a budget is a no-op.

```
host.sizes_for :: proc(input_bytes: int) -> (s: Sizes, ok:
bool) #optional_ok
```

`sizes_for` estimates capacities from the input length. The estimate need not be tight: a capacity result names the region, and the session grows it and retries. `ok` is false when a capacity cannot be represented, which a session reports as its budget reached.

```
host.spill_disk :: proc(b: ^Budget) -> ^Disk
```

`spill_disk` is the Disk that working memory `b` cannot hold goes to, or nil when `b` does not spill.

```
host.stack_budget_limit :: proc(stack: int) -> int
```

`stack_budget_limit` is the largest stack budget a thread with stack bytes can hold: all but `STACK_HEADROOM`, and never more than half the stack. An 8 MiB thread (the main thread's and `core:thread`'s usual stack on Linux and macOS) holds 6 MiB; a 1 MiB one (Windows) holds `core.DEFAULT_STACK_BYTES`, which `stack.odin` sizes to fit it.

```
host.suggest_budget :: proc(used, need, limit, available:
int, available_known := false) -> (budget: int, fits: bool)
```

`suggest_budget` is the budget, in bytes, to suggest for a step that asked for need bytes while the budget held used of its limit, so that one change is enough: what is held, and room for the request `STEP_COPIES` times over (a step that builds a value of that size also holds its output and the copy it writes), or twice the old limit, whichever is larger, in steps of 64 MiB. When the machine has less memory free (available; 0 when unknown), the budget is capped to what fits there (`fitting_budget`), or, when what is needed does not fit, is what is needed, in steps of 16 MiB: then `fits` is false. `available_known` distinguishes measured zero from the default, unknown zero.

```
host.sync_all :: proc(paths: []string) -> Problem
```

`sync_all` flushes files and folders to storage, `SYNC_WORKERS` at a time, and returns the first that failed. A folder is flushed so the names in it are durable. Windows keeps its folders' names durable itself and flushes no folder, so they are skipped there.

```
host.sync_each :: proc(paths: []string) -> Problem
```

`sync_each` flushes each path in turn on the calling thread, as `sync_all` does with threads, and returns the first that failed; it allocates nothing.

```
host.thread_stack_bytes :: proc() -> int
```

対応環境: Windows, Darwin, Linux, FreeBSD, OpenBSD, NetBSD.
`thread_stack_bytes` is the stack of a thread `core:thread` starts.

```
host.was_interrupted :: proc() -> bool
```

`was_interrupted` reports whether Ctrl+C was pressed since `catch_interrupts`.

```
host.write_unsynced :: proc(path: string, data: []u8) ->
Problem
```

`write_unsynced` writes bytes as `publish` does, through a sibling temporary file renamed over `path`, so `path` holds either its old bytes or the new ones, but does not flush them: a caller that needs them durable syncs them later (`sync_all`).

Constants

```
host.ARENA_BLOCK_BYTES :: 64 << 10
```

ARENA_BLOCK_BYTES is the size of the blocks an Arena takes from its backing allocator; a request larger than a quarter of it gets a block of its own.

```
host.AVAILABLE_OVERRIDE :: "GUIDEDOG_AVAILABLE_MIB"
```

AVAILABLE_OVERRIDE names the environment variable that states the memory available, in MiB, in place of what the system reports: for machines whose report misleads, and for tests that need the same advice on every machine.

```
host.CRASH_EXIT :: 70
```

CRASH_EXIT is the exit status of a command that crashed: EX_SOFTWARE, an internal software error, which no other failure uses.

```
host.DEFAULT_BUDGET :: 256 << 20
```

DEFAULT_BUDGET is a session's budget in bytes when none is given (256 MiB).

```
host.DIAGNOSTICS_SCHEMA :: 1
```

DIAGNOSTICS_SCHEMA is the version of the JSON form of Report (render_json), which a consumer checks before reading the fields; it changes only when a field does.

```
host.DISK_CHUNK_BYTES :: 64 << 20
```

DISK_CHUNK_BYTES is the size of the temporary files a Disk maps; a larger request gets a file of its own size.

```
host.DISK_MARGIN :: 256 << 20
```

DISK_MARGIN is the free space a Disk leaves on the disk it maps files from, so that spilling working memory never fills the disk the rest of the system writes to.

```
host.ISSUE_TRACKER :: #config(GUIDEDOG_ISSUE_TRACKER, "")
```

ISSUE_TRACKER is where a crash is to be reported: the address of the issue tracker of whoever distributes the build, set when it is built (-define:GUIDEDOG_ISSUE_TRACKER=https://...). Without it, the report asks the user to send the details to whoever provided their build, and names no address.

```
host.MAX_RETRIES :: 64
```

MAX_RETRIES only guards against a bug: the budget is what bounds a session. Each retry at least doubles the region that filled, so growth reaches the budget in a few dozen steps however the regions fill in turn, as a document made of includes fills them.

```
host.MAX_SOURCE_BYTES :: 32 << 20
```

MAX_SOURCE_BYTES is the default size limit of one input file (32 MiB).

```
host.MEMORY_HINT
```

MEMORY_HINT is what to do when the system refuses memory.

```
host.PATH_BYTES :: 4096
```

PATH_BYTES is the longest path, with its terminator, these operations take: the longest that Linux takes (PATH_MAX), and four times macOS's.

```
host.STAGE_PREFIX :: ".guidedog-stage-"
```

STAGE_PREFIX begins the name of every temporary file stage_file makes. It is followed by the STAGE_DIGITS digits os.create_temp_file draws, then the destination's extension.

Foreign imports

```
foreign import host.kernel32 "system:Kernel32.lib"
```

対応環境: Windows.

```
foreign import host.system "system:System"
```

対応環境: Darwin.

i18n

Package i18n translates Guidedoc documents the way Sphinx does with gettext: it extracts every translatable message of a document into a POT template (Sphinx's gettext builder), and it replaces each message with its translation from a catalog (Sphinx's Locale transform), re-parsing the translation as inline markup of the document's own language. It imports the Guidedoc core and the gettext library only; the reader that parses translations is passed in (gd.Reader.read_inline).

Types

```
i18n.Compact :: struct
```

Compact is gettext_compact: per top-level directory (true, Sphinx’s default), per document (false), or one catalog of the given name.

```
mode: enum u8 {Per_Directory, Per_Document, Single}
```

```
name: string
```

for Single.

```
i18n.Extract_Options :: struct
```

Extract_Options are the gettext builder’s settings for one document.

```
targets: Targets
```

gettext_additional_targets.

```
location: bool
```

gettext_location: 「#: path:line」 for every occurrence.

```
uuid: bool
```

gettext_uuid: a 「#: uuid」 line for every occurrence.

```
path: string
```

how locations name the main source, e.g. 「../..index.rst」.

```
docname: string
```

the document’s name, e.g. 「guide/usage」; seeds the uuids.

```
base: string
```

included files are named relative to this directory, if set.

```
i18n.Image_Exists :: struct
```

Image_Exists reports whether an image path, as a document would write it, names a file. The host resolves it against the document’s directory.

```
procedure: proc(user: rawptr, docname, path: string) -
> bool
```

```
user: rawptr
```

```
i18n.Image_Localization :: struct
```

Image_Localization is figure_language_filename for the translation pass.

```
format: string
```

「」 is DEFAULT_FIGURE_FILENAME.

```
docname: string
```

```
language: string
```

```
exists: Image_Exists
```

```
i18n.Index_Part :: struct
```

Index_Part is one translatable piece of an index entry line, e.g. 「b」 of 「single: a; b」. first and length locate it in the line.

```
text: string
```

```
first: int
```

```
length: int
```

```
i18n.Pot_Header :: struct
```

Pot_Header holds the values of a template's header, as Sphinx fills them.

```
project: string
```

```
version: string
```

```
copyright: string
```

```
last_translator: string
```

gettext_last_translator.

```
language_team: string
```

gettext_language_team.

```
creation_date: string
```

e.g. 「2026-09-29 12:00+0000」 .

```
i18n.Progress :: enum u8
```

Progress is translation_progress_classes: which translatable nodes get the class 「translated」 or 「untranslated」 , so a stylesheet can show what is left to translate.

```
Translated
```

```
Untranslated
```

```
i18n.Progress_Classes :: bit_set[Progress; u8]
```

```
i18n.Target :: enum u8
```

Target names the extra kinds of content gettext_additional_targets makes translatable.

```
Index
```

「index」 : index entries.

```
Literal_Block
```

「literal-block」 : literal and code blocks, parsed literals.

```
Doctest_Block
```

「doctest-block」

```
Raw
```

「raw」

```
Image
```

「image」 : image alt text.

```
i18n.Targets :: bit_set[Target; u8]
```

```
i18n.Template :: struct
```

Template collects the messages of one or more documents into a POT catalog, as Sphinx's gettext builder does: a message found again gains a

location instead of a second entry, and messages keep the order of their first occurrence.

catalog: gettext.Catalog

seen: map[string]int

msgid -> message index.

uuids: map[string]int

uuids per message, for their positions.

il8n.Translation :: struct

Translation configures the translation pass, Sphinx's Locale transform: every translatable node whose message the catalog translates gets the translation. Inline content is re-parsed with `read_inline`, in the markup the document was written in (read carries the reader's settings, e.g. a `sphinx.Config`), and keeps the original's references; titles keep their anchors. The pass allocates nothing: it takes scratch memory, and registers the catalog's translation pool as one source, so nodes parsed from a translation have exact spans and diagnostics quote the translation.

catalog: ^gettext.Catalog

read_inline: gd.Read_Inline_Proc

read: gd.Read_Options

policy: gd.Policy

limits: gd.Limits

zero: `gd.default_limits()`.

targets: Targets

`gettext_additional_targets`.

source_name: string

names the translations in diagnostics, e.g. 「`de/index.po`」.

progress: Progress_Classes

`translation_progress_classes`.

images: Image_Localization

figure_language_filename; applies without a catalog.

translated: int

Counts of the last run.
messages replaced.

kept: int

translations refused (bad markup or references); warned.

i18n.UI_Problem :: *struct*

UI_Problem is a project catalog that could not be read.

path: string

error: gettext.Error

i18n.Unit :: *enum* u8

Unit says how a node carries a message.

None

Inline

inline content: the source text with its markup; translations re-parse.

Literal

the node's text, verbatim: code, doctest, raw, and math blocks.

Parsed

a parsed literal: inline content, only with the literal-block target.

Alt

an image's Alt attribute.

Graphviz

diagram alt text and, when requested, its literal source.

Index

each part of each index entry.

Caption

a toctree's caption attribute.

Entry

a toctree entry's explicit title.

Meta

the values of a meta directive.

Contents

a contents directive's title.

Procedures

```
i18n.add :: proc(t: ^Template, msgid, location, uuid: string)
```

add adds one occurrence of a message: its location (「path:line」, or 「」) and uuid. The template copies what it keeps, with its own allocator, so the arguments are borrowed.

```
i18n.destroy_template :: proc(t: ^Template)
```

destroy_template frees a template and everything added to it, and clears it.

```
i18n.domain_for :: proc(docname: string, compact: Compact) -> string
```

domain_for returns the catalog (text domain) a document's messages belong to, as Sphinx's docname_to_domain does: 「guide/usage」 is 「guide」 when compact, else 「guide/usage」; 「index」 is 「index」 either way.

```
i18n.embedded_catalog :: proc(language: string) -> (string, bool)
```

embedded_catalog returns the built-in catalog text for a language, trying it, then its base language (pt_BR, then pt). The text is static, borrowed from the binary; the candidates are made in the temp allocator.


```
i18n.extract :: proc(t: ^Template, v: gd.View, options:
Extract_Options) -> int
```

`extract` adds every translatable message of a document to the template and returns how many occurrences it found. The template copies the messages with its allocator; the view is borrowed, and scratch strings go to the temp allocator.

```
i18n.figure_filename :: proc(format, filename, docname,
language: string, allocator := context.allocator) -> (string,
bool, string)
```

`figure_filename` is `write_figure_filename` into a new string allocated with allocator, which the caller owns; on failure the third result explains the format's problem, in the temp allocator.

```
i18n.figure_for_language :: proc(format, filename, docname,
language: string, exists: Image_Exists, allocator :=
context.allocator) -> string
```

`figure_for_language` returns the localized image when it exists, else the image as written, as Sphinx's `search_image_for_language` does; with no exists.procedure, no localized image exists. Either way the path is a new string allocated with allocator, which the caller owns.

```
i18n.fill_labels :: proc(tr: ^gettext.Translator, labels:
^gd.Labels)
```

`fill_labels` translates every renderer label into labels, for `Render_Options.labels`.

```
i18n.format_date :: proc(format: string, moment:
datetime.DateTime, language: string, allocator :=
context.allocator) -> string
```

`format_date` formats a moment with a strftime format, as Sphinx's `format_date` does, with the month and day names of language. The text is allocated with allocator, and the caller owns it.

```
i18n.index_parts :: proc(line: string, parts: []Index_Part) -
> int
```

`index_parts` splits an index entry line into its translatable pieces, as Sphinx's `split_index_msg` does: the entry type and 「!」 main marker are kept out of them.

```
i18n.make_template :: proc(name := "", allocator :=
context.allocator) -> Template
```

`make_template` starts an empty POT catalog named `name`. It and everything added to it are allocated with `allocator`; the caller frees them with `destroy_template`.

```
i18n.parse_target :: proc(name: string) -> (Target, bool)
```

`parse_target` returns the target a `gettext_additional_targets` name stands for.

```
i18n.set_header :: proc(t: ^Template, h: Pot_Header)
```

`set_header` gives the template the header Sphinx's `gettext` builder writes, copied with the template's allocator.

```
i18n.translation_pass :: proc(t: ^Translation) -> gd.Pass
```

`translation_pass` returns the pass; `t` must outlive the transform. The workspace keeps views of the catalog's pool and of `source_name` (spans, and reports quoting them), so the catalog and `source_name` must outlive every use of the document and its reports.

```
i18n.ui_text :: proc(tr: ^gettext.Translator, msgid: string)
-> string
```

`ui_text` translates one of Guidedog's words.

```
i18n.ui_translator :: proc(language: string, dirs: []string,
allow_fuzzy := false, allocator := context.allocator) ->
(gettext.Translator, []UI_Problem)
```

`ui_translator` makes the translator of Guidedog's own words for a language. `dirs` are the project's locale directories (absolute). A catalog that fails to parse is left out and reported in problems; the others still apply. The translator and its catalogs are allocated with `allocator`, and the caller owns them (`gettext.destroy_translator`); the problems too.

```
i18n.unit_of :: proc(v: gd.View, id: gd.Node_Id, targets:
Targets) -> Unit
```

`unit_of` classifies a node as Sphinx's `is_translatable` does, with the extra targets. The switch is exhaustive, so a new kind does not compile until it says whether it is translated.

```
i18n.write_figure_filename :: proc(out: ^gd.Buffer, format,
filename, docname, language: string) -> (ok: bool, bad_field:
string)
```

`write_figure_filename` fills a `figure_language_filename` format for an image path as written in a document, as Sphinx's `get_image_filename_for_language` does:

- `{root}`: the path without its extension (`「img/logo」` of `「img/logo.png」`);
- `{path}`: its directory with a trailing slash, or `「」` (`「img/」`);
- `{basename}`: the file name without directory or extension (`「logo」`);
- `{ext}`: the extension with its dot (`「.png」`);
- `{docpath}`: the document's directory with a trailing slash, or `「」` ;
- `{language}`: the language.

`「{」` and `「}」` stand for braces. On an unknown field it returns the field's name.

```
i18n.write_message :: proc(v: gd.View, id: gd.Node_Id, out:
^gd.Buffer) -> bool
```

`write_message` writes a node's msgid as Sphinx makes it from the node's source: line breaks and the indentation after them become one space, and the ends are trimmed.

```
i18n.write_template :: proc(t: ^Template, allocator :=
context.allocator) -> string
```

`write_template` writes the template as Sphinx lays out a POT file: every location on its own `「#:」` line, sorted, then the uuids, and strings broken only after newlines. The text is allocated with `allocator`, and the caller owns it.

Constants

```
i18n.DEFAULT_FIGURE_FILENAME :: "{root}.{language}{ext}"
```

`DEFAULT_FIGURE_FILENAME` is Sphinx's default `figure_language_filename`.

```
i18n.UI_DOMAIN :: "guidedog"
```

inventory

Types

```
inventory.Error :: enum u8
```

```
None
```

Unknown_Version

the header names no inventory version Sphinx writes.

Truncated

the header ends early.

Bad_Compression

the zlib data does not inflate.

Out_Of_Memory

the allocator refused memory: the inventory may be incomplete.

```
inventory.Inventory :: struct
```

```
project: string
```

```
version: string
```

```
items: [dynamic]Item
```

```
by_type: map[string]map[string]int
```

by_type[type][name] is the index of the item in items.

```
folded: map[string]map[string]string
```

folded[type][lowercase name] is the first name of that spelling, for the types Sphinx matches ignoring case (labels and terms).

```
inventory.Item :: struct
```

```
project: string
```

the inventory's project and version, for 「(in Python v3.14)」 .

```
version: string
```

```
name: string
```

```
type: string
```

domain:objtype, such as 「py:function」 .

```
priority: int
```

```
uri: string
```

absolute, or relative to the inventory's base.

```
display: string
```

「-」 when it is the name itself.

Procedures

```
inventory.add :: proc(inv: ^Inventory, item: Item) -> bool
```

add appends an item and indexes it; a later item of the same type and name replaces the earlier one in the index, as a later line overwrites in Sphinx's dictionaries. It allocates with context allocator, the arena the inventory lives in, and keeps item's strings, which must live as long. It is false, and the item may be left out, when the allocator refuses memory.

```
inventory.find :: proc(inv: ^Inventory, type, name: string) -> (Item, bool)
```

find returns the item of a type and name, which borrows the inventory.

```
inventory.find_folded :: proc(inv: ^Inventory, type, name: string) -> (Item, bool)
```

find_folded returns the item of a type whose name matches ignoring case: the first such name listed, as Sphinx matches labels and terms. The item borrows the inventory; the lowercase key is made in the temp allocator.

```
inventory.load :: proc(data: []u8, base: string, allocator := context.allocator) -> (inv: Inventory, err: Error)
```

load reads an inventory. base joins relative locations, as Sphinx joins them with the URL or folder the inventory came from; 「」 leaves them as written. It allocates the inventory and its temporary work with allocator, an arena the caller frees; data is only borrowed. A header it does not know, or data that does not inflate, is an Error.

```
inventory.sort_items :: proc(items: []Item)
```

sort_items orders items as Sphinx writes them: by domain, then by the name, display, type, and location of each object.

```
inventory.write :: proc(project, version: string, items:
[]Item, allocator := context.allocator) -> []u8
```

write returns an inventory in version 2 format, lines in the order given, as Sphinx writes them sorted. Locations ending in the name are shortened with \$, and a display equal to the name is written as -. It allocates the bytes, and its temporary work, with allocator, an arena the caller frees.

jinja

Types

```
jinja.Call :: struct
```

Call is what a native procedure receives: the positional and keyword arguments (for a filter or test, args[o] is the value being filtered or tested), the user pointer given at registration, and the receiver of a bound method.

```
args: []Value
```

```
kwargs: []Keyword
```

```
user: rawptr
```

```
self: Value
```

```
name: string
```

```
state: ^State
```

```
jinja.Callable :: struct
```

Callable is a function value: a native procedure (optionally bound to self) or a macro.

```
name: string
```

```
procedure: Native
```

```
user: rawptr
```

```
self: Value
```

```
macro: ^Macro
```

```
jinja.Cycler :: struct
```

Cycler is the object made by the `cycler()` global.

```
items: []Value
```

```
pos: int
```

```
jinja.Dict :: struct
```

Dict is an insertion-ordered dictionary. String keys are indexed; other keys are compared with Python equality (`1 == 1.0 == True`). allocator owns the dict as it owns a List.

```
keys: [dynamic]Value
```

```
values: [dynamic]Value
```

```
index: map[string]int
```

```
kind: Dict_Kind
```

```
name: string
```

the template name of a module

```
allocator: mem.Allocator
```

```
jinja.Dict_Kind :: enum u8
```

```
Dict
```

a plain dict

```
Namespace
```

`namespace()`: the only object whose attributes templates may assign

```
Module
```

the result of `{% import %}`: exported macros and variables

```
jinja.Environment :: struct
```

Environment holds options, filters, tests, globals and the template cache. Its own memory (parsed templates, tables) comes from an arena freed by `destroy_environment`.

```
options: Options
```

```
filters: map[string]Registered
```

```
tests: map[string]Registered
```

```
globals: map[string]Value
```

```
cache: map[string]^Compiled
```

```
arena: ^Arena
```

```
allocator: mem.Allocator
```

```
backing: mem.Allocator
```

```
stack: Stack_Guard
```

started by the outermost call on the environment

```
out_of_memory: bool
```

Set when the environment's memory could not hold its tables: every template it parses then fails with a Memory error.

```
jinja.Error :: struct
```

Error describes a failure. `kind == .None` means success. `excerpt` is the template line the error points at; `hint`, when set, suggests a fix.

```
kind: Error_Kind
```

```
template: string
```

```
line: int
```

```
message: string
```


excerpt: string

hint: string

jinja. Error_Kind :: <i>enum</i> u8
--

None

Syntax

the template does not parse (TemplateSyntaxError)

Undefined

an undefined value was used (UndefinedError)

Type

an operation got the wrong kind of value (TypeError)

Not_Found

a template could not be loaded (TemplateNotFound)

Limit

a recursion, iteration or output limit was reached

Memory

the memory the parse or render may use could not hold it (MemoryError)

Runtime

any other failure while rendering

jinja. Keyword :: <i>struct</i>
--

Keyword is one keyword argument of a call.

name: string

value: Value

jinja.List :: struct

List is a list, or a tuple when tuple is set. fields names the items of a named tuple (the groups made by groupby), which can then also be read as attributes. allocator owns the list: its items grow there, and what a template stores in the list is copied there when the render ends (see Memory in the README).

items: [dynamic]Value

tuple: bool

fields: []string

allocator: mem.Allocator

jinja.Loader :: struct

Loader is a procedure plus its data. destroy, when set, is called by destroy_environment.

procedure: Loader_Proc

user: rawptr

destroy: proc(user: rawptr)

jinja.Loader_Proc :: #type *proc(user: rawptr, name: string, allocator: mem.Allocator) -> (source: string, found: bool)*

Loader_Proc returns the source of a template, allocated with allocator.

jinja.Loop :: struct

Loop is the special loop variable of a for loop.

items: []Value

index0: int

depth0: int

last_seen: Value

for loop.changed()

changed: bool

whether `changed()` was called before

recurse: ^Recursion

set in a recursive loop

jinja.Markup :: *distinct* string

Markup is a string that is already safe HTML: autoescaping leaves it alone.

jinja.Native :: #type *proc*(call: ^Call) -> Value

Native is the one signature of every filter, test, global function and method written in Odin. Report errors with `fail(call, ...)` and return anything.

jinja.None :: struct

None is Jinja's none / None.

jinja.Options :: struct

autoescape: bool

autoescape_extensions: []string

When set, templates whose names end with one of these (「.html」) autoescape and others do not, like `select_autoescape()`; `autoescape` is then ignored.

trim_blocks: bool

lstrip_blocks: bool

keep_trailing_newline: bool

undefined: Undefined_Mode

loader: Loader

max_recursion: int

nested macros, includes and recursive loops; 0 means 200

max_iterations: int

loop iterations per render; 0 means 10,000,000

max_output: int

Bytes per render, and per value a render builds (a repetition, padding, a join), checked before the value is allocated; 0 means unlimited.

max_nesting: int

how deep template syntax nests; 0 means 100

stack_bytes: int

machine stack parsing and rendering may use; 0 means 512 KiB

random_seed: u64

seeds |random and lipsum(); renders are deterministic

jinja.State :: struct

State is one render in progress. Natives reach it through Call.state.

env: ^Environment**out:** ^strings.Builder**err:** Error**failed:** bool**autoescape:** bool**depth:** int**iterations:** int**ctx:** ^Context**current:** ^Compiled**line:** int

rng: u64

guard: ^Guard

the render's allocator; see `memory.odin`.

scratch: ^Arena

the render's own memory, or nil; see `ownership.odin`.

caller: mem.Allocator

context.allocator of the call that renders.

stored: map[rawptr]Value

containers not the render's own it stored values in.

jinja.Template :: struct

Template is a handle to a parsed template.

env: ^Environment

compiled: ^Compiled

name: string

jinja.Undefined :: struct

Undefined is what a missing variable, attribute or item evaluates to. The fields only feed the error message produced when the value is used in a way Jinja forbids.

name: string

the missing variable or attribute

owner: string

「dict object」 when an attribute or item was missing

hint: string

a complete message that replaces the generated one

```
jinja.Undefined_Mode :: enum u8
```

Default

renders as 「」, fails on attribute access, calls and arithmetic

Strict

fails on every use except `is defined` and `|default`

Chainable

like Default, but attributes of undefined are undefined too

Debug

like Default, but renders as `{{ name }}`

```
jinja.Value :: union #no_nil {Undefined, None, bool, i64, f64, string, Markup, ^List, ^Dict, ^Callable, ^Loop, ^Cycler}
```

Value is every value a template can see. The zero value is Undefined.

Procedures

```
jinja.add_filter :: proc(env: ^Environment, name: string,
procedure: Native, user: rawptr = nil)
```

`add_filter` registers a filter: `call.args[0]` is the filtered value. The environment copies name into its arena; user is borrowed until `destroy_environment`.

```
jinja.add_function :: proc(env: ^Environment, name: string,
procedure: Native, user: rawptr = nil)
```

`add_function` registers a native procedure as a global function, allocated in the environment's arena; user is borrowed until `destroy_environment`.

```
jinja.add_global :: proc(env: ^Environment, name: string,
value: Value)
```

`add_global` makes a value visible to every template. The environment copies name into its arena; the value is borrowed and must outlive renders.

```
jinja.add_template :: proc(env: ^Environment, name, source:
string) -> Error
```

`add_template` parses source and caches it under name, so that `extends`, `include` and `import` find it without a loader. An error is allocated with `context allocator` (`destroy_error`).

```
jinja.add_test :: proc(env: ^Environment, name: string,
procedure: Native, user: rawptr = nil)
```

add_test registers a test: call.args[o] is the tested value; return a bool. The environment copies name into its arena; user is borrowed until destroy_environment.

```
jinja.append_value :: proc(l: Value, items: ..Value)
```

append_value adds values to the end of a list value, growing it with the list's own allocator; nothing happens when l is not a list or that allocator refuses.

```
jinja.arg :: proc(c: ^Call, index: int, name: string,
default: Value = Undefined{}) -> Value
```

arg returns positional argument index, else the keyword argument name, else default.

```
jinja.arg_int :: proc(c: ^Call, index: int, name: string,
default: i64) -> (i64, bool)
```

arg_int reads an integer argument (booleans and integral floats accepted).

```
jinja.autoescaping :: proc(c: ^Call) -> bool
```

autoescaping reports whether the call happens while autoescape is on.

```
jinja.boolean :: proc(b: bool) -> Value
```

boolean makes a boolean value.

```
jinja.call_filter :: proc(s: ^State, name: string, args:
[]Value, kwargs: []Keyword) -> Value
```

call_filter runs a registered filter by name (used by map, and by hosts).

```
jinja.call_test :: proc(s: ^State, name: string, args:
[]Value, kwargs: []Keyword) -> bool
```

call_test runs a registered test by name (used by select and reject).

```
jinja.check_args :: proc(c: ^Call, who: string, names:
[]string, required := 0) -> bool
```

check_args enforces a Python signature: at most len(names) positional arguments and only the named keywords, each at most once. who names the callable in errors.

```
jinja.clear_cache :: proc(env: ^Environment)
```

clear_cache forgets parsed templates so the next load reads them again.

```
jinja.create_environment :: proc(options: Options = {}) -  
> Environment
```

create_environment makes an environment with options, zero fields taking their defaults. It keeps its own arena, backed by context allocator, for parsed templates and registrations; destroy_environment frees it. When that memory cannot hold the environment's tables, out_of_memory is set and every parse fails with a Memory error.

```
jinja.destroy_environment :: proc(env: ^Environment)
```

destroy_environment frees the environment's arena, with every template parsed in it, and its loader's state (Loader.destroy).

```
jinja.destroy_error :: proc(err: Error, allocator :=  
context.allocator)
```

destroy_error frees an error returned by this package.

```
jinja.dict :: proc() -> Value
```

dict makes an empty dict, allocated with context.allocator (Undefined when it refuses the memory); fill it with set().

```
jinja.escape :: proc(v: Value, allocator :=  
context.allocator) -> Markup
```

escape converts a value to Markup: Markup stays as it is, everything else is converted to a string and escaped (MarkupSafe's escape()), allocating with allocator when it must; the result may borrow v, so it belongs in an arena, as values do.

```
jinja.escape_string :: proc(s: string, allocator :=  
context.allocator) -> string
```

escape_string replaces the five HTML special characters as MarkupSafe does. When s has none it is returned as it is, borrowed; otherwise the result is allocated with allocator, which the caller owns (or, as usual, an arena holds).

```
jinja.evaluate :: proc(env: ^Environment, expression: string,  
ctx: Value = {}) -> (Value, Error)
```

evaluate evaluates one expression against ctx (Jinja's compile_expression). The value and an error's message are allocated with context.allocator, the caller's; the lists and dicts of the value grow through a small allocator

record kept there too. When that allocator refuses memory the error is a Memory error.

```
jinja.fail :: proc(c: ^Call, format: string, args: ..any) -> Value
```

fail records a runtime error from a native procedure; return its result. The message is formatted with context.allocator, the render's arena.

```
jinja.filesystem_loader :: proc(directories: []string) -> Loader
```

filesystem_loader serves templates from directories, searched in order; the first match wins. Names use 「/」 and may not climb out of a directory with 「..」. The loader copies directories with context.allocator and frees them when its environment is destroyed (Loader.destroy). When that allocator refuses the copy, the loader finds nothing.

```
jinja.float :: proc(f: f64) -> Value
```

float makes a floating-point value.

```
jinja.format_error :: proc(err: Error, allocator := context.allocator) -> string
```

format_error renders an error as a friendly report in the style of Elm's compiler, in a string allocated with allocator that the caller owns (「」 for no error):

```
-- UNDEFINED VALUE ----- page.html:3
'user' is undefined
3| <h1>{{ user.name }}</h1>
Hint: Pass it in the render context, or test it with `is defined`.
```

```
jinja.function :: proc(name: string, procedure: Native, user: rawptr = nil) -> Value
```

function wraps a native procedure as a callable value for globals or context data, allocated with context.allocator (Undefined when it refuses the memory); name and user are borrowed.

```
jinja.get :: proc(d: Value, key: string) -> Value
```

get reads a string key from a dict value; missing keys give Undefined.

```
jinja.has_arg :: proc(c: ^Call, index: int, name: string) -> bool
```

has_arg reports whether the argument was passed at all.

```
jinja.integer :: proc(i: i64) -> Value
```

integer makes an integer value (int would shadow Odin's type).

```
jinja.is_markup :: proc(v: Value) -> bool
```

is_markup reports a safe-HTML string.

```
jinja.is_none :: proc(v: Value) -> bool
```

is_none reports None.

```
jinja.is_undefined :: proc(v: Value) -> bool
```

is_undefined reports an undefined value.

```
jinja.keys :: proc(d: Value, allocator := context.allocator)
-> []string
```

keys returns the string keys of a dict value, in insertion order, in a slice allocated with allocator that the caller owns; the keys borrow the dict.

```
jinja.length :: proc(v: Value) -> int
```

length is the number of items of a list or dict, or of runes of a string.

```
jinja.list :: proc(items: ..Value) -> Value
```

list makes a list holding the given values, allocated with context.allocator; when it refuses the memory, the result is Undefined.

```
jinja.load_template :: proc(env: ^Environment, name: string)
-> (Template, Error)
```

load_template loads a template through the environment's loader, cached in the environment's arena until destroy_environment (or clear_cache). An error is allocated with context.allocator; the caller frees it with destroy_error.

```
jinja.markup :: proc(s: string) -> Value
```

markup makes a string value that is safe HTML, which autoescaping leaves alone; the value borrows s.

```
jinja.memory_loader :: proc(sources: ^map[string]string) -
> Loader
```

memory_loader serves templates from a map the host owns (name -> source).

```
jinja.none :: proc() -> Value
```

none makes Python's None.

```
jinja.parse_template :: proc(env: ^Environment, name, source:
string) -> (Template, Error)
```

parse_template parses source directly (not cached; not visible to includes). The template lives in the environment's arena, which copies source; an error is allocated with context allocator (destroy_error).

```
jinja.render :: proc(t: ^Template, ctx: Value = {}) ->
(string, Error)
```

render renders a template with ctx (a dict value, or Undefined for none). The result and any error message are allocated with context allocator; everything else lives in a scratch arena freed before returning. When that allocator refuses memory the render stops with a Memory error at the line it was rendering.

```
jinja.render_position :: proc() -> (template: string, line:
int, excerpt: string, ok: bool)
```

render_position is where the render running on this thread is: its template's name, the line being rendered, and that line's text; false when none runs. A host asked about a render while it runs, as when the render's allocator asks what to do about memory, names the place with it. The strings borrow the template.

```
jinja.render_source :: proc(env: ^Environment, source:
string, ctx: Value = {}, name := "<template>") -> (string,
Error)
```

render_source parses and renders source in one call (nothing is cached), in a scratch arena it frees before returning. The output, and an error, are allocated with context allocator; the caller owns them.

```
jinja.repr :: proc(v: Value, allocator := context.allocator)
-> string
```

repr converts a value as Python's repr() does, in a string allocated with allocator, which the caller owns.

```
jinja.set :: proc(d: Value, key: string, value: Value)
```

set stores value under the string key in a dict value, growing it with the dict's own allocator; nothing happens when d is not a dict or that allocator refuses.

```
jinja.set_key :: proc(d: Value, key: Value, value: Value)
```

set_key stores value under any key in a dict value.

```
jinja.str :: proc(s: string) -> Value
```

str makes a string value; the value borrows s.

```
jinja.to_string :: proc(v: Value, allocator :=
context.allocator) -> string
```

to_string converts a value as Python's str() does (Undefined gives 「」). A string or Markup value is returned as it is, borrowed; anything else is written into a string allocated with allocator, so the result belongs in an arena, as values do.

```
jinja.truthy :: proc(v: Value) -> bool
```

truthy is Python truthiness (Undefined is false).

```
jinja.tuple :: proc(items: ..Value) -> Value
```

tuple makes a tuple holding the given values, allocated as list allocates.

```
jinja.type_name :: proc(v: Value) -> string
```

type_name is the Python type name used in Jinja's error messages.

```
jinja.undefined :: proc(name := "") -> Value
```

undefined makes an undefined value; name, borrowed, is what error messages call it.

```
jinja.values_equal :: proc(a, b: Value) -> bool
```

values_equal compares as Python's == does: lists and dicts by content, 200 levels deep.

odindoc

Package odindoc reads Odin packages by static analysis, with Odin's own parser (core:odin/parser), and writes the reStructuredText of the Guidedoc `odin` domain that documents them, as sphinx.ext.autodoc writes the Python domain's: packages, procedures, procedure groups, types and their fields or enumerators, constants, variables, and foreign imports, each with its doc comment turned into reST.

It never runs the compiler: types are shown as written. A package's files are read the way Odin builds them for every target at once: file-name suffixes (`_windows`, `_linux_arm64`) and `#+build` tags choose the targets a file builds for, and file-scope `when ODIN_OS == ...` blocks are evaluated when their conditions are that simple. A declaration made for every target the package builds for is documented once; one made for some targets only says which (see `platform.odin`).

It is a host library: it reads files through a `File_Reader` and allocates with the context allocator, freeing nothing itself, so a host runs it in an arena: a `Project`, the packages it loads, and the lines document returns live until the arena is freed.

Types

<code>odindoc.Arch</code> :: <code>runtime.Odin_Arch_Type</code>
--

<code>odindoc.Archs</code> :: <code>runtime.Odin_Arch_Types</code>
--

<code>odindoc.Config</code> :: <i>struct</i>
--

<code>default_options</code> : <code>[]Option</code>
--

<code>member_order</code> : <code>string</code>

「source」 (the default), 「alphabetical」 , 「groupwise」 .

<code>odindoc.Decl</code> :: <i>struct</i>
--

Decl is a package-level declaration. signature is one line, without attributes, as the odin domain's directive takes it (「clone :: proc(s: string) -> string」).

<code>name</code> : <code>string</code>

<code>kind</code> : <code>Kind</code>

<code>signature</code> : <code>string</code>
--

<code>attributes</code> : <code>[]string</code>

as written, in order: 「@(require_results)」 .

<code>docs</code> : <code>string</code>

<code>private</code> : <code>Private</code>

<code>members</code> : <code>[]Member</code>
--

<code>foreign_lib</code> : <code>string</code>
--

the library of a foreign block it is declared in.

<code>targets</code> : <code>Targets</code>

availability: string

its targets for people, when not the package's all.

variant: int

0; 1, 2, ... for later declarations of the name that differ.

file: string

base name.

line: int

order: int

in the package: files by name, then position.

`odindoc.File_Read :: struct`

path: string

text: string

`odindoc.File_Reader :: struct`

`File_Reader` reads the files the analysis needs. `procedure` reads a file; `list` names the entries of a folder (folders end with 「/」). Either may be nil, which uses the file system; a host confines both to its readable roots.

`reserve`, when set, is asked for the memory parsing a file may take (`parse_bound`) before the file is parsed; a file it refuses is not parsed (`Package.too_large`). Odin's parser does not survive a failed allocation, so a host whose allocator can refuse reserves the bound ahead, and its allocator then serves the parse from what was reserved.

procedure: `proc(user: rawptr, path: string) -> (string, bool)`

list: `proc(user: rawptr, dir: string) -> ([string, bool])`

reserve: `proc(user: rawptr, bytes: int) -> bool`

user: rawptr

```
odindoc.Import :: struct
```

Import is a package another package imports: the name it is used by in the source (its alias) and the id of the package it names.

```
alias: string
```

```
id: string
```

```
odindoc.Kind :: enum u8
```

```
Proc
```

```
Proc_Group
```

```
Struct
```

```
Union
```

```
Enum
```

```
Bit_Set
```

```
Bit_Field
```

```
Type
```

```
Const
```

```
Var
```

```
Foreign_Import
```

```
odindoc.Member :: struct
```

```
name: string
```

```
kind: Member_Kind
```

```
signature: string
```

docs: string

the comment above it and the one after it on its line.

`odindoc.Member_Kind :: enum u8`

Field

of a struct: 「[using]name: T[tag]」 .

Enumerator

of an enum: 「Name[= value]」 .

Bit_Field

of a bit_field: 「name: T | bits」 .

`odindoc.Message :: struct`

severity: Severity

code: string

title: string

text: string

hint: string

`odindoc.OS :: runtime.Odin_OS_Type`

`odindoc.Option :: struct`

name: string

value: string

`odindoc.Package :: struct`

Package is an analysed package. doc is its package comment (from doc.odin first, then the first file by name that has one) with comment markers stripped; synopsis is that comment's first sentence.

id: string

name: string

dir: string

doc: string

synopsis: string

decls: []Decl

imports: []Import

targets: Targets

every target some file of the package builds for.

platform: string

targets shown for people; 「」 when it builds for all.

files: []string

the package's files that were read, sorted.

errors: int

syntax errors the parser reported.

too_deep: [dynamic]string

files nested too deeply to parse (nesting.odin).

too_large: [dynamic]string

files whose parse_bound the reader refused.

odindoc.Package_Ref :: struct

id: string

dir: string

```
odindoc.Private :: enum u8
```

```
Public
```

```
Package
```

@(private), or a #+private file.

```
File
```

@(private=「 file」), or a #+private file file.

```
odindoc.Project :: struct
```

```
roots: []Root
```

```
reader: File_Reader
```

```
files: [dynamic]File_Read
```

every file read, for the host's dependencies.

```
packages: map[string]^Package
```

loaded packages by id; nil for one that failed.

```
odindoc.Request :: struct
```

```
directive: string
```

「odin:autopackage」, 「odin:autoproc」, ...

```
argument: string
```

```
options: []Option
```

```
current: string
```

the reader's current odin package, or 「」.

```
odindoc.Result :: struct
```

```
lines: [dynamic]string
```

```
messages: [dynamic]Message
```

```
odindoc.Root :: struct
```

Root is a folder holding packages. Without a collection, a package's id is its path relative to dir (「readers/sphinx」), and dir's own package is named by dir's base name; with one, ids are import paths in that collection (「core:fmt」).

```
dir: string
```

```
collection: string
```

```
odindoc.Severity :: enum u8
```

```
Note
```

```
Warning
```

```
Error
```

```
odindoc.Targets :: [OS]Archs
```

Targets is a set of (os, arch) pairs: per system, its architectures.

Procedures

```
odindoc.document :: proc(p: ^Project, config: ^Config,  
request: Request) -> Result
```

document runs one autodoc directive (request.directive, 「odin:autopackage」 and the others, with its argument and options over config's defaults) and returns the reStructuredText lines it generates, with any messages: an unknown directive, option, package, or name is an Error message and no lines, never a panic. Packages are loaded through p as needed. The result lives in the context arena.

```
odindoc.init :: proc(p: ^Project, roots: []Root, reader :=  
File_Reader{})
```

init starts a project over roots, reading through reader (the file system when it is zero). The project and everything it loads live in the context arena.

```
odindoc.list_packages :: proc(p: ^Project) -> []Package_Ref
```

list_packages lists every folder below the roots that holds a package, by id.

```
odindoc.load :: proc(p: ^Project, id: string) -> (^Package,
bool)
```

load analyses the package with an id, once; later calls return the same package, which the project keeps. It is false when no root holds a package with that id.

```
odindoc.parse_bound :: proc(n: int) -> int
```

parse_bound is the memory parsing a file of n bytes may take.

Constants

```
odindoc.PARSE_BASE_BYTES :: 256 << 10
```

PARSE_BASE_BYTES is what the parser allocates for any file, and an arena block's room.

```
odindoc.PARSE_BYTES_PER_BYTE :: 256
```

PARSE_BYTES_PER_BYTE bounds what Odin's parser allocates, with the build tags read after it, for each byte of a file it parses. Its tree takes 20 to 30 bytes for each byte of ordinary code, and up to about 170 for the densest input measured: long operator chains, runs of semicolons, and one-character list items (parse_bounds_hold). The bound counts every growth of a list as a new allocation, as an arena that cannot grow it in place makes it.

project

Package project builds a documentation project the way Sphinx does: a source directory with conf.toml, builders writing _build/<builder>, and incremental builds. It is host code and may allocate; documents are read and rendered by the Guidedoc engine.

Its surface is the verbs on a Project (project.odin says who owns what they allocate) and the types they take and return; every other file is #+private. lib/project/README.md lists the surface.

Types

```
project.Build_Options :: struct
```

Build_Options are one build's command-line options; build copies them.

```
all: bool
```

-a: write every output, not only outdated ones.

```
fresh: bool
```

-E: ignore the saved environment.

nitpicky: bool`-n`**warnings_as_errors:** bool`-W`**keep_going:** bool`--keep-going`**untrusted:** bool`--untrusted:` see `untrusted_policy`.**jobs:** int`-j:` reading threads; 0 or 1 is one, -1 one per core.**budget:** int`--budget:` bytes; 0 is `BUILD_BUDGET`.**memory:** `host.Budget_Policy`when the budget is reached: `-memory`, or `ask`.**heap:** `mem.Allocator`

Where steps' scratch memory comes from (a document read, a page rendered); zero: the heap. Reading threads use it at once, so it must be thread-safe.

max_depth: int`--max-depth:` how deep content may nest; 0 is the default.**max_nodes:** int`--max-nodes:` nodes per document; 0 is the default.**stack_bytes:** int`--stack-kib:` stack budget; 0 is the default. See `build_limits`.**tags:** `[]string``-t`

output_dir: string

sphinx-build form; otherwise build_dir/<builder>.

arena_block: int

bytes the build's arenas ask for at a time; 0 is 64 KiB.

now: datetime.DateTime

the build's moment; zero is now, in UTC.

year: string

project.**Build_Result** :: struct

Build_Result is what a build did: exit 0, or exit != 0 with a problem; the reports of the whole build either way, and its counters. It owns everything it refers to, in its arena, until release.

exit: int

problem: host.Problem

reports: []host.Report

read: int

documents parsed.

unread: int

documents that could not be read.

reused: int

documents taken from the environment.

written: int

outputs written: published, or only prepared when not published.

unchanged: int

outputs already up to date.

output: string

reached: `Build_Stage`

how far the build got.

published: `bool`

the outputs are in place; otherwise the previous ones stay,

unsettled: `bool`

unless this is set: a publication neither finished nor undone.

peak: `int`

the most memory the budget held at once, in bytes.

budget: `int`

the budget at the end, in bytes: `Budget_Policy` may raise it.

memory: `^Memory`

everything the build allocated; see `release`.

owner: `mem.Allocator`

the allocator memory came from.

`project.Build_Stage :: enum`

`Build_Stage` is how far a build got, so its summary says what it did and no more.

Preparing

settings, the documents to read, the readers: no document read yet.

Reading

reading the documents and resolving their references.

Writing

writing the outputs beside the output folder.

Written

every output written; publishing them is what is left.

```
project.Builder_Kind :: enum u8
```

Builder_Kind is the output a build writes; BUILDER_NAMES are the names the command line gives them, and builder_named reads a name.

Html

Dirhtml

Singlehtml

Pdf

Text

Gettext

Dummy

```
project.Config :: struct
```

Config holds conf.toml. Names follow conf.py so a Sphinx user recognizes them.

source_dir: string

the directory holding conf.toml (absolute).

build_dir: string

_build beside it, or build/ in the separated layout.

project: string

Project.

author: string

Project.

copyright: string

Project.

version: string

Project.

release: string

Project.

today: string

Project.

today_fmt: string

Project.

language: string**root_doc:** string

General.

source_suffix: []string**exclude_patterns:** []string**include_roots:** []string

folders beyond the source folder files may come from.

include_patterns: []string**templates_path:** []string**extensions:** []string**rst_prolog:** string**rst_epilog:** string**default_role:** string**primary_domain:** string

highlight_language: string

pygments_style: string

Pygments style for code; 「」 is the theme's.

pygments_dark_style: string

for dark pages; 「」 is github-dark.

numfig: bool

numfig_format: map[string]string

numfig_secnum_depth: int

math_number_all: bool

math_eqref_format: string

math_numfig: bool

nitpicky: bool

nitpick_ignore: [][]string

nitpick_ignore_regex: [][]string

nitpick_patterns: [][]regex.Regular_Expression

nitpick_ignore_regex, compiled.

show_authors: bool

add_function_parentheses: bool

add_module_names: bool

toc_object_entries: bool

toc_object_entries_show_parents: string

「domain」, 「hide」, or 「all」.

trim_footnote_reference_space: bool

smartquotes: bool

keep_warnings: bool

suppress_warnings: []string

todo_include_todos: bool

autosectionlabel_prefix_document: bool

labels are 「docname:Title」.

autosectionlabel_maxdepth: int

sections this deep or deeper get no label; 0: all.

trim_doctest_flags: bool

doctest flags and <BLANKLINE> hidden in sessions.

manpages_url: string

option_emphasise_placeholders: bool

extlinks: map[string][2]string

extlinks_detect_hardcoded_links: bool

object_types: [dynamic]sphinx.Object_Type

see object_types.odin.

directive_aliases: []sphinx.Directive_Alias

mathjax_path: string

sphinx.ext.mathjax, Sphinx's HTML math; see mathjax.odin.

mathjax_options: map[string]string

mathjax_inline: []string

each an opening and a closing delimiter.

mathjax_display: []string

each an opening and a closing delimiter.

mathjax3_config: string

JSON, as json.dumps writes the table.

mathjax4_config: string

JSON, as json.dumps writes the table.

mathjax_config_path: string

intersphinx_mapping: []Intersphinx_Project

sphinx.ext.intersphinx; see intersphinx.odin.

intersphinx_cache_limit: int

days an inventory is reused; below 0: always.

intersphinx_timeout: int

seconds; 0: no limit, as None in conf.py.

intersphinx_disabled_reftypes: []string

intersphinx_resolve_self: string

autodoc_source_paths: []string

sphinx.ext.autodoc, which reads Python source without running it;
see autodoc.odin.

folders holding the packages, like sys.path.

autoclass_content: string

autodoc_class_signature: string

autodoc_default_options: []ad.Default_Option

autodoc_docstring_signature: bool

autodoc_inherit_docstrings: bool

autodoc_member_order: string

autodoc_mock_imports: []string

accepted: nothing is imported, so nothing is mocked.

autodoc_preserve_defaults: bool

autodoc_typehints: string

autodoc_typehints_description_target: string

autodoc_typehints_format: string

autodoc_type_aliases: map[string]string

autodoc_use_type_comments: bool

autodoc_warningiserror: bool

python_display_short_literal_types: bool

strip_signature_backslash: bool

odin_autoapi_dirs: []string

The odin domain's generating directives and API pages; see `odin.odin`.
folders whose packages get API pages.

odin_collections: map[string]string

collection name to folder (「core」).

odin_autoapi_root: string

the folder of the generated pages: 「api」 .

odin_autoapi_options: []string

members, undoc-members, private-members.

odin_autoapi_member_order: string

source, alphabetical, groupwise.

odin_autoapi_add_tocentry: bool

the API index joins the root's toctree.

odin_autoapi_generate_api_docs: bool

false: only the directives read the dirs.

html_theme: string

HTML.

html_theme_options: map[string]string

html_title: string

html_short_title: string

html_logo: string

html_favicon: string

html_static_path: []string

html_extra_path: []string

html_css_files: []string

html_js_files: []string

html_last_updated_fmt: Maybe(string)

unset: no date; 「」 : the default format.

html_permalink: bool

```
html_permalinks_icon: string
```

```
html_use_index: bool
```

```
html_domain_indices: bool
```

the Python module index, `py-modindex.html`.

```
modindex_common_prefix: []string
```

prefixes the module index leaves out.

```
html_split_index: bool
```

```
html_copy_source: bool
```

```
html_show_sourcelink: bool
```

```
html_sourcelink_suffix: string
```

```
html_show_copyright: bool
```

```
html_show_sphinx: bool
```

```
html_baseurl: string
```

```
html_file_suffix: string
```

```
html_link_suffix: string
```

```
html_secnumber_suffix: string
```

```
html_context: Toml_Value
```

a table, as written: the pages『 variables.

```
html_additional_pages: map[string]string
```

page name: the template it is made from.

```
pdf_documents: []Pdf_Document
```

PDF.

pdf_paper_size: string

pdf_logo: string

pdf_toplevel_sectioning: string

pdf_show_urls: string

pdf_preamble: string

pdf_packages: string

pdf_fonts: string

pdf_font_paths: []string

typst: string

graphviz_output_format: string

Graphviz, linked in through graphdog.
svg; png is read as svg.

graphviz_dot_args: []string

-Gname=value, -Nname=value, -Ename=value.

graphviz_dot: string

accepted from conf.py; Graphviz is built in.

myst_enable_extensions: []string

Markdown, with myst-parser 0.16.1's names; the reader has the support built in.
optional syntax, such as colon_fence.

myst_heading_anchors: int

heading depth that gets slug anchors; 0: none.

myst_dmath_allow_labels: bool

myst_dmath_allow_space: bool

myst_dmath_allow_digits: bool

myst_dmath_double_inline: bool

myst_linkify_fuzzy_links: bool

myst_substitutions: map[string]string

myst_sub_delimiters: []string

two characters, each doubled: {{ name }}.

myst_url_schemes: []string

schemes that make external links; empty: all.

myst_footnote_transition: bool

myst_html_meta: map[string]string

myst_commonmark_only: bool

CommonMark syntax only; headings still make sections.

myst_disable_syntax: []string

markdown-it rule names not parsed.

myst_words_per_minute: int

myst_dmath_enable: bool

deprecated spellings of the extensions.

myst_amsmath_enable: bool

deprecated spellings of the extensions.

locale_dirs: []string

Internationalization, with Sphinx's names (see lib/i18n).
relative to the source directory.

gettext_compact: Flag_Or_Name

true, false, or one catalog's name.

gettext_uuid: bool

gettext_location: bool

gettext_auto_build: bool

compile .po files to .mo beside them.

gettext_additional_targets: []string

gettext_exclude_patterns: []string

document paths excluded from extraction only.

gettext_allow_fuzzy_translations: bool

gettext_last_translator: string

gettext_language_team: string

figure_language_filename: string

translation_progress_classes: Flag_Or_Name

true, false, translated, untranslated.

html_title_derived: bool

html_title was made from the project name.

html_short_title_derived: bool

project.Flag_Or_Name *:: struct*

Flag_Or_Name is a setting that is true, false, or a name, as gettext_compact is.

on: bool

name: string

the name, when one was given; on is then true.

```
project.Intersphinx_Project :: struct
```

Intersphinx_Project is one intersphinx_mapping entry: the name references use as a prefix, the base URL of the other project's pages, and where its inventory is, tried in order; 「」 stands for None, the base URL's objects.inv.

```
name: string
```

```
uri: string
```

```
locations: []string
```

```
project.Intl_Action :: enum u8
```

Intl_Action is what intl_update did to one catalog.

```
Create
```

```
Update
```

```
Not_Changed
```

```
project.Intl_File :: struct
```

Intl_File is one catalog intl_update or intl_stat looked at, with its statistics.

```
action: Intl_Action
```

```
path: string
```

```
stats: gettext.Stats
```

```
project.Intl_Options :: struct
```

Intl_Options are sphinx-intl's options for update and stat.

```
pot_dir: string
```

-p: the templates; default: the gettext builder's output.

```
locale_dir: string
```

-d: default: the first of locale_dirs.

languages: []string

-l, repeatable; default: the project's language.

width: int

-w: line width; 0 means 76.

obsolete: bool

keep obsolete messages (sphinx-intl's default).

project.Load_Options :: struct

Load_Options are the command line's say in where a project is and how it is set; load borrows them for the call.

conf_dir: string

-c: where conf.toml is, when not in the source directory.

no_config: bool

-C: build with defaults only.

overrides: []string

-D name=value, applied after conf.toml.

source: string

sphinx-build form: explicit source directory.

output: string

sphinx-build form: explicit output directory.

project.Migrate_Options :: struct

input: string

conf.py, guidedog.toml, or a folder holding one.

output: string

where conf.toml goes; 「」 is beside the input, 「-」 writes nothing.

force: bool

replace an existing conf.toml.

```
year: string
```

what datetime's current year evaluates to.

```
project.Migrate_Result :: struct
```

Migrate_Result is what migrate wrote (toml, and output unless that was 「-」), with notes on what it could not convert as written, and how many settings it converted.

```
input: string
```

```
output: string
```

```
toml: string
```

```
notes: []host.Report
```

```
converted: int
```

settings written to conf.toml.

```
attention: int
```

settings left out that may matter.

```
project.Pdf_Document :: struct
```

Pdf_Document is one book of pdf_documents: which document it starts from, and the file, title, and author it gets.

```
root: string
```

the root document of the book.

```
file: string
```

the PDF name in _build/pdf.

```
title: string
```

```
author: string
```

project.Project :: struct

Project is a loaded documentation project. The API is a handful of verbs over it:

```
p, problem := project.load("docs", {}, defaults.registry())
defer project.unload(&p)
if problem.exit != 0 do return problem.exit
result := project.build(&p, .Html, {})
defer project.release(&result)
project.clean(&p)
```

Who owns what each entry point allocates, all of it taken from context.allocator:

- **load**: the project owns everything, in an arena of its own; **unload** releases it, including the text of a problem load returned, so it is called whether load succeeded or not.
- **build**: the result owns everything the build allocated, in an arena of its own; **release** frees it. The project is never changed by a build (see **build**), so it may be built any number of times, and must outlive every result of its builds only while their reports are used. **load**, **build**, **release**, **unload** leaves nothing live.
- **serve**: each request works in an arena it frees; the rebuild callback owns (and releases) the results of the builds it runs.
- **clean**, **intl_update**, **intl_stat**, **migrate**, **quickstart**: their results and problems are allocated in context.allocator and not freed one by one; a caller on the heap runs them with an arena (or the temporary allocator) as context.allocator and frees it once it has used them.
- **builder_named**, **absolute**, **join**: plain values; **absolute** and **join** allocate the path.

config: Config

warnings: [dynamic]host.Report

from configuration; builds add their own.

registry: gd.Registry

settings: string

digest of conf.toml and the -D overrides: what every output uses.

memory: ^Memory

everything load allocated; see **unload**.

owner: mem.Allocator

the allocator memory came from.

```
project.Quickstart_Options :: struct
```

Quickstart_Options are guidedog quickstart's answers; 「」 takes each default.

```
dir: string
```

```
project: string
```

```
author: string
```

```
version: string
```

```
release: string
```

```
language: string
```

```
suffix: string
```

「.rst」 by default.

```
root_doc: string
```

「index」 by default.

```
separate: bool
```

source/ and build/ instead of one folder with _build.

```
year: string
```

for the copyright line.

```
date: string
```

for the root document's comment.

```
project.Server :: struct
```

Server previews the built html output on the loopback interface. Before serving a page it checks whether any source changed and rebuilds incrementally if so.

```
project: ^Project
```

```
rebuild: proc(s: ^Server) -> bool
```

returns false when the rebuild failed.

```
user: rawptr
```

```
stamp: i64
```

fingerprint of watched paths, sizes, and modification times.

```
project.Toml_Kind :: enum u8
```

TOML 1.0: tables, arrays of tables, dotted and quoted keys, inline tables, arrays, every string form, integers in all bases, floats, booleans, and date-times (kept as text). Errors carry the line, so a mistake in `conf.toml` points at itself.

```
String
```

```
Integer
```

```
Float
```

```
Boolean
```

```
Datetime
```

```
Array
```

```
Table
```

```
project.Toml_Value :: struct
```

`Toml_Value` is one parsed TOML value; `Config.html_context` keeps a table of them as written. Its strings and lists belong to the project that loaded it.

```
kind: Toml_Kind
```

```
text: string
```

String and Datetime.

```
integer: i64
```


float: f64

boolean: bool

items: [dynamic]Toml_Value

Array.

keys: [dynamic]string

Table, in definition order.

values: [dynamic]Toml_Value

Table, parallel to keys.

line: int

defined: bool

Table: defined by a header or as a value, not only implied.

inline: bool

written inline, so it may not be extended later.

Procedures

project.absolute :: *proc(path: string) -> string*

absolute makes a path absolute against the working directory without resolving it, so it works for folders a build has not created yet. The path is allocated in the context allocator and owned by the caller.

project.build :: *proc(p: ^Project, kind: Builder_Kind, o: Build_Options) -> Build_Result*

build reads, resolves, and writes the project with one builder. Outputs are written only for documents whose inputs changed, unless o.all is set.

Ownership: the build allocates everything, its result included, in an arena of its own taken from context.allocator, and release frees it: a caller releases each result once it is done with its reports. The build never changes the project: it works on its own copy of the configuration (build_config), so an –untrusted build confines only itself, and any number of builds may follow on the same project.

The memory of each step (a document read, a page rendered) comes from o.heap, the heap by default, and is freed when the step ends. When

context.allocator or o.heap refuses memory, the build stops with host.memory (exit 3) and publishes nothing.

```
project.build_limits :: proc(o: Build_Options) -> (gd.Limits,
host.Problem)
```

build_limits are the limits every session of a build reads and renders with: the documented defaults, with `-max-depth`, `-max-nodes`, and `-stack-kib`. Reading runs on the main thread, or on core:thread's threads with `-j`, and pages render on the main thread, so the stack budget may be at most what the smaller of those stacks holds (host.stack_budget_limit); without `-stack-kib` it is the default, or less on a thread too small for it. A `-stack-kib` above that is the problem build.stack (exit 2); a caller may call it before build to refuse the options early. Only the problem's message is allocated.

```
project.builder_named :: proc(name: string) -> (Builder_Kind,
bool)
```

builder_named accepts Guidedog's builder names and the Sphinx names that map onto them: the LaTeX builders become pdf, because Guidedog typesets PDF with Typst.

```
project.clean :: proc(p: ^Project) -> host.Problem
```

clean removes everything in the project's build directory, as `make clean` does for Sphinx, and leaves the directory itself; a missing one is nothing to clean. It stops at the first file it cannot remove and returns its problem (host.io), whose text is allocated in the caller's context allocator and owned by the caller.

```
project.intl_stat :: proc(p: ^Project, options: Intl_Options)
-> ([Intl_File, host.Problem)
```

intl_stat is sphinx-intl stat: the translated, fuzzy, and untranslated messages of every catalog of the languages. It writes nothing; the files are allocated in context.allocator (see Project).

```
project.intl_update :: proc(p: ^Project, options:
Intl_Options) -> ([Intl_File, host.Problem)
```

intl_update is sphinx-intl update: for each template of pot_dir and each language, it creates the translation catalog, or merges the template into the existing one (gettext.merge: kept, fuzzy, and obsolete messages, as msgmerge does). It returns what it did to each catalog, allocated in context.allocator (see Project), or the first problem, with the catalogs written before it kept.

```
project.join :: proc(parts: ..string) -> string
```

join joins path parts with the system's separator and cleans the result, as filepath.join does; the path is allocated in context.allocator.

```
project.load :: proc(dir: string, options: Load_Options,
registry: gd.Registry) -> (p: Project, problem: host.Problem)
```

load finds the project from dir (see find_project) or the explicit directories in options, and reads its configuration. The registry is checked first: an adapter with no ID, version, or procedure, or an ID registered twice, is a problem, as it is for gd.convert. Whether it has the readers the documents need is known once a build has found them (find_readers). The project owns everything load allocates, in an arena taken from context allocator, which unload frees; a problem's text lives in that arena too. When that allocator refuses memory, load stops with host.memory (exit 3), and unload is still what frees the rest.

```
project.migrate :: proc(o: Migrate_Options) -> (r:
Migrate_Result, problem: host.Problem)
```

migrate turns the Sphinx conf.py (or a guidedog.toml) that o.input names into conf.toml, and writes it unless o.output is `[-]`: literal assignments become settings, and what Python would have computed is reported in r.notes. The written conf.toml is loaded first, so it always builds; an existing one is refused unless o.force is set. Everything is allocated in context allocator (see Project); a failure is the problem, with nothing written.

```
project.quickstart :: proc(o: Quickstart_Options) ->
(created: []string, p: host.Problem)
```

quickstart creates a documentation project in o.dir laid out as Sphinx lays one out, with conf.toml, a root document, and the default templates copied in so they can be edited, and returns the paths it created. It refuses to overwrite a project. The paths and the problem are allocated in context allocator (see Project).

```
project.release :: proc(r: ^Build_Result)
```

release frees everything a build allocated, its result included, which is empty after.

```
project.serve :: proc(s: ^Server, port: int) -> host.Problem
```

serve previews the project's built html output on 127.0.0.1:port, rebuilding through s.rebuild before a page is served when a source changed since the last build. It borrows s and the project for as long as it runs, which is until the process ends: it returns only the problem of a port it cannot listen on. Each request works in an arena of its own on the heap, freed when it is answered.

```
project.unload :: proc(p: ^Project)
```

unload releases everything load allocated.

Constants

```
project.BUILD_BUDGET :: 1 << 30
```

BUILD_BUDGET bounds what the build holds at once: its sessions, and the files it reads: reading threads wait rather than exceed it (see `host.Budget`). – budget sets another.

```
project.CONFIG_FILE :: "conf.toml"
```

CONFIG_FILE is the name of a project's configuration, in its source directory.

Variables

```
@(rodata) project.BUILDER_NAMES
```

pyscan

Package `pyscan` analyses Python source statically, through tree-sitter: modules and their docstrings, `__all__`, and imports; classes with their bases, decorators, and members in source order; functions with their parameters as written; and the attribute documentation Sphinx's `ModuleAnalyzer` finds (`#:` comments and docstrings after assignments). Names resolve across the modules of the configured source roots, so a class re-exported by a package is found where it is defined. Nothing is executed.

It is a host library: it reads files (through a reader the host may confine) and allocates with the context allocator, freeing none of it, so a host runs it in an arena: the project and every module, class, and string it returns live until the arena is freed. `destroy` frees what tree-sitter holds, which no arena can.

Types

```
pyscan.All :: struct
```

All is a module's `__all__` when it can be read without running code.

```
names: [dynamic]string
```

```
defined: bool
```

the module assigns `__all__`.

```
known: bool
```

and its value was read: literal lists, `+=`, `append`, `extend`.

pyscan.Analysis :: struct

Analysis is what Sphinx's ModuleAnalyzer (sphinx.pycode.parser.Parser) finds in a module, keyed as it keys them: attribute documentation from #: comments and from string literals after assignments, annotations, definition order, @final, @overload.

attr_docs: map[Attr_Key][dynamic]string

lines, ending with 「」 .

attr_order: [dynamic]Attr_Key

first-insertion order of attr_docs.

annotations: map[Attr_Key]string

annotation_order: [dynamic]Attr_Key

first-insertion order of annotations.

tagorder: map[string]int

dotted qualname -> order of its last definition.

counter: int

finals: map[string]bool

overloads: map[string][dynamic]^Function

pyscan.Attr_Key :: struct

Attr_Key is (scope, name): scope is 「」 at module level, else the class's qualname.

scope: string

name: string

pyscan.Class :: struct

module: ^Module

name: string

qualname: string

node: ts.Node

bases: [dynamic]ts.Node

positional arguments of the class statement.

keywords: [dynamic]Keyword

metaclass=..., and others.

decorators: [dynamic]ts.Node

the decorator expressions.

doc: Doc

scope: Scope

outer: ^Class

the class it is nested in, or nil.

mro_done: bool

mro_complete: bool

mro_order: [dynamic]Mro_Entry

pyscan.**Def** :: struct

Def is one binding of a name in a module or class body, in source order.

name: string

kind: Def_Kind

node: ts.Node

the statement, or the definition.

runtime: bool

false under if TYPE_CHECKING:.

class: ^Class

function: ^Function

value: ts.Node

right-hand side; null when there is none.

annotation: ts.Node

the type node of an annotated assignment.

target: string

imports: the absolute module imported from, or imported.

member: string

Import_From: the name imported; 「*」 for a wildcard.

tuple: bool

bound by unpacking: its value is one part of value.

line: int

1-based line of the statement.

pyscan.**Def_Kind** :: *enum* u8

Class

Function

Assign

name = value, name: type = value.

Annotation

name: type, with no value.

Import

import a.b [as c].

Import_From

from m import name [as alias].

Type_Alias

type Name = value (PEP 695).

pyscan.**Doc** :: *struct*

Doc is a docstring as Python evaluates it (escapes processed), or none.

text: string

ok: bool

pyscan.**File_Read** :: *struct*

path: string

text: string

pyscan.**File_Reader** :: *struct*

File_Reader reads a file for the analyser. The host confines reads to what it may read; ok is false for a file that is missing or refused.

procedure: proc(user: rawptr, path: string) -> (text: string, ok: bool)

user: rawptr

pyscan.**Function** :: *struct*

module: ^Module

name: string

qualname: string

node: ts.Node

the function_definition, or the lambda.

is_async: bool

is_lambda: bool

decorators: [dynamic]ts.Node

params: [dynamic]Param

returns: ts.Node

the return type node, or null.

doc: Doc

class: ^Class

the class whose body defines it, or nil.

type_comment: bool

A PEP 484 type comment after the header, 「# type: (int, str) -> bool」: the parameter types as written (「...」 alone when elided) and the return type.

type_args: [dynamic]string

type_returns: string

pyscan.**Keyword** :: struct

name: string

value: ts.Node

pyscan.**Module** :: struct

project: ^Project

name: string

dotted.

kind: Module_Kind

path: string

the file read; the folder of a namespace package.

source: string

with line breaks normalised to \n.

tree: ^ts.Tree

root: ts.Node

doc: Doc

scope: Scope

all: All

info: Analysis

what Sphinx's ModuleAnalyzer finds.

broken: bool

tree-sitter found syntax errors.

too_deep: bool

it nests past MAX_SYNTAX_DEPTH, so it is not analysed (kind .Native).

origin: ^Module

A parsed snippet (parse_expression) names things in the namespace of origin.

pyscan.**Module_Kind** :: enum u8

Source

a .py file.

Package

a folder with __init__.py.

Namespace

a folder without `__init__.py` (PEP 420).

Native

a compiled extension (`.so`, `.pyd`): it cannot be read.

`pyscan.Mro_Entry` *:: struct*

`Mro_Entry` is one class of a method resolution order: a class in the roots, or one outside them known by path.

class: `^Class`

path: `string`

for a class outside the roots: 「builtins.object」, 「werkzeug.X」.

`pyscan.Object` *:: struct*

kind: `Object_Kind`

module: `^Module`

.Module: the module itself; else the module that defines it.

class: `^Class`

function: `^Function`

value: `ts.Node`

.Value: the expression assigned.

tuple: `bool`

.Value: bound by unpacking, so value is only its source.

path: `string`

.External, .Builtin: the dotted name.

wrapper: `Wrapper`

abstract: booldecorated with `abstractmethod`.**owner:** ^Class

the class whose body defines it, when found in one.

def: ^Def

the binding it was found through.

decorator: string

.Unknown wrapper: the decorator as written.

doc: Doca docstring given explicitly: `property(fget, doc=「 ...」 )`.**class_property:** bool

a property under classmethod: a property of the class.

partial_args: int`functools.partial` and `partialmethod`: how many leading parameters (after self, for `partialmethod`) the call already gave, which the signature leaves out.**partial:** bool**partial_method:** bool`partialmethod`: as got from the class, a function with no docstring.`pyscan.Object_Kind :: enum u8`**None**

not found.

Module**Class****Function**

Value

anything an assignment made: value is its expression.

External

from a module outside the source roots: only path is known.

Builtin

a builtin such as str or Exception: path is 「builtins.<name>」 .

```
pyscan.Param :: struct
```

name: string

kind: Param_Kind

annotation: ts.Node

the type node, or null.

default: ts.Node

the default expression, or null.

type_comment: string

「a, # type: str」 : a per-parameter type comment, or 「」 .

```
pyscan.Param_Kind :: enum u8
```

Positional_Only

Positional_Or_Keyword

Var_Positional

Keyword_Only

Var_Keyword

```
pyscan.Project :: struct
```

Project finds and analyses modules below its source roots, each once.

roots: []string

reader: File_Reader

a nil procedure reads with core:os.

parser: ^ts.Parser

modules: map[string]^Module

by dotted name; nil records a module not found.

files: [dynamic]File_Read

every source read, in order: a build's dependencies.

trees: [dynamic]^ts.Tree

steps: int

The lookup in progress: its steps (MAX_STEPS), where its stack starts (LOOKUP_STACK_BYTES), and how deep lookups and MRO computations are nested.

stack_base: uintptr

resolving: int

mro_depth: int

pyscan.**Scope** :: struct

defs: [dynamic]Def

pyscan.**Slot** :: struct

Slot is one entry of a class's `__slots__`, with the docstring a dictionary gives it.

name: string

doc: Doc

`pyscan.String_Kind :: enum u8`

Str

Bytes

Formatted

an f-string or t-string: not a constant.

`pyscan.Unparse_Mode :: enum u8`

Unparse_Mode chooses whose unparser to follow.

Sphinx

Sphinx's `sphinx.pycode.ast.unparse`, which `preserve_defaults` and `@overload` signatures use: numbers as written when source is given, lambda bodies as 「...」, and no parentheses added for precedence.

Sphinx_Repr

Sphinx's unparser without the source, as for overload signatures: numbers as repr.

Python

Python's own `ast.unparse`, which makes the annotation strings of `from __future__ import annotations`: parentheses kept, every expression.

`pyscan.Wrapper :: enum u8`

Wrapper is what the decorators of a function made of it.

None

Staticmethod

Classmethod

Property

Cached_Property

Unknown

a decorator from outside the source roots that is not known to keep functions.

Procedures

```
pyscan.base_objects :: proc(c: ^Class) -> [dynamic]Object
```

base_objects resolves the bases a class statement lists; subscripted generics give their origin.

```
pyscan.class_attr :: proc(c: ^Class, name: string, depth := 0) -> Object
```

class_attr is getattr(cls, name): the first class along the MRO whose body binds name.

```
pyscan.def_object :: proc(m: ^Module, owner: ^Class, d: ^Def, depth: int) -> Object
```

def_object turns a binding into the object it holds.

```
pyscan.destroy :: proc(p: ^Project)
```

destroy frees the trees and the parser. Everything else is in the caller's allocator.

```
pyscan.exported :: proc(m: ^Module, name: string) -> bool
```

exported is whether `from m import *` binds name: `__all__` when m has one, else the public names.

```
pyscan.find_def :: proc(s: ^Scope, name: string) -> ^Def
```

find_def returns the binding of name that is in force when the body has run: the last runtime one; nil when there is none.

```
pyscan.get_attr :: proc(obj: Object, name: string, depth := 0) -> Object
```

get_attr is getattr(obj, name) as far as the source tells.

```
pyscan.init :: proc(p: ^Project, roots: []string, reader := File_Reader{}) -> bool
```

init prepares a project over roots (folders, absolute or relative to the working directory). It returns false when tree-sitter cannot read the Python grammar.


```
pyscan.is_builtin :: proc(name: string) -> bool
```

is_builtin lists the names of Python's builtins module that matter to documentation.

```
pyscan.is_builtin_exception :: proc(name: string) -> bool
```

is_builtin_exception lists the builtin exception classes.

```
pyscan.is_exception :: proc(c: ^Class) -> bool
```

is_exception is whether a class derives from BaseException, as far as its MRO is known.

```
pyscan.lookup_name :: proc(code: ^Module, scope: ^Class,
name: string, depth := 0) -> Object
```

lookup_name resolves a name where a class body (scope) or the module runs: the class body first, then the module, then builtins.

```
pyscan.module :: proc(p: ^Project, name: string) -> ^Module
```

module finds, reads, and analyses a module by dotted name, once; nil when no root has it.

```
pyscan.module_attr :: proc(m: ^Module, name: string, depth :=
0) -> Object
```

module_attr looks a name up in a module's namespace, then among its submodules.

```
pyscan.module_object :: proc(m: ^Module) -> Object
```

module_object is the object of a module.

```
pyscan.mro :: proc(c: ^Class) -> (order: [dynamic]Mro_Entry,
complete: bool)
```

mro is the class's method resolution order, by C3 linearisation, ending with object. complete is false when a base could not be resolved or lies outside the roots (it then appears by path, and what it defines is unknown).

```
pyscan.number_repr :: proc(literal: string) -> string
```

number_repr is Python's repr of a numeric literal: integers in decimal, floats in their shortest round-trip form, imaginary numbers with j.

```
pyscan.parse_expression :: proc(m: ^Module, s: string) ->
(^Module, ts.Node, bool)
```

parse_expression parses text, such as a string annotation, as an expression whose names resolve in m. The module returned holds the snippet's source.

```
pyscan.prepare_docstring :: proc(s: string, tab_width := 8) -
> [dynamic]string
```

prepare_docstring is Sphinx's sphinx.util.docstrings.prepare_docstring: lines of reST with the common indentation of the lines after the first removed, the first line stripped, leading blank lines dropped, and a blank line at the end.

```
pyscan.reference :: proc(m: ^Module, scope: ^Class, n:
ts.Node, depth := 0) -> Object
```

reference evaluates an expression that names an object: a name, or attributes of one, or a subscripted generic (its origin). Other expressions give None.

```
pyscan.repr_bytes :: proc(s: string) -> string
```

repr_bytes is Python's repr of bytes.

```
pyscan.repr_str :: proc(s: string) -> string
```

repr_str is Python's repr of a str: single quotes unless the text has one and no double quote, and escapes for what is not printable.

```
pyscan.slots :: proc(c: ^Class) -> (out: [dynamic]Slot, ok:
bool)
```

slots is inspect's getslots: the __slots__ the class has (its own, or the nearest base's), as a list, tuple, dictionary, or single string of names. ok is false when there are none, or they are computed.

```
pyscan.split_lines :: proc(s: string) -> [dynamic]string
```

split_lines is str.splitlines for \n, \r\n, \r and the other line boundaries Python knows; a final line break does not start another line.

```
pyscan.str_value :: proc(m: ^Module, n: ts.Node) -> (string,
bool)
```

str_value is string_value for str constants only.

```
pyscan.string_value :: proc(m: ^Module, n: ts.Node) ->
(value: string, kind: String_Kind, ok: bool)
```

string_value evaluates a string or concatenated_string node as Python does: prefixes, escape sequences, and adjacent literals joined. ok is false for other nodes.

```
pyscan.text :: proc(m: ^Module, n: ts.Node) -> string
```

text is the source a node spans.

```
pyscan.try_unparse :: proc(m: ^Module, n: ts.Node, mode :=
Unparse_Mode.Sphinx) -> (string, bool)
```

try_unparse is unparse that reports expressions Sphinx's unparser refuses (comparisons, comprehensions, f-strings, conditional expressions...).

```
pyscan.unparse :: proc(m: ^Module, n: ts.Node, mode :=
Unparse_Mode.Sphinx) -> string
```

unparse writes an expression back as Sphinx's unparser does (Sphinx mode).

readers/commonmark

Package commonmark reads Markdown into a Guidedoc document without calling an allocator, as two readers with distinct ids: 「commonmark」 (reader, read, read_inline) is pure CommonMark 0.31.2 and takes no settings; 「myst」 (myst_reader, read_myst, read_myst_inline) is the MyST Markdown that Sphinx projects write (myst-parser 0.16.1): directives, roles, targets, comments, front matter, footnotes, GFM tables, and the optional syntax that Settings.enable_extensions turns on.

Block structure is parsed line by line into a skeleton in scratch; link reference definitions are collected over the whole document; then the skeleton is emitted in preorder, with inline content parsed per paragraph and heading. Directive content is parsed the same way when the directive asks for it (see extension.odin), so a parse works on line views of any source.

Types

```
commonmark.Directive :: struct
```

Directive is a directive the reader has split into its parts. Lines are views of source.

```
name: string
```

```
name_span: gd.Source_Span
```

span: `gd.Source_Span`

source: `gd.Source_Id`

arg: `[]Line`

the argument; empty when there is none.

options: `[]Option`

content: `[]Line`

sections: `bool`

the directive stands where sections are allowed.

classes: `gd.Text`

classes a handler leaves for the next element (Docutils' `class` directive without content); the reader gives them to the next block it emits.

`commonmark.Directive_Handler` *:: struct*

name: `string`

compared case-insensitively.

run: `Directive_Proc`

`commonmark.Directive_Proc` *:: #type proc(h: ^Host, d: ^Directive, user: rawptr) -> Directive_Result*

`Directive_Proc` runs a parsed directive; `user` is `Extension.user`. The directive and its lines are borrowed for the call.

`commonmark.Directive_Result` *:: enum u8*

Ran

Unknown

the handler does not know the name; the reader reports it.

Failed

the handler refused the directive and reported why.

```
commonmark.Directive_Shape :: struct
```

Directive_Shape tells the reader how a directive takes its first line: as an argument, or, for a directive without arguments, as the first line of content; and whether it takes options at all.

```
argument: bool
```

```
options: bool
```

it has options; without, an option block stays content.

```
commonmark.Extension :: struct
```

```
directives: []Directive_Handler
```

```
directive: Directive_Proc
```

directive is offered every directive the table does not list.

```
shape: proc(h: ^Host, name: string, user: rawptr) ->
(Directive_Shape, bool)
```

shape describes a directive by name; found is false for an unknown directive.

```
roles: []Role_Handler
```

```
role: Role_Proc
```

role is offered every role the table does not list.

```
table: proc(h: ^Host, id: gd.Node_Id, span:
gd.Source_Span, columns: int, user: rawptr) -> bool
```

table runs after the reader opens a Table node, before its rows, so a table directive around it can add its title and attributes; it returns true when it did, and the reader then adds no attributes of its own.

```
begin: proc(h: ^Host, user: rawptr)
```

before the document's blocks.

```
end: proc(h: ^Host, user: rawptr)
```

after them.

```
resolve: proc(h: ^Host, user: rawptr) -> gd.Status
```

resolve, when set, links references, targets, and footnotes instead of the reader's own resolution (resolve.odin); the Sphinx layer resolves as for reStructuredText.

```
user: rawptr
```

```
commonmark.Host :: struct
```

Host is the reader's document-wide state, shared by the parse of the document and the parses of nested content. Directive and role handlers receive it and treat it as opaque: they reach the read through context_of and the procedures below.

```
ctx: ^gd.Read_Context
```

```
b: ^gd.Builder
```

```
settings: Settings
```

```
ext: ^Extension
```

```
myst: bool
```

the MyST syntax is on: the 「myst」 reader, not 「commonmark」 .

```
disabled: Rules
```

myst_disable_syntax.

```
syntax: Syntax_Extensions
```

myst_enable_extensions in force.

```
refs: []Ref
```

Link reference definitions, collected over the whole document.

```
nrefs: int
```

```
names: gd.Symbol_Table
```

```
labels: gd.Buffer
```

```
label_tmp: []u8
```

```
html_format: gd.Text
```

```
rel_images: bool
```

MyST include options in force: rewrite image paths, and link paths starting with `rel_docs`, of included sources to be relative to the main source.

```
rel_docs: string
```

```
using state: Myst_State
```

```
commonmark.Line :: gd.Line_View
```

Line is a view of one input line: its text without the line break, and the source offset of `text[o]`. Directive handlers pass nested content as lines.

```
commonmark.Meta :: struct
```

Meta is one HTML meta tag: key as MyST writes it (「description lang=en」 , 「http-equiv=Content-Type」 , 「keywords」) and its content.

```
key: string
```

```
content: string
```

```
commonmark.Option :: struct
```

Option is one directive option: its name and value lines. A value written as a YAML scalar has its quoting and folding processed; its lines may then be copies in scratch, whose offset is where the value starts in the source.

```
name: string
```

```
value: []Line
```

```
span: gd.Source_Span
```

```
commonmark.Role :: struct
```

Role is one role, `{name}`content``. lines hold the whole construct, split at line breaks; content is the range of the content in the lines joined by 「\n」 .

```
name: string
```

```
lines: []Line
```

```
content: [2]int
```

```
span: gd.Source_Span
```

```
source: gd.Source_Id
```

```
commonmark.Role_Handler :: struct
```

```
name: string
```

compared case-insensitively.

```
run: Role_Proc
```

```
commonmark.Role_Proc :: #type proc(h: ^Host, r: ^Role, user: rawptr) -> bool
```

Role_Proc handles a role; it returns false for a role it does not know.

```
commonmark.Rule :: enum u8
```

Rule names a syntax element that `myst_disable_syntax` can turn off, by markdown-it's rule name.

```
Table
```

```
Code
```

```
Fence
```

```
Blockquote
```

```
Hr
```

```
List
```

```
Reference
```

Html_Block

Heading

Lheading

Emphasis

Backticks

Link

Image

Autolink

Html_Inline

Entity

Escape

Newline

Front_Matter

Myst_Target

Myst_Line_Comment

Myst_Block_Break

Myst_Role

Math_Block

Math_Inline

Footnote_Def**Footnote_Ref****Myst_Directive**

not a markdown-it rule: {name} fences (off with `commonmark_only`).

```
commonmark.Settings :: struct
```

Settings are the 「myst」 reader's settings: myst-parser 0.16.1's `myst_*` configuration values, named without the prefix. They reach the reader through `gd.Read_Options.extension`, made with `gd.read_extension(&settings)`; an empty extension means `default_settings()`. The 「commonmark」 reader (pure CommonMark) takes none. A caller that supplies Settings starts from `default_settings()`, because several defaults are not zero.

extension: ^Extension

Directive and role handlers (the Sphinx layer provides them); nil means none, and directives and roles are then reported as unknown.

commonmark_only: bool

`myst_commonmark_only`: CommonMark syntax only (no directives, roles, or other MyST syntax), but a document's tree as for MyST: headings make sections.

enable_extensions: Syntax_Extensions

`myst_enable_extensions`: the optional syntax. Default `{.Dollarmath}`.

disable_syntax: []string

`myst_disable_syntax`: markdown-it rule names whose syntax is not parsed, e.g. 「emphasis」, 「table」, 「html_block」 (see `syntax_rule`).

url_schemes: []string

`myst_url_schemes`: schemes of `[text](scheme:...)` links that are external; other destinations are cross-references. nil recognizes every scheme. Default `http`, `https`, `mailto`, and `ftp`.

linkify_fuzzy_links: bool

`myst_linkify_fuzzy_links`: linkify recognizes URLs without a scheme. Default `true`.

heading_anchors: int

myst_heading_anchors: headings up to this level get a GitHub-style slug anchor that `[](#slug)` and `[](file.md#slug)` reach. 0 means none.

heading_slug_func: string

myst_heading_slug_func is a Python callable and cannot be supported; a non-empty value is reported and the built-in slugs are used.

substitutions: []Substitution

myst_substitutions: global substitutions; front matter overrides them.

sub_delimiters: [2]u8

myst_sub_delimiters: the characters doubled around substitution references. Default `{『{『, 『}』}`, giving `{{ name }}`.

html_meta: []Meta

myst_html_meta: HTML meta tags for every document; front matter html_meta overrides entries with the same key.

footnote_transition: bool

myst_footnote_transition: a transition of class `「footnotes」` precedes the footnotes at the end of the document. Default true.

dmath_allow_labels: bool

myst_dmath_allow_labels: `$$...$$ (label)`. Default true.

dmath_allow_space: bool

myst_dmath_allow_space: inline math may start or end with a space. Default true.

dmath_allow_digits: bool

myst_dmath_allow_digits: inline math may follow or precede a digit. Default true.

dmath_double_inline: bool

myst_dmath_double_inline: `$$...$$` inside a paragraph. Default false.

words_per_minute: int

myst_words_per_minute: for the wordcount-minutes substitution. Default 200.

docname: string

The document's name (「usage/intro」) and configuration values, which substitutions reach as env.docname and env.config.<name>.

env_config: []Substitution

today: string

The {sub-ref} today: the project's formatted date; empty is Read_Options.today.

commonmark.**Substitution** :: struct

A substitution's value is Markdown, or a mapping written as nested YAML that dotted names reach (「key.sub」).

name: string

value: string

commonmark.**Syntax_Extension** :: enum u8

Syntax_Extension names an entry of myst_enable_extensions.

Amsmath

Colon_Fence

Deflist

Dollarmath

Fieldlist

Html_Admonition

Html_Image

Linkify**Replacements****Smartquotes****Substitution****Tasklist**

```
commonmark.Syntax_Extensions :: bit_set[Syntax_Extension;
u16]
```

Procedures

```
commonmark.context_of :: proc(h: ^Host) -> ^gd.Read_Context
```

context_of returns the read context, for handlers that build nodes.

```
commonmark.default_settings :: proc() -> Settings
```

default_settings returns myst-parser 0.16.1's defaults. The value borrows only static data (the default URL schemes); fields a caller sets must outlive the read.

```
commonmark.include_paths :: proc(h: ^Host, images: bool,
docs: string) -> (bool, string)
```

include_paths sets MyST's include options :relative-images: and :relative-docs: for the content of the include directive being run; it returns the previous values.

```
commonmark.list_items :: proc(h: ^Host, lines: []Line,
source: gd.Source_Id) -> (items: [][]Line, starts: []int,
ok: bool)
```

list_items splits lines that hold one bullet list into the content lines of its items, and where each item starts in lines (for list-table). ok is false when the lines are anything but one bullet list. The results live in scratch.

```
commonmark.myst_reader :: proc() -> gd.Reader
```

myst_reader returns the 「myst」 reader by value: MyST Markdown with ^Settings.

```
commonmark.parse_body :: proc(h: ^Host, lines: []Line,
source: gd.Source_Id, sections := false, classes :=
gd.Text{})
```

`parse_body` parses lines of a source as Markdown body content of the builder's open node. `sections` lets headings open sections, as at the top of a document; `classes`, when set, go to every top-level element.

```
commonmark.parse_inline :: proc(h: ^Host, lines: []Line,
source: gd.Source_Id)
```

`parse_inline` parses lines of a source as Markdown inline content of the builder's open node.

```
commonmark.read :: proc(ctx: ^gd.Read_Context) -> gd.Status
```

`read` reads `ctx.source` as pure CommonMark 0.31.2 (`gd.Read_Proc`): headings, no sections. Any settings in the read options are refused with `Invalid_Input`. It borrows `ctx` for the call; its skeleton lives in scratch, released before it returns. Problems go to `ctx.reports`; the result is the builder's status (`.Capacity` when a table is full).

```
commonmark.read_inline :: proc(ctx: ^gd.Read_Context, span:
gd.Source_Span) -> gd.Status
```

`read_inline` parses a span of `ctx.source` as CommonMark inline content of the builder's open node, for a translated message (`gd.Read_Inline_Proc`). Link reference definitions and footnote definitions of the surrounding document are not known to it.

```
commonmark.read_myst :: proc(ctx: ^gd.Read_Context) ->
gd.Status
```

`read_myst` reads `ctx.source` as MyST Markdown (`gd.Read_Proc`), with the `^Settings` the read options carry (`gd.read_extension`), or `default_settings` when they carry none. Settings of another type are refused with `Invalid_Input`. Ownership and failures are read's; the settings are borrowed for the call.

```
commonmark.read_myst_inline :: proc(ctx: ^gd.Read_Context,
span: gd.Source_Span) -> gd.Status
```

`read_myst_inline` is `read_inline` for MyST, with the settings of `ctx.options.extension` (roles included), as `read_myst` takes them.

```
commonmark.reader :: proc() -> gd.Reader
```

`reader` returns the 「commonmark」 reader by value: pure CommonMark 0.31.2, no settings.

```
commonmark.syntax_extension_named :: proc(name: string) ->
(Syntax_Extension, bool)
```

syntax_extension_named finds a myst_enable_extensions entry by its name.

```
commonmark.syntax_rule :: proc(name: string) -> (Rule, bool)
```

syntax_rule finds a myst_disable_syntax entry by its markdown-it rule name.

Constants

```
commonmark.CONFORMANCE :: gd.Conformance.Conformant
```

```
commonmark.MYST_CONFORMANCE :: gd.Conformance.Partial
```

```
commonmark.MYST_VERSION :: "0.16.1"
```

MyST follows myst-parser 0.16.1, with the differences CONFORMANCE.md documents.

```
commonmark.VERSION :: "0.31.2"
```

readers/rst

Package rst reads reStructuredText as the Docutils markup specification defines it, implemented independently. It owns lines, block structure, and directives; inline markup and reference resolution live in the inline subpackage.

Types

```
rst.Arg_Rule :: enum u8
```

None

Optional

Required

```
rst.Content_Rule :: enum u8
```

None

Optional

Required

```
rst.Dir_Opt :: struct
```

```
spec: Opt_Spec
```

```
lines: []Line
```

the value: the rest of the marker line, then continuation lines.

```
span: gd.Source_Span
```

```
rst.Dir_Spec :: struct
```

```
name: string
```

```
kind: Dir_Kind
```

```
arg: Arg_Rule
```

```
final_ws: bool
```

```
content: Content_Rule
```

```
opts: Opt_Set
```

```
value: gd.Admonition_Kind
```

```
rst.Directive :: struct
```

```
spec: ^Dir_Spec
```

```
span: gd.Source_Span
```

```
name_span: gd.Source_Span
```

```
arg: []Line
```

argument lines, trimmed; empty when there is no argument.

```
opts: [MAX_OPTIONS]Dir_Opt
```

```
nopts: int
```


content: []Line

sections: bool

subst: bool

inside a substitution definition.

ext: ^Directive_Handler

a dialect's handler, or nil for a standard directive.

arg_block: []Line

the argument lines with their relative indentation.

subst_name: string

the name of the substitution definition it is in, as written.

rst.Directive_Handler :: struct

name: string

compared case-insensitively, as directive names are.

arg: Arg_Rule

final_ws: bool

the last argument may contain white space.

content: Content_Rule

options: []Opt_Spec

run: Directive_Proc

rst.Directive_Proc :: #type proc(p: ^Parser, d: ^Directive, user: rawptr)

Directive_Proc runs a parsed directive. user is Extension.user.

rst.Directive_Shape :: struct

Directive_Shape tells a host how a directive takes its parts.

arg: Arg_Rule

final_ws: bool

the last argument may contain white space.

content: Content_Rule

options: []Opt_Spec

rst.Extension :: struct

directives: []Directive_Handler

lookup: proc(name: string, user: rawptr) ->
^Directive_Handler

lookup, when set, finds handlers the table does not list, such as names that depend on reader state (a default domain). It returns nil for unknown names.

roles: []Role_Handler

role: Role_Proc

role, when set, is offered every role the table does not list.

begin: Hook_Proc

before the main source's blocks (reader state).

prologue: Hook_Proc

prologue runs after the main source's leading field lines, where Sphinx inserts rst_prolog: after a document's bibliographic fields, else before everything.

end: Hook_Proc

after them, before sections close (e.g. an epilogue).

substitution: proc(name: string, user: rawptr) -> (text:
string, ok: bool)

substitution, when set, supplies default substitutions: resolution offers it each reference whose name no definition in the document has (see

`rst_inline.Substitution_Hook`). It runs after parsing, with the extension's user.

```
user: rawptr
```

```
rst.Frame :: struct
```

```
src: gd.Source_Id
```

```
text: string
```

```
tab: int
```

```
rst.Hook_Proc :: #type proc(p: ^Parser, user: rawptr)
```

`Hook_Proc` runs at a fixed point of the read, with the reader's state.

```
rst.Host :: struct
```

Host procedures receive `Host.user`. Lines are views of the current source (`p.src`), as the block parser's are, so every node keeps exact spans.

```
body: proc(p: ^Parser, lines: []Line, sections: bool,  
each: gd.Text, user: rawptr)
```

`body` parses nested body content of the open node; `sections` allows headings to make sections, and `each`, when set, gives these classes to every top-level element (the class directive).

```
inline: proc(p: ^Parser, lines: []Line, user: rawptr)
```

`inline` parses inline content of the open node.

```
items: proc(p: ^Parser, lines: []Line, user: rawptr) ->  
(items: [][]Line, starts: []int, ok: bool)
```

`items` splits lines that hold one bullet list into its items' content, with where each item starts in lines (list-table, with the host's list syntax); `ok` is false when the lines are not one bullet list.

```
user: rawptr
```

```
rst.Host_Directive :: struct
```

`Host_Directive` is a directive the host has split into its parts.

name: string

name_span: gd.Source_Span

span: gd.Source_Span

arg: []Line

the argument lines; empty when there is none.

options: []Host_Option

content: []Line

sections: bool

the directive stands where sections are allowed.

rst.Host_Option :: struct

Host_Option is a directive option as the host wrote it: its name and value lines.

name: string

value: []Line

span: gd.Source_Span

rst.Hosted_Result :: enum u8

Ran

Unknown

no directive of that name; nothing was reported.

Failed

the arguments, options, or content were refused and reported.

rst.Line :: struct

Line is a view of one input line in columns: tabs are expanded to tab stops and trailing white space is stripped. A line that held a tab, form feed, or vertical tab is a copy in scratch; raw then keeps the source line so columns map back

to source bytes. Nested blocks use cut views of their parent's lines, so every node keeps exact spans.

text: string

raw: string

the source line of an expanded copy, else 「」.

offset: u32

source offset of text[o]; of raw[o] when raw is set.

col: i32

column of text[o] in the expanded line when raw is set.

rst.Opt_Spec :: struct

name: string

type: Opt_Type

choices: string

space-separated, for Choice.

rst.Opt_Type :: enum u8

Flag

Text

Text_Required

Int

Nonneg_Int

Pos_Int

Opt_Int

Length**Length_Percent****Figwidth****Percentage****Class****Name****Choice****Encoding****Char****Widths****Path****rst.Parser** *:: struct***ctx:** ^gd.Read_Context**b:** ^gd.Builder**inl:** rst_inline.Inline_State**src:** gd.Source_Id

the source the current lines come from.

text: string**tab:** int**depth:** int

stack: `gd.Stack_Guard`

bounds the machine stack of the recursion (enter).

styles: `[MAX_LEVELS]Style`

styles_used: `int`

sections: `[MAX_LEVELS]Section_Entry`

sections_used: `int`

section_base: `int`

sections open before titled content (`parse_titled`) began.

level_base: `int`

the level of the last of them: titled content's sections nest in it.

top_count: `int`

top-level elements since the current section began.

top_transition: `bool`

the last top-level element was a transition.

transition_span: `gd.Source_Span`

pending: `gd.Text`

classes from a class directive, for the next element.

pending_span: `gd.Source_Span`

has_pending: `bool`

in_sidebar: `int`

ext: `^Extension`

a dialect's directives and roles; see `extension.odin`.

host: `^Host`

another markup that uses the directives; see `host.odin`.

hosted: `bool`

the running directive was written in the host markup.

table_extra: `^Table_Extra`

a table directive's title and options for the host.

`rst.Role_Handler` `:: struct`

name: `string`

compared case-insensitively.

run: `Role_Proc`

`rst.Role_Proc` `:: #type proc(p: ^Parser, call: ^rst_inline.Role_Call, user: rawptr) -> bool`

`Role_Proc` handles interpreted text; it returns false for a role it does not know.

`rst.Settings` `:: struct`

Settings are the reader's dialect settings. They reach the reader through `gd.Read_Options.extension`, made with `gd.read_extension(&settings)`; settings of another type are refused. A dialect's own configuration embeds `Settings` and passes a pointer to that embedded field when it runs this reader. An empty extension means defaults (`default_settings`); a caller that supplies `Settings` starts from `default_settings`, because the zero value turns bibliographic fields off.

extension: `^Extension`

Directive and role handlers consulted before the standard ones.

host: `^Host`

Another markup (MyST Markdown) that reads the main source and nested directive content with these directives and roles; see `host.odin`.

pep_references: `bool`

Implicit 「PEP nnn」 and 「RFC nnn」 references in running text become links.

rfc_references: bool

pep_base_url: string

「」 means <https://peps.python.org/>

rfc_base_url: string

「」 means <https://datatracker.ietf.org/doc/html/>

trim_footnote_reference_space: bool

White space before a footnote reference is removed.

smart_quotes: Smart_Quotes

Typographic quotation marks, apostrophes, dashes, and ellipses in text.

smart_quotes_locales: string

Quotation marks per language, overriding the built-in ones, as 「tag: “”’,tag2:«»<>」 : primary open and close, then secondary open and close.

language: string

BCP 47 tag of the document language: bibliographic field names, generated titles, and quotation marks. 「」 means 「en」 .

docinfo: bool

A field list first in the document becomes bibliographic data (docinfo).

sectsubtitle: bool

A lone subsection title becomes the section’s subtitle.

tab_width: int

Tab stops; zero means `gd.Read_Options.tab_width`, which zero makes 8.

character_level_inline_markup: bool

Inline markup is recognized inside words (Docutils’ `character_level_inline_markup`).

`rst.Smart_Quotes :: rst_inline.Smart_Quotes`

```
rst.Table_Extra :: struct
```

Table_Extra carries what table directives add to a table: a title and attributes.

```
title: []Line
```

```
classes: gd.Text
```

```
name: gd.Text
```

```
align: string
```

```
width: string
```

```
widths: []int
```

from :widths:: nil keeps the table's own widths.

```
auto_widths: bool
```

:widths: auto leaves widths to the renderer.

```
width_class: string
```

「colwidths-auto」 or 「colwidths-given」 from :widths:.

```
found: bool
```

set by the table parser when it emits a table.

Procedures

```
rst.add_common :: proc(p: ^Parser, id: gd.Node_Id, d: ^Directive, class_opt := "class")
```

add_common adds the :class: and :name: options as attributes. Directive nodes keep their Name attribute for the directive name, so :name: becomes Custom 「name」 .

```
rst.blank :: proc(l: Line) -> bool
```

blank tells whether l has no text (white space is stripped when lines are split).

```
rst.bullet_width :: proc(t: string) -> int
```

bullet_width returns the byte length of a bullet that starts t, or 0.

```
rst.class_directive :: proc(p: ^Parser, d: ^Directive)
```

class_directive runs the class directive: its classes go to the content's elements, or, without content, to the next element; invalid names are reported.

```
rst.close :: proc(p: ^Parser, id: gd.Node_Id)
```

close ends the element open returned id for.

```
rst.code_text :: proc(p: ^Parser, lines: []Line) -> gd.Text
```

code_text writes lines to the text pool, each ending in 「\n」 .

```
rst.custom :: proc(p: ^Parser, id: gd.Node_Id, name, value: string)
```

custom adds a Custom attribute name=value to node id; both strings are copied to the text pool. A full pool poisons the builder, as every builder write does.

```
rst.cut :: proc(l: Line, k: int) -> Line
```

cut drops the first k columns.

```
rst.dedent :: proc(lines: []Line)
```

dedent strips the least indentation of the non-blank lines, in place.

```
rst.default_settings :: proc() -> Settings
```

default_settings returns Docutils' defaults: bibliographic fields on, everything else off or empty. The value owns nothing.

```
rst.directive_node :: proc(p: ^Parser, d: ^Directive, name: string) -> gd.Node_Id
```

directive_node opens a generic Directive node named after the directive.

```
rst.directive_shape :: proc(p: ^Parser, name: string) -> (shape: Directive_Shape, found: bool)
```

directive_shape looks a directive up as the reader would, a dialect's handlers first.

```
rst.embed_begin :: proc(p: ^Parser, ctx: ^gd.Read_Context)
```

embed_begin prepares p for a host that reads the main source itself: the settings of ctx.options.extension apply as for read, and the dialect's begin hook runs.

```
rst.embed_end :: proc(p: ^Parser) -> gd.Status
```

embed_end runs the dialect's end hook and resolves the document as read does.

```
rst.enter_source :: proc(p: ^Parser, id: gd.Source_Id, tab: int) -> Frame
```

enter_source makes lines of another source current, for spans and inline parsing.

```
rst.find_role :: proc(p: ^Parser, name: string) -> int
```

find_role returns the index of the role the document defined with the role directive under name, in p.inl.roles, or -1. The search spends work budget.

```
rst.get_opt :: proc(d: ^Directive, name: string) -> ^Dir_Opt
```

get_opt returns the option called name that the directive was given, or nil. The result points into d and lives as long as it.

```
rst.has_option :: proc(d: ^Directive, name: string) -> bool
```

has_option reports whether a directive was given an option, flags included.

```
rst.head :: proc(l: Line, n: int) -> Line
```

head keeps the first n columns, without trailing spaces.

```
rst.host_lines :: proc(p: ^Parser, views: []rst_inline.Line) -> ([]Line, bool)
```

host_lines makes lines of views of source text (a host's lines): tabs are expanded to the parser's tab stops, as split_lines does.

```
rst.host_views :: proc(p: ^Parser, lines: []Line) -> ([]rst_inline.Line, bool)
```

host_views gives the host the source text of lines, tabs unexpanded (inline_view).

```
rst.indent_of :: proc(s: string) -> int
```

indent_of counts the spaces s starts with.

```
rst.inline :: proc(p: ^Parser, lines: []Line)
```

inline parses lines as inline content of the currently open node.

```
rst.inline_node :: proc(p: ^Parser, kind: gd.Node_Kind,
lines: []Line)
```

`inline_node` adds a node of kind holding lines parsed as inline content (`inline`); nothing for no lines.

```
rst.is_indented :: proc(l: Line) -> bool
```

`is_indented` tells whether `l` starts with a space.

```
rst.join_lines :: proc(p: ^Parser, lines: []Line, sep:
string) -> string
```

`join_lines` joins lines into a scratch copy, or returns the single line's text.

```
rst.leave_source :: proc(p: ^Parser, f: Frame)
```

`leave_source` makes the source that `enter_source` returned `f` for current again.

```
rst.line_span :: proc(p: ^Parser, l: Line) -> gd.Source_Span
```

`line_span` is the source span of one line (`span_of`).

```
rst.lines_span :: proc(p: ^Parser, lines: []Line) ->
gd.Source_Span
```

`lines_span` is the source span of lines without trailing blank lines; empty lines give an empty span at the start of the source.

```
rst.mark :: proc(p: ^Parser) -> gd.Scratch_Mark
```

`mark` records the scratch position; `release` returns scratch to it. Pair them with `defer` around work that takes scratch.

```
rst.open :: proc(p: ^Parser, kind: gd.Node_Kind, span:
gd.Source_Span) -> gd.Node_Id
```

`open` begins an element and gives it the classes of a preceding class directive.

```
rst.opt_value :: proc(p: ^Parser, d: ^Directive, name:
string) -> (string, bool)
```

`opt_value` returns an option's value lines joined by spaces and trimmed; `ok` is false when the option was not given. The text is a view of the source or a scratch copy, valid until the caller's scratch mark is released.

```
rst.option_lines :: proc(d: ^Directive, name: string) ->
([]Line, bool)
```

`option_lines` returns the value lines of a directive option, or nil.

```
rst.parse_blocks :: proc(p: ^Parser, lines: []Line,
sections := false, each := gd.Text{})
```

`parse_blocks` parses body elements. `sections` allows titles and transitions; each gives classes to every element parsed at this level (class directive content).

```
rst.parse_body :: proc(p: ^Parser, lines: []Line, sections :=
false, each := gd.Text{})
```

`parse_body` parses nested body content of the open node in the markup of the running directive (see `Host`). Outside hosted directives it is `parse_blocks`.

```
rst.parse_source :: proc(p: ^Parser, id: gd.Source_Id,
sections: bool)
```

`parse_source` parses a whole registered source (`gd.add_source`) as body content at the current position, as if its lines stood there. `sections` allows titles.

```
rst.parse_titled :: proc(p: ^Parser, lines: []Line)
```

`parse_titled` parses content in which titles begin sections, in a title context of its own (Sphinx's nested parse with titles): adornments take levels afresh, the first nesting in the innermost open section, and the sections close when the content ends. The surrounding adornments are kept for what follows.

```
rst.path_of :: proc(p: ^Parser, lines: []Line) -> string
```

`path_of` joins path lines without white space, as the specification treats paths.

```
rst.place_titled :: proc(p: ^Parser, lines: []Line)
```

`place_titled` prepares for titled content (`parse_titled`) as Sphinx's only directive does: when the content's first title has an adornment the document already uses, the content stands where a title of that level would, so deeper sections close first.

```
rst.pool_classes :: proc(p: ^Parser, s: string) -> (gd.Text,
bool)
```

`pool_classes` writes white-space separated class names as identifiers, one space apart. It fails when a name has no letters.

```
rst.pool_lines :: proc(p: ^Parser, lines: []Line, sep:
string) -> gd.Text
```

`pool_lines` writes the lines joined by `sep` into the text pool and returns the text, which lives as long as the document; a full pool poisons the builder.

```
rst.pool_lower :: proc(p: ^Parser, s: string) -> string
```

pool_lower writes s in lower case to the text pool and returns the pooled string.

```
rst.push_lines :: proc(p: ^Parser, lines: []Line, sep: string)
```

push_lines writes the lines joined by sep into the text pool.

```
rst.read :: proc(ctx: ^gd.Read_Context) -> gd.Status
```

read reads ctx.source as reStructuredText into ctx.b (gd.Read_Proc). Settings of another type than ^Settings are refused with Invalid_Input before anything is read. It borrows ctx for the call and keeps nothing; its tables live in scratch, released before it returns. Problems go to ctx.reports; the result is the builder's status (.Capacity when a table is full, .Limit past the nesting or stack limits), or that of reference resolution.

```
rst.read_inline :: proc(ctx: ^gd.Read_Context, span: gd.Source_Span) -> gd.Status
```

read_inline parses a span of ctx.source as inline content of the builder's open node, with the settings of ctx.options.extension, as Sphinx parses a translated message (gd.Read_Inline_Proc). Each line's leading white space is dropped, as the lines of a paragraph lose their indentation. References are left for the caller to resolve.

```
rst.reader :: proc() -> gd.Reader
```

reader returns the 「rst」 reader descriptor, for a gd.Registry. It is a value with no state: registering it twice or copying it is harmless.

```
rst.release :: proc(p: ^Parser, m: gd.Scratch_Mark)
```

release frees the scratch taken since mark m; views into it become invalid.

```
rst.report :: proc(p: ^Parser, m: gd.Message, span: gd.Source_Span, a := gd.Diagnostic_Arg{}, b := gd.Diagnostic_Arg{})
```

report adds diagnostic m at span, with its arguments, to the read's reports; a full reports region counts it as suppressed (gd.report) rather than failing the read.

```
rst.run_hosted :: proc(p: ^Parser, h: Host_Directive) ->
Hosted_Result
```

run_hosted checks and runs a directive written in the host markup. Options are checked as in reStructuredText; a host with other rules checks them first (directive_shape, option_problem) and passes only the ones it keeps.

```
rst.span_of :: proc(p: ^Parser, first, last: Line) ->
gd.Source_Span
```

span_of is the source span from the start of first to the end of last, in the current source, clamped to its text.

```
rst.standard_role :: proc(name: string) ->
(rst_inline.Role_Base, bool)
```

standard_role finds a standard role or alias by name, compared case-insensitively; ok is false (base .Custom) for another name.

```
rst.table_decorate :: proc(p: ^Parser, id: gd.Node_Id,
span: gd.Source_Span, cols: int, widths: []int, extra:
^Table_Extra)
```

table_decorate gives an open Table, before its rows, its widths and a table directive's attributes and title. A host calls it for the table it emits inside a table directive (Parser.table_extra).

```
rst.take :: proc(p: ^Parser, $T: typeid, n: int) -> ([]T,
bool)
```

take returns n zeroed items of T from scratch, valid until the enclosing mark is released. When scratch is short it records the capacity need on the builder (poisoning it) and returns false.

```
rst.trim_blank_end :: proc(lines: []Line) -> []Line
```

trim_blank_end returns lines without its trailing blank lines (a subslice).

Constants

```
rst.CONFORMANCE :: gd.Conformance.Partial
```

```
rst.DIRECTIVE_UNKNOWN
```

DIRECTIVE_UNKNOWN is what the reader reports for a directive no handler knows; a handler that finds it cannot serve a directive it was given reports it too.

```
rst.VERSION :: "0.1.0"
```


readers/rst/inline

Package `rst_inline` parses reStructuredText inline markup and resolves references. The block reader in `lib/readers/rst` owns lines, directives, and document structure; it calls `parse_inline` for every inline context and resolve once parsing ends.

Types

```
inline.Inline_State :: struct
```

`Inline_State` is shared, long-lived reader state for inline parsing.

```
ctx: ^gd.Read_Context
```

```
source: gd.Source_Id
```

the source the current lines come from.

```
roles: []Role
```

custom roles; storage owned by the block reader.

```
roles_used: int
```

```
default_role: string
```

「」 means title-reference.

```
hook: Role_Hook
```

a dialect's roles, consulted first; see `extension.odin`.

```
options: Options
```

settings of the block reader; see `options.odin`.

```
inline.Line :: gd.Line_View
```

`Line` is a view of one line of text. `offset` is the source byte offset of `text[o]`; for a line copied during tab expansion it is the offset of the original line start.

```
inline.Options :: struct
```

`Options` are the settings of the block reader (`rst.Settings`) that inline parsing and resolution need.

```
pep_references: bool
```

rfc_references: bool

pep_base_url: string

rfc_base_url: string

trim_footnote_reference_space: bool

smart_quotes: Smart_Quotes

smart_quotes_locales: string

language: string

docinfo: bool

sectsubtitle: bool

character_level: bool

tab_width: int

tab stops of the main source; 0 means 8.

default_substitution: Substitution_Hook

inline.Role :: struct

Role is a role defined by the role directive, or a standard role.

name: string

normalized (lowercase) role name.

base: Role_Base

classes: string

space-separated, from :class:.

format: string

raw roles: the output format, e.g. 「html」 .

language: string

code-derived roles.

`inline.Role_Base :: enum u8`

Emphasis

Strong

Literal

Code

Math

Subscript

Superscript

Title_Reference

Pep_Reference

Rfc_Reference

Abbreviation

Acronym

Raw

Custom

a Role node carrying the role name and classes.

`inline.Role_Call :: struct`

Role_Call describes one interpreted text for an extension’s role handler. The strings live in scratch and are valid only during the call.

state: ^Inline_State

b: `^gd.Builder`

name: `string`

the role name as written, or the default role's name.

marked: `string`

the content with every escaping backslash as NUL; breaks as 「\n」 .

text: `string`

the content with escapes processed; line breaks become spaces.

span: `gd.Source_Span`

the whole interpreted text, role included.

content: `gd.Source_Span`

the content between the backquotes.

inline.Role_Hook *:: struct*

Role_Hook lets a dialect handle interpreted text before the document's and the standard roles are consulted. The procedure returns false for a role it does not know, and the standard lookup continues.

procedure: `proc(user: rawptr, call: ^Role_Call) -> bool`

user: `rawptr`

inline.Smart_Quotes *:: enum u8*

Smart_Quotes selects typographic quotation marks, apostrophes, dashes, and ellipses.

Off

On

the language's primary quotation marks outside, secondary inside.

Alt

the language's alternative marks, where it has them.

```
inline.Substitution_Hook :: struct
```

Substitution_Hook supplies substitutions a dialect defines by default, such as Sphinx’s |version|, |release|, and |today|. Resolution offers it every reference whose name no definition in the document has exactly (the case-insensitive match comes after it); a procedure returning true replaces the reference with that text.

```
procedure: proc(user: rawptr, name: string) -> (text:
string, ok: bool)
```

```
user: rawptr
```

Procedures

```
inline.apply_host_role :: proc(s: ^Inline_State, name:
string, lines: []Line, q, j: int) -> gd.Status
```

apply_host_role runs the role called name on interpreted text written in another markup, such as MyST’s {name}`content`, as the dialect’s hook and the standard roles would for reStructuredText. lines hold the whole construct; the content is the joined text’s range [q, j) (lines joined by ‘\n’) and is taken literally: a backslash is an ordinary character. An unknown role is reported, and the construct stays as text.

```
inline.footnote_symbol :: proc(n: int, out: ^gd.Buffer)
```

footnote_symbol writes the label of symbol footnote n (1-based) into out; nothing for n < 1. A label that does not fit sets out’s overflow, as every Buffer write does.

```
inline.parse_inline :: proc(s: ^Inline_State, lines: []Line)
-> gd.Status
```

parse_inline emits inline nodes for the lines as children of the builder’s currently open node. Markup may span lines; a line break outside inline literals and roles becomes a Soft_Break node, as in CommonMark. Markup that cannot be completed is kept as text with a warning; unknown or misused roles are errors. It borrows s and lines for the call; its working memory is scratch it releases. Problems go to the reports; the result is the builder’s status, which a full table or spent work budget sets.

```
inline.python_space :: proc(r: rune) -> bool
```

python_space reports whether r is white space as Python’s str.isspace and the \s of Python’s regular expressions define it, for readers that follow Python patterns.

```
inline.resolve :: proc(ctx: ^gd.Read_Context, options :=
Options{docinfo = true}) -> gd.Status
```

resolve links references to targets, numbers footnotes, checks substitutions, and promotes the document title when ctx.options.promote_title is set. It edits the builder's nodes in place, with scratch it releases; problems go to ctx.reports; the result is the builder's status.

```
inline.unescape :: proc(out: ^gd.Buffer, marked: string)
```

unescape writes marked text (escapes as NUL, see Role_Call.marked) with escapes processed: an escaped space or line break disappears, any other escaped character stays, and an unescaped line break becomes a space.

readers/sphinx

Package sphinx reads reStructuredText with the Sphinx layer: Sphinx's directives, roles, and domains (GDS 「Guidedog: everything Sphinx does, the Odin way」), implemented independently on the base reader's extension interface. It produces the shapes of lib/core/AST.md, 「Sphinx layer」; cross-document work (toctrees, cross-references, numbering) is left to the project host. The same handlers serve Markdown (MyST) read by the commonmark reader: see markdown.odin.

Types

```
sphinx.Autodoc_Message :: struct
```

Autodoc_Message is a problem the host found, reported at the directive.

```
severity: gd.Severity
```

```
code: string
```

```
title: string
```

```
message: string
```

```
hint: string
```

```
sphinx.Autodoc_Option :: struct
```

```
name: string
```

```
value: string
```

「」 for a flag.

```
sphinx.Autodoc_Provider :: struct
```

Autodoc_Provider belongs to the caller's trust boundary and may allocate.

```
procedure: proc(user: rawptr, request: ^Autodoc_Request) -
> Autodoc_Result
```

```
user: rawptr
```

```
sphinx.Autodoc_Request :: struct
```

Autodoc_Request is one autodoc directive and the context it needs: the current py:module and py:class (ref_context) and the document.

```
directive: string
```

```
「autoclass」
```

```
argument: string
```

```
options: []Autodoc_Option
```

```
content: string
```

the directive's content, lines joined with 「\n」 .

```
module: string
```

```
class: string
```

```
docname: string
```

```
sphinx.Autodoc_Result :: struct
```

Autodoc_Result borrows host memory valid for the whole conversion. name names the generated text as a source (for diagnostics), such as 「<autodoc flask.Flask>」 .

```
name: string
```

```
text: string
```

```
messages: []Autodoc_Message
```

```
sphinx.Config :: struct
```

Config holds the `conf.toml` settings that reading needs, with Sphinx's names. It reaches the reader through `gd.Read_Options.extension`. Start from `default_config`: several defaults are true. Settings that the base reader owns (smart quotes, PEP and RFC base URLs, trimmed footnote reference space, language) live in settings.

```
using settings: rst.Settings
```

```
docname: string
```

the document's name, e.g. `「usage/intro」` .

```
default_role: string
```

`「」` means Docutils' title-reference.

```
primary_domain: string
```

`「py」` ; `「」` means none.

```
highlight_language: string
```

`「default」` .

```
extlinks: []Extlink
```

```
rst_prolog: string
```

parsed before every document.

```
rst_epilog: string
```

parsed after every document.

```
numfig: bool
```

```
math_number_all: bool
```

```
todo_include_todos: bool
```

```
show_authors: bool
```


add_function_parentheses: bool

true.

add_module_names: bool

true.

toc_object_entries: bool

true: objects have table of contents entries.

toc_object_entries_show_parents: Toc_Parents

manpages_url: string

「{page}」, 「{section}」, 「{path}」 substituted.

option_emphasise_placeholders: bool

trim_doctest_flags: bool

true: doctest flags and <BLANKLINE> are hidden.

version: string

The default substitutions |version|, |release|, and |today|, for documents that do not define them: the short version, the full release, and today's date already formatted (Sphinx's today, or today_fmt applied).

release: string

today: string

intersphinx: bool

sphinx.ext.intersphinx: the :external: roles are read when it is on, and an external role naming intersphinx_resolve_self is an ordinary reference.

intersphinx_resolve_self: string

labels: ^gd.Labels

The group names of info fields (「Parameters」, 「Returns」) in the document's language, as a host translates its labels; nil for English.

object_types: `[]Object_Type`

Directives and roles the project declares (see `object_types.odin`), and other names for directives.

directive_aliases: `[]Directive_Alias`

autodoc: `Autodoc_Provider`

`sphinx.ext.autodoc`: the host that generates what autodoc directives stand for; `nil` when the extension is off. See `autodoc.odin`.

odin_autodoc: `Autodoc_Provider`

The `odin` domain's generating directives (`odin:autopackage`, ...): the host that reads the `Odin` source; `nil` leaves them unknown.

`sphinx.Directive_Alias :: struct`

name: `string`

「`django-admin-option`」 names 「`option`」 ; a target may name

target: `string`

「`django-admin-option`」 names 「`option`」 ; a target may name

`sphinx.Extlink :: struct`

`Extlink` is one `sphinx.ext.extlinks` entry: `:name:part` links to `url` with `%s` replaced by `part`; `caption`, when set, is the link text with `%s` replaced likewise.

name: `string`

url: `string`

caption: `string`

`sphinx.Markdown :: struct`

`Markdown` serves the `commonmark` reader with this layer's directives and roles, the same handlers that serve `reStructuredText`: the reader parses `MyST Markdown` and hands each directive and role here (`commonmark.Extension`); they run on an embedded `reStructuredText` parser whose host is the `Markdown` reader (`rst.Host`), so their nested content is parsed as `Markdown` again. `eval-rst` parses its content as `reStructuredText`. The document is

resolved as reStructuredText is (references, footnotes, targets, bibliographic fields, smart quotes).

Use: `markdown_settings(&m, config, myst)` gives the reader settings to pass as `gd.Read_Options.extension` to the 「commonmark」 reader; `m` must outlive the read.

config: Config

settings: cm.Settings

ext: cm.Extension

state: State

parser: rst.Parser

host: rst.Host

reader: ^cm.Host

sphinx.Object_Type :: struct

directive: string

「setting」

role: string

「setting」 ; 「」 for none.

index: string

Sphinx's indextemplate, 「pair: %s; setting」 ; 「」 for none.

target: bool

`add_crossref_type`: a target and an index entry, no description.

first_word: bool

the object's name is the signature up to its first space.

display: string

how the signature is shown, 「%s」 for it; 「」 shows it as written.

program: bool

the name becomes the program options that follow belong to.

sphinx.Toc_Parents :: enum u8

Toc_Parents is toc_object_entries_show_parents: how much of an object's hierarchy its table of contents entry shows.

Domain

as the domain names it: Python's Class.method, C++'s enclosing objects.

Hide

the object's own name.

All

every part: module.Class.method, ns::Class::f.

Procedures

sphinx.default_config :: proc() -> Config

default_config returns Sphinx's defaults, with the base reader's default_settings and smart quotes on. The value owns nothing; fields a caller sets must outlive the read.

sphinx.markdown_settings :: proc(m: ^Markdown, config: Config, myst: cm.Settings) -> ^cm.Settings

markdown_settings prepares m for one read of a Markdown document with the Sphinx configuration config and the MyST settings myst, and returns the reader settings. It copies both into m; the result points into m, so m must outlive the read and serves one read at a time. It cannot fail.

sphinx.read :: proc(ctx: ^gd.Read_Context) -> gd.Status

read reads reStructuredText with the Sphinx layer. ctx.options.extension is a ^Config (gd.read_extension), or empty for default_config; another type is refused with Invalid_Input. The read copies the Config and keeps nothing of it afterwards; its tables live in scratch, released before it returns. Otherwise it fails as rst.read.

sphinx.read_inline :: proc(ctx: ^gd.Read_Context, span: gd.Source_Span) -> gd.Status

read_inline parses a span of ctx.source as reStructuredText inline content with Sphinx's roles, for a translated message (gd.Read_Inline_Proc). rst_prolog

and `rst_epilog` do not apply to a message. Settings, ownership, and failures are read's.

```
sphinx.reader :: proc() -> gd.Reader
```

`reader` returns the 「sphinx」 reader descriptor, for a `gd.Registry`. It is a value with no state: registering it twice or copying it is harmless.

Constants

```
sphinx.CONFORMANCE :: gd.Conformance.Partial
```

```
sphinx.MAX_OBJECT_TYPES :: 64
```

```
sphinx.VERSION :: "0.1.0"
```

render/html

Package `html` renders a Guidedoc document as HTML without calling an allocator. The default output is semantic HTML with anchors, notes, and tables; the reference profile reproduces the CommonMark reference renderer byte for byte for conformance tests. Raw content and unsafe URLs are governed by the conversion policy.

Procedures

```
html.render :: proc(ctx: ^gd.Render_Context) -> gd.Status
```

`render` writes `ctx.view` as HTML through `ctx.out`: a whole page when `ctx.options.standalone`, else a fragment, and the CommonMark reference renderer's bytes when `ctx.options.reference_profile`. It borrows the context for the call and keeps nothing; its scratch is released before it returns. Problems go to `ctx.reports`; the result is `.Ok`, the status of the first error (a raw block or URL the policy refuses, an unresolved reference, nesting past the limits), or `.Capacity` when scratch runs out.

```
html.renderer :: proc() -> gd.Renderer
```

`renderer` returns the 「html」 renderer descriptor, for a `gd.Registry`. It is a value with no state: registering it twice or copying it is harmless.

Constants

```
html.CONFORMANCE :: gd.Conformance.Partial
```

```
html.VERSION :: "0.1.0"
```

render/text

Package `text_render` renders a Guidedoc document as plain text without calling an allocator, following the conventions of Sphinx's text builder: paragraphs wrapped at 70 columns, titles underlined with `= - ~` and the other section characters, `「*」` and `「1.」` list markers with hanging indents, definitions and literal blocks indented by three spaces, grid tables drawn with `+--+` borders, and `「Note:」` admonitions.

Inline content is laid out twice, once to measure and once into scratch, and then wrapped; table cells are laid out into scratch at trial widths before the table is drawn. Output goes only through the emitter, so the engine's measure and write runs produce identical bytes.

Procedures

```
text.render :: proc(ctx: ^gd.Render_Context) -> gd.Status
```

`render` writes `ctx.view` as plain text through `ctx.out`. It borrows the context for the call and keeps nothing; its scratch is released before it returns. Problems go to `ctx.reports`; the result is `.Ok`, the status of the first error (raw content the policy refuses, an unresolved reference, nesting past the limits), or `.Capacity` when scratch cannot hold the indentation frames or a laid-out paragraph.

```
text.renderer :: proc() -> gd.Renderer
```

`renderer` returns the `「text」` `renderer` descriptor, for a `gd.Registry`. It is a value with no state: registering it twice or copying it is harmless.

Constants

```
text.CONFORMANCE :: gd.Conformance.Partial
```

```
text.VERSION :: "0.1.0"
```

render/typst

Package `typst_render` encodes a validated Guidedoc document as one Typst source file.

The output is a fixed, trusted preamble of `gd-` helper definitions followed by the body. Source text never becomes executable Typst: markup text escapes every ASCII punctuation character, and code, URLs, labels, and paths are string literals. Working memory comes only from the render context's scratch; output goes only through the emitter, so the Measure and Write runs produce identical bytes.

Procedures

```
typst.emit_preamble :: proc(e: ^gd.Emitter)
```

emit_preamble writes the helper definitions every body relies on. A host composing a book from body-only chapters writes it once, before the chapters. The text is a constant: it cannot fail, and the emitter counts what does not fit.

```
typst.render :: proc(ctx: ^gd.Render_Context) -> gd.Status
```

render writes ctx.view as Typst source through ctx.out: the preamble, page settings, and body when ctx.options.standalone, else the body alone (see emit_preamble). It borrows the context for the call and keeps nothing but scratch taken from ctx, which the caller's scratch mark releases. Problems go to ctx.reports; the result is .Ok, the status of the first error (raw content the policy refuses, nesting past the limits), or .Capacity when scratch cannot hold the per-node marks.

```
typst.renderer :: proc() -> gd.Renderer
```

renderer returns the 「typst」 renderer descriptor, for a gd.Registry. It is a value with no state: registering it twice or copying it is harmless.

Constants

```
typst.CONFORMANCE :: gd.Conformance.Partial
```

```
typst.VERSION :: "0.1.1"
```

treesitter

Package treesitter is a thin binding of the tree-sitter C library and its Python grammar: parsers, trees, nodes, and cursors, and nothing more. It imports only Odin's own packages, so any Odin program can use it; Guidedog's autodoc reads Python source through it.

Types

```
treesitter.Cursor :: struct
```

Cursor is TSTreeCursor; cursor_delete frees it.

```
tree: rawptr
```

```
id: rawptr
```

```
ctx: [3]u32
```

```
tree sitter.Field_Id :: u16
```

```
tree sitter.Iterator :: struct
```

Iterator visits a node's children in order, named or all, for for child in

```
node: Node
```

```
index: int
```

```
count: int
```

```
named: bool
```

```
tree sitter.Language :: struct
```

```
tree sitter.Node :: struct
```

Node is TSNODE, a value that is valid while its tree lives.

```
ctx: [4]u32
```

```
id: rawptr
```

```
tree: ^Tree
```

```
tree sitter.Parser :: struct
```

Opaque handles.

```
tree sitter.Point :: struct
```

```
row: u32
```

```
column: u32
```

in bytes

```
tree sitter.Symbol :: u16
```

```
tree sitter.Tree :: struct
```


Procedures

```
tree-sitter.abi_version :: proc(l: ^Language) -> int
```

abi_version is the grammar's ABI, which the runtime must support.

```
tree-sitter.child :: proc(n: Node, i: int) -> Node
```

child is the node's child i of child_count, or a null node.

```
tree-sitter.child_count :: proc(n: Node) -> int
```

child_count counts the node's children, named and anonymous.

```
tree-sitter.children :: proc(n: Node, named := true) ->
Iterator
```

children iterates n's children, named ones only unless named is false, with next_child.

```
tree-sitter.cursor_delete :: proc(c: ^Cursor)
```

cursor_delete frees what cursor_new allocated.

```
tree-sitter.cursor_field_name :: proc(c: ^Cursor) -> string
```

cursor_field_name names the field the cursor's node fills in its parent, or 「」.

```
tree-sitter.cursor_first_child :: proc(c: ^Cursor) -> bool
```

cursor_first_child moves to the first child; false, and no move, when there is none.

```
tree-sitter.cursor_new :: proc(n: Node) -> Cursor
```

cursor_new starts a cursor at n, for walks that allocate nothing per step. tree-sitter allocates the cursor's stack; the caller frees it with cursor_delete.

```
tree-sitter.cursor_next_sibling :: proc(c: ^Cursor) -> bool
```

cursor_next_sibling moves to the next sibling; false, and no move, when there is none.

```
tree-sitter.cursor_node :: proc(c: ^Cursor) -> Node
```

cursor_node is the node the cursor is at.

```
treesitter.cursor_parent :: proc(c: ^Cursor) -> bool
```

cursor_parent moves to the parent; false, and no move, at the node the cursor started at.

```
treesitter.deeper_than :: proc(n: Node, limit: int) -> bool
```

deeper_than reports whether the tree below n nests more than limit levels (n's children are level 1). Parsing needs no recursion however deep the source nests, but recursive walks over the tree do: a program checks first, and refuses trees deeper than its walks can hold. It walks with a cursor, so it needs no recursion itself.

```
treesitter.end_byte :: proc(n: Node) -> int
```

end_byte is the byte offset just past the node in the parsed text.

```
treesitter.end_point :: proc(n: Node) -> Point
```

end_point is the o-based row and byte column where the node ends; see start_point.

```
treesitter.equal :: proc(a, b: Node) -> bool
```

equal reports whether a and b are the same node of the same tree.

```
treesitter.field :: proc(n: Node, name: string) -> Node
```

field is the child the grammar names, such as 「name」 or 「body」 ; a null node when absent.

```
treesitter.field_name_for_child :: proc(n: Node, i: int) -> string
```

field_name_for_child names the field child i fills, or 「」 .

```
treesitter.has_error :: proc(n: Node) -> bool
```

has_error reports whether the node or any node below it is an error or missing node.

```
treesitter.is_error :: proc(n: Node) -> bool
```

is_error reports a node the parser could not fit into the grammar.

```
treesitter.is_extra :: proc(n: Node) -> bool
```

is_extra: comments and other tokens that may appear anywhere.

```
tree sitter.is_missing :: proc(n: Node) -> bool
```

is_missing reports a node the parser inserted to recover from a syntax error.

```
tree sitter.is_named :: proc(n: Node) -> bool
```

is_named reports a node the grammar names, rather than an anonymous token such as 「(」.

```
tree sitter.is_null :: proc(n: Node) -> bool
```

is_null reports a null node: an absent field, child, sibling, or parent.

```
tree sitter.kind :: proc(n: Node) -> string
```

kind is the node's type name, such as 「function_definition」, or the token itself for anonymous nodes such as 「(」 or 「def」.

```
tree sitter.named_child :: proc(n: Node, i: int) -> Node
```

named_child is the node's named child i of named_child_count, or a null node.

```
tree sitter.named_child_count :: proc(n: Node) -> int
```

named_child_count counts the node's named children.

```
tree sitter.next_child :: proc(it: ^Iterator) -> (n: Node, i: int, ok: bool)
```

next_child returns the next child and its index, for
for child, i in next_child(&it).

```
tree sitter.next_named_sibling :: proc(n: Node) -> Node
```

next_named_sibling is the next named node under n's parent, or a null node.

```
tree sitter.next_sibling :: proc(n: Node) -> Node
```

next_sibling is the node after n under its parent, or a null node.

```
tree sitter.parent :: proc(n: Node) -> Node
```

parent is the node's parent, or a null node for the root.

```
tree sitter.parse :: proc(p: ^Parser, text: string) -> ^Tree
```

parse parses UTF-8 text into a tree, which tree-sitter allocates and the caller frees with tree_delete. The tree refers to text by byte offsets only, so text need not outlive it, but text slices whatever text it is given. Nil when parsing was

impossible (text over 4 GiB, or no language set); syntax errors give a tree with error nodes instead (`has_error`).

```
tree sitter.parser_delete :: proc(p: ^Parser)
```

`parser_delete` frees a parser from `parser_new`; nil is ignored. Trees it parsed stay valid.

```
tree sitter.parser_new :: proc(language: ^Language) -> ^Parser
```

`parser_new` returns a parser for `language`, which tree-sitter allocates and the caller frees with `parser_delete`, or nil when this runtime cannot read the grammar (a shared runtime older than the grammar's ABI).

```
tree sitter.prev_named_sibling :: proc(n: Node) -> Node
```

`prev_named_sibling` is the previous named node under `n`'s parent, or a null node.

```
tree sitter.prev_sibling :: proc(n: Node) -> Node
```

`prev_sibling` is the node before `n` under its parent, or a null node.

```
tree sitter.python :: proc() -> ^Language
```

`python` returns the Python grammar, static data that is never freed.

```
tree sitter.root :: proc(t: ^Tree) -> Node
```

`root` is the tree's root node, valid while the tree lives.

```
tree sitter.start_byte :: proc(n: Node) -> int
```

`start_byte` is the byte offset where the node starts in the parsed text.

```
tree sitter.start_point :: proc(n: Node) -> Point
```

`start_point` and `end_point` are 0-based rows and byte columns.

```
tree sitter.symbol :: proc(n: Node) -> Symbol
```

`symbol` is the grammar's number for the node's type.

```
tree sitter.text :: proc(n: Node, source: string) -> string
```

`text` is the source the node spans, a slice of the parsed text.

```
tree_sitter.tree_delete :: proc(t: ^Tree)
```

`tree_delete` frees a tree from parse, and with it every node of the tree; nil is ignored.

Constants

```
tree_sitter.NOTICES :: #load("licenses/tree-sitter.txt",  
string) + "\n" + #load("licenses/tree-sitter-python.txt",  
string)
```

The MIT licence notices of the two libraries, which a program that links them ships.

```
tree_sitter.PYTHON_VERSION :: "0.25.0"
```

```
tree_sitter.RUNTIME_VERSION :: "0.26.13"
```

The versions native/tree-sitter/build.sh pins; a shared build links whatever is installed.

```
tree_sitter.STATIC :: #config(TREESITTER_STATIC, BUILT)
```

Foreign imports

```
@(extra_linker_flags = #config(TREESITTER_LINK_FLAGS, ""))  
foreign import tree_sitter.ts {"../..../native/tree-sitter/out/  
lib/tree-sitter-python.lib", "../..../native/tree-sitter/out/  
lib/tree-sitter.lib"}
```

```
@(extra_linker_flags = #config(TREESITTER_LINK_FLAGS, ""))  
foreign import tree_sitter.ts {"../..../native/tree-sitter/out/  
lib/libtree-sitter-python.a", "../..../native/tree-sitter/out/  
lib/libtree-sitter.a"}
```

```
@(extra_linker_flags = #config(TREESITTER_LINK_FLAGS, ""))  
foreign import tree_sitter.ts {"system:tree-sitter-python",  
"system:tree-sitter"}
```

typst

Package `typst` adapts the Typst toolchain for the host. The development backend runs an installed, pinned `typst` executable with an argument vector, never a shell; the embedded backend links the compiler in (bridge.odin). It is outside Guidedoc's allocation-free core.

Typst is not a sandbox of its own; what it may do is what the caller gives it:

- Files: Typst reads only paths below the root it is given, and refuses a path that leaves it; a link below the root is followed. Fonts come from `font_paths` and, unless `embedded_fonts_only` is set, from the system.
- Packages: an `@preview` import downloads the package, unless `offline` is set (the embedded backend only; the typst program has no such switch).
- Memory and time: Typst's own, in the process (embedded) or in the typst program; no budget of the host bounds them. Only the output the embedded backend copies back is counted, through `reserve`.

So Typst code reaches everything below its root: a caller that compiles Typst it does not trust gives a root holding only what that code may read. Guidedog's builds do (project/typst_root.odin), and an `-untrusted` build compiles only Typst it generated.

Types

`typst.Backend` *:: struct*

Backend is the compiler a compile runs: an installed typst program, or the embedded one. `locate` makes it; its strings are the caller's.

program: string

version: string

embedded: bool

the statically linked compiler; no external program runs.

`typst.Diagnostic` *:: struct*

Diagnostic is one problem Typst reported.

file: string

line: int

1-based; 0 when the compiler gave no position.

column: int

severity: string

message: string

stdin: bool

the position is in the generated source fed on `stdin`.

```
typst.Options :: struct
```

Options mirror the typst CLI. Zero values mean: download missing packages into the system cache, and use system fonts before the embedded ones.

```
offline: bool
```

use only packages already on disk.

```
package_cache: string
```

package cache directory; 「」 is the system cache.

```
embedded_fonts_only: bool
```

ignore system fonts, for reproducible output.

```
font_paths: []string
```

extra font directories, searched first.

```
reserve: Reserve
```

When set, the embedded compiler's output is counted before it is copied into host memory; a refusal fails the compile. The typst program writes its output itself.

```
typst.Outcome :: struct
```

Outcome is what a compile did. Its diagnostics and text are allocated in context allocator and owned by the caller; launch_error is set when Typst could not run.

```
ok: bool
```

```
exit_code: int
```

```
diagnostics: []Diagnostic
```

```
detail: string
```

unparsed compiler output, bounded.

```
launch_error: string
```

```
typst.Reserve :: struct
```

Reserve asks the caller whether bytes may be allocated, before the embedded compiler's output is copied into host memory; false refuses the compile.

```
procedure: proc(user: rawptr, bytes: int) -> bool
```

```
user: rawptr
```

```
typst.Target :: enum u8
```

Target is the artifact a compile writes.

```
Pdf
```

```
Html
```

Procedures

```
typst.compile_file :: proc(b: Backend, source, root, output: string, target: Target, options := Options{}) -> Outcome
```

compile_file compiles a native Typst source file into output; paths resolve against root, and Typst reads nothing outside it. The Outcome is allocated in context.allocator and owned by the caller. Write to a staged path and rename it on success when a failed compile must not disturb an earlier artifact.

```
typst.compile_text :: proc(b: Backend, text, root, output: string, target: Target, options := Options{}) -> Outcome
```

compile_text compiles generated Typst fed on stdin, so no file is written beside the user's sources. Relative paths in the text resolve against root. The Outcome is allocated in context.allocator and owned by the caller.

```
typst.find_program :: proc(name: string) -> (string, bool)
```

find_program looks a program up on PATH, handling Windows' 「;」 and 「.exe」. The path is allocated in context.allocator and owned by the caller; false when none is found.

```
typst.locate :: proc(program: string) -> (Backend, bool)
```

locate returns the embedded compiler when this build links one and the caller did not name another program; otherwise it finds the program and asks for its version. The backend's strings are allocated in context.allocator and owned by the caller; false when no program is found. A program that cannot tell its version has version 「」.


```
typst.offset_of :: proc(text: string, line, column: int) -> int
```

offset_of converts a 1-based line and column (in characters) into a byte offset of text, clamped to its end; it allocates nothing.

```
typst.pinned :: proc(b: Backend) -> bool
```

pinned reports whether the backend is the Typst release Guidedog is tested with.

Constants

```
typst.EMBEDDED :: #config(GUIDEDOG_TYPST_BRIDGE, false)
```

EMBEDDED selects the statically linked Typst compiler. Build the bridge first:

```
cargo build --release --offline --manifest-path native/typst_bridge/
Cargo.toml
odin build cmd/guidedog -define:GUIDEDOG_TYPST_BRIDGE=true -out:build/
guidedog
```

Without it, Guidedog uses the development backend: an installed typst program.

```
typst.PINNED_VERSION :: "0.15.1"
```

PINNED_VERSION is the Typst release Guidedog is built and tested with.

Foreign imports

```
foreign import typst.bridge {"../../native/typst_bridge/target/release/guidedog_typst_bridge.lib",
"system:ws2_32.lib", "system:userenv.lib",
"system:ntdll.lib", "system:bcrypt.lib",
"system:advapi32.lib", "system:kernel32.lib"}
```

対応環境: Windows.

```
foreign import typst.bridge {"../../native/typst_bridge/target/release/libguidedog_typst_bridge.a", "system:Security.framework",
"system:CoreFoundation.framework",
"system:System", "system:iconv"}
```

対応環境: Darwin.

```
foreign import typst.bridge {"../../native/typst_bridge/target/release/libguidedog_typst_bridge.a", "system:gcc_s",
"system:util", "system:rt", "system:pthread", "system:m",
"system:dl", "system:c"}
```

対応環境: Linux, FreeBSD, OpenBSD, NetBSD, WASI, JS, Orca, Freestanding.

索引

A

- abi_version (procedure in treesitter), 341
- absolute (procedure in project), 277
- add (procedure in i18n), 228
 - (procedure in inventory), 233
- add_attribute (procedure in core), 123
- add_attribute_string (procedure in core), 123
- add_catalog (procedure in gettext), 160
- add_common (procedure in readers/rst), 318
- add_filter (procedure in jinja), 242
- add_flags (procedure in core), 124
- add_function (procedure in jinja), 242
- add_global (procedure in jinja), 242
- add_leaf (procedure in core), 124
- add_message (procedure in gettext), 160
- add_source (procedure in core), 124
- add_template (procedure in jinja), 242
- add_test (procedure in jinja), 243
- add_text (procedure in core), 124
- Admonition_Kind (enum in core), 88
- Admonition_Kind.Attention (enumerator in core), 88
- Admonition_Kind.Caution (enumerator in core), 88
- Admonition_Kind.Danger (enumerator in core), 88
- Admonition_Kind.Error (enumerator in core), 88
- Admonition_Kind.Generic (enumerator in core), 88
- Admonition_Kind.Hint (enumerator in core), 88
- Admonition_Kind.Important (enumerator in core), 88
- Admonition_Kind.Note (enumerator in core), 88
- Admonition_Kind.Seealso (enumerator in core), 88
- Admonition_Kind.Tip (enumerator in core), 88
- Admonition_Kind.TODO (enumerator in core), 88
- Admonition_Kind.Warning (enumerator in core), 88
- admonition_label (procedure in core), 124
- admonition_name (procedure in core), 124
- Align (enum in core), 88
- Align.Center (enumerator in core), 88
- Align.Left (enumerator in core), 88
- Align.None (enumerator in core), 88
- Align.Right (enumerator in core), 88
- All (struct in pyscan), 280
- All.defined (field in pyscan), 280
- All.known (field in pyscan), 280
- All.names (field in pyscan), 280
- Analysis (struct in pyscan), 281
- Analysis.annotation_order (field in pyscan), 281
- Analysis.annotations (field in pyscan), 281
- Analysis.attr_docs (field in pyscan), 281
- Analysis.attr_order (field in pyscan), 281
- Analysis.counter (field in pyscan), 281
- Analysis.finals (field in pyscan), 281
- Analysis.overloads (field in pyscan), 281
- Analysis.tagorder (field in pyscan), 281
- Anchor_Extra (struct in core), 89
- Anchor_Extra.item (field in core), 89
- Anchor_Extra.note (field in core), 89
- Anchor_Extra.number (field in core), 89
- Anchor_Extra.owner (field in core), 89
- Anchor_Extra.span (field in core), 89
- anchor_extras (procedure in core), 124
- anchor_ids (procedure in core), 124
- Anchor_Ids (struct in core), 89
- Anchor_Ids.span (field in core), 89
- anchor_of (procedure in core), 124
- anchors (struct in core), 89
- anchors.bytes (field in core), 90
- anchors.count (field in core), 90
- anchors.counter (field in core), 90
- anchors.extra_count (field in core), 90
- anchors.extras (field in core), 90
- anchors.index (field in core), 90
- anchors.labels (field in core), 90
- anchors.names (field in core), 90
- anchors.next (field in core), 90
- anchors.own (field in core), 90
- anchors.prefix (field in core), 90
- anchors.prefix_length (field in core), 90
- anchors.spans (field in core), 90
- append_value (procedure in jinja), 243
- apply (procedure in gds), 148
- apply_host_role (procedure in readers/rst/inline), 329
- Arch (type in odindoc), 249
- Archs (type in odindoc), 249

- Arena (struct in host), 195
- Arena.backing (field in host), 195
- Arena.block (field in host), 195
- Arena.blocks (field in host), 196
- Arena.bytes (field in host), 196
- Arena.last (field in host), 196
- arena_allocator (procedure in host), 208
- ARENA_BLOCK_BYTES (constant in host), 221
- arena_destroy (procedure in host), 208
- arena_init (procedure in host), 208
- arg (procedure in jinja), 243
- arg_int (procedure in core), 125
 - (procedure in jinja), 243
- Arg_Kind (enum in core), 91
- Arg_Kind.Int (enumerator in core), 91
- Arg_Kind.None (enumerator in core), 91
- Arg_Kind.Span (enumerator in core), 91
- Arg_Kind.Static (enumerator in core), 91
- Arg_Rule (enum in readers/rst), 307
- Arg_Rule.None (enumerator in readers/rst), 307
- Arg_Rule.Optional (enumerator in readers/rst), 307
- Arg_Rule.Required (enumerator in readers/rst), 307
- arg_span (procedure in core), 125
- arg_static (procedure in core), 125
- Artifact_Request (struct in host), 196
- Artifact_Request.allocator (field in host), 197
- Artifact_Request.budget (field in host), 197
- Artifact_Request.emit_typst (field in host), 196
- Artifact_Request.force (field in host), 196
- Artifact_Request.from (field in host), 196
- Artifact_Request.input (field in host), 196
- Artifact_Request.limits (field in host), 196
- Artifact_Request.output (field in host), 196
- Artifact_Request.policy (field in host), 196
- Artifact_Request.read (field in host), 196
- Artifact_Request.render (field in host), 196
- Artifact_Request.roots (field in host), 197
- Artifact_Request.to (field in host), 196
- Artifact_Request.typst_options (field in host), 197
- Artifact_Request.typst_program (field in host), 196
- Artifact_Result (struct in host), 197
- Artifact_Result.exit (field in host), 197
- Artifact_Result.problem (field in host), 197
- Artifact_Result.reports (field in host), 197
- Artifact_Result.stdout (field in host), 197
- Artifact_Result.written (field in host), 197
- Attr_Key (struct in pyscan), 281
- Attr_Key.name (field in pyscan), 281
- Attr_Key.scope (field in pyscan), 281
- attribute (procedure in core), 125
- Attribute (struct in core), 91
 - (struct in graphdog), 167
- Attribute.key (field in core), 91
- Attribute.kind (field in graphdog), 167
- Attribute.name (field in core), 91
 - (field in graphdog), 167
- Attribute.value (field in core), 91
 - (field in graphdog), 168
- Attribute_Key (enum in core), 91
- Attribute_Key.Align (enumerator in core), 92
- Attribute_Key.Alt (enumerator in core), 91
- Attribute_Key.Class (enumerator in core), 91
- Attribute_Key.Custom (enumerator in core), 91
- Attribute_Key.Format (enumerator in core), 92
- Attribute_Key.Height (enumerator in core), 91
- Attribute_Key.Id (enumerator in core), 91
- Attribute_Key.Language (enumerator in core), 91
- Attribute_Key.Name (enumerator in core), 91
- Attribute_Key.Scale (enumerator in core), 91
- Attribute_Key.Source (enumerator in core), 92
- Attribute_Key.Start (enumerator in core), 92
- Attribute_Key.Target (enumerator in core), 92
- Attribute_Key.Title (enumerator in core), 92
- Attribute_Key.Width (enumerator in core), 91
- Attribute_Key.Widths (enumerator in core), 92
- Attribute_Kind (enum in graphdog), 168
- Attribute_Kind.Edge (enumerator in graphdog), 168
- Attribute_Kind.Graph (enumerator in graphdog), 168
- Attribute_Kind.Node (enumerator in graphdog), 168
- Attribute_Range (struct in core), 92
- Attribute_Range.count (field in core), 92
- Attribute_Range.first (field in core), 92
- attributes (procedure in core), 125
- Attributes (struct in core), 92
- Attributes.end (field in core), 92
- Attributes.next (field in core), 92
- autodoc (package), 82
- Autodoc_Message (struct in readers/sphinx), 330
- Autodoc_Message.code (field in readers/sphinx), 330
- Autodoc_Message.hint (field in readers/sphinx), 330
- Autodoc_Message.message (field in readers/sphinx), 330
- Autodoc_Message.severity (field in readers/sphinx), 330
- Autodoc_Message.title (field in readers/sphinx), 330
- Autodoc_Option (struct in readers/sphinx), 330
- Autodoc_Option.name (field in readers/sphinx), 330
- Autodoc_Option.value (field in readers/sphinx), 330

Autodoc_Provider (struct in readers/
sphinx), 331
Autodoc_Provider.procedure (field in
readers/sphinx), 331
Autodoc_Provider.user (field in readers/
sphinx), 331
Autodoc_Request (struct in readers/
sphinx), 331
Autodoc_Request.argument (field in
readers/sphinx), 331
Autodoc_Request.class (field in readers/
sphinx), 331
Autodoc_Request.content (field in readers/
sphinx), 331
Autodoc_Request.directive (field in
readers/sphinx), 331
Autodoc_Request.docname (field in
readers/sphinx), 331
Autodoc_Request.module (field in readers/
sphinx), 331
Autodoc_Request.options (field in readers/
sphinx), 331
Autodoc_Result (struct in readers/sphinx),
331
Autodoc_Result.messages (field in readers/
sphinx), 331
Autodoc_Result.name (field in readers/
sphinx), 331
Autodoc_Result.text (field in readers/
sphinx), 331
autoescaping (procedure in jinja), 243
available_memory (procedure in host), 209
AVAILABLE_OVERRIDE (constant in
host), 221

B

Backend (struct in typst), 346
Backend.embedded (field in typst), 346
Backend.program (field in typst), 346
Backend.version (field in typst), 346
backend_problem (procedure in host), 209
base_objects (procedure in pyscan), 292
begin_node (procedure in core), 125
bind_text_domain (procedure in gettext),
160
blank (procedure in readers/rst), 318
boolean (procedure in jinja), 243
Bound_Catalog (struct in gettext), 151
Bound_Catalog.catalog (field in gettext),
151
Bound_Catalog.domain (field in gettext),
151
bridge (foreign import in typst), 349
Budget (struct in host), 197
Budget.active (field in host), 198
Budget.freed (field in host), 198
Budget.limit (field in host), 198
Budget.mutex (field in host), 198
Budget.peak (field in host), 198
Budget.policy (field in host), 198
Budget.refused_need (field in host), 198
Budget.refused_used (field in host), 198
Budget.stopped (field in host), 198
Budget.used (field in host), 198

Budget.waiting (field in host), 198
budget_advice (procedure in host), 209
budget_hint (procedure in host), 209
budget_join (procedure in host), 209
budget_leave (procedure in host), 209
budget_limit (procedure in host), 209
Budget_Policy (struct in host), 198
Budget_Policy.disk (field in host), 199
Budget_Policy.locate (field in host), 199
Budget_Policy.locate_user (field in host),
199
Budget_Policy.on_refusal (field in host),
199
Budget_Policy.spill (field in host), 199
Budget_Policy.user (field in host), 199
budget_problem (procedure in host), 210
budget_report (procedure in host), 210
Buffer (struct in core), 92
Buffer.bytes (field in core), 92
Buffer.overflow (field in core), 92
Buffer.used (field in core), 92
Buffer.wanted (field in core), 92
buffer_string (procedure in core), 125
buffer_write (procedure in core), 125
buffer_write_byte (procedure in core), 125
build (procedure in gds), 148
(procedure in project), 277
build_anchors (procedure in core), 125
BUILD_BUDGET (constant in project), 280
build_index (procedure in gettext), 160
build_limits (procedure in project), 278
Build_Options (struct in project), 256
Build_Options.all (field in project), 256
Build_Options.arena_block (field in
project), 258
Build_Options.budget (field in project), 257
Build_Options.fresh (field in project), 256
Build_Options.heap (field in project), 257
Build_Options.jobs (field in project), 257
Build_Options.keep_going (field in
project), 257
Build_Options.max_depth (field in
project), 257
Build_Options.max_nodes (field in
project), 257
Build_Options.memory (field in project),
257
Build_Options.nitpicky (field in project),
257
Build_Options.now (field in project), 258
Build_Options.output_dir (field in project),
258
Build_Options.stack_bytes (field in
project), 257
Build_Options.tags (field in project), 257
Build_Options.untrusted (field in project),
257
Build_Options.warnings_as_errors (field in
project), 257
Build_Options.year (field in project), 258
Build_Result (struct in project), 258
Build_Result.budget (field in project), 259
Build_Result.exit (field in project), 258
Build_Result.memory (field in project), 259
Build_Result.output (field in project), 258
Build_Result.owner (field in project), 259
Build_Result.peak (field in project), 259

Build_Result.problem (field in project), 258
 Build_Result.published (field in project), 259
 Build_Result.reached (field in project), 259
 Build_Result.read (field in project), 258
 Build_Result.reports (field in project), 258
 Build_Result.reused (field in project), 258
 Build_Result.unchanged (field in project), 258
 Build_Result.unread (field in project), 258
 Build_Result.unsettled (field in project), 259
 Build_Result.written (field in project), 258
 Build_Stage (enum in project), 259
 Build_Stage.Preparing (enumerator in project), 259
 Build_Stage.Reading (enumerator in project), 259
 Build_Stage.Writing (enumerator in project), 259
 Build_Stage.Written (enumerator in project), 259
 Builder (struct in core), 93
 Builder.cancel (field in core), 93
 Builder.limits (field in core), 93
 Builder.need (field in core), 93
 Builder.open (field in core), 93
 Builder.reports (field in core), 93
 Builder.snap (field in core), 93
 Builder.status (field in core), 93
 Builder.work (field in core), 93
 builder_capacity (procedure in core), 126
 builder_fail (procedure in core), 126
 builder_finish (procedure in core), 126
 builder_init (procedure in core), 126
 Builder_Kind (enum in project), 260
 Builder_Kind.Dirhtml (enumerator in project), 260
 Builder_Kind.Dummy (enumerator in project), 260
 Builder_Kind.Gettext (enumerator in project), 260
 Builder_Kind.Html (enumerator in project), 260
 Builder_Kind.Pdf (enumerator in project), 260
 Builder_Kind.Singlehtml (enumerator in project), 260
 Builder_Kind.Text (enumerator in project), 260
 builder_named (procedure in project), 278
 BUILDER_NAMES (variable in project), 280
 BUILDER_OPEN (constant in core), 141
 bullet_width (procedure in readers/rst), 318

C

Call (struct in jinja), 234
 Call.args (field in jinja), 234
 Call.kwargs (field in jinja), 234
 Call.name (field in jinja), 234
 Call.self (field in jinja), 234
 Call.state (field in jinja), 234
 Call.user (field in jinja), 234

call_filter (procedure in jinja), 243
 call_test (procedure in jinja), 243
 Callable (struct in jinja), 234
 Callable.macro (field in jinja), 235
 Callable.name (field in jinja), 234
 Callable.procedure (field in jinja), 234
 Callable.self (field in jinja), 234
 Callable.user (field in jinja), 234
 cancel_probe (procedure in host), 210
 Cancel_Probe (struct in core), 93
 Cancel_Probe.procedure (field in core), 93
 Cancel_Probe.user (field in core), 93
 canonical_roots (procedure in host), 210
 Capacity_Need (struct in core), 93
 Capacity_Need.exact (field in core), 93
 Capacity_Need.minimum (field in core), 93
 Capacity_Need.region (field in core), 93
 Catalog (struct in gds), 145
 (struct in gettext), 151
 Catalog allocator (field in gettext), 152
 Catalog.allow_fuzzy (field in gettext), 152
 Catalog.arena (field in gettext), 152
 Catalog.has_header (field in gettext), 151
 Catalog.header (field in gettext), 151
 Catalog.index (field in gettext), 152
 Catalog.messages (field in gettext), 151
 Catalog.name (field in gettext), 151
 Catalog.owner (field in gettext), 152
 Catalog.plural (field in gettext), 152
 Catalog.records (field in gds), 145
 Catalog.reserved (field in gds), 145
 Catalog_Arena (struct in gettext), 152
 Catalog_Arena.refused (field in gettext), 152
 Catalog_Arena.using (field in gettext), 152
 catalog_bytes (procedure in gettext), 161
 catalog_refused (procedure in gettext), 161
 catch_interrupts (procedure in host), 210
 cell_info (procedure in core), 126
 Cell_Info (struct in core), 94
 Cell_Info.align (field in core), 94
 Cell_Info.col_span (field in core), 94
 Cell_Info.row_span (field in core), 94
 Change (struct in gds), 145
 Change.after (field in gds), 146
 Change.before (field in gds), 145
 Change.existed (field in gds), 146
 Change.path (field in gds), 145
 Change.present (field in gds), 146
 charge (procedure in host), 210
 charset (procedure in gettext), 161
 check_args (procedure in jinja), 243
 check_code (procedure in highlight), 190
 check_index (procedure in gds), 149
 check_render (procedure in core), 126
 check_sources (procedure in gds), 149
 check_style (procedure in highlight), 190
 child (procedure in treesitter), 341
 child_count (procedure in core), 126
 (procedure in treesitter), 341
 children (procedure in core), 126
 (procedure in treesitter), 341
 Children (struct in core), 94
 Children.end (field in core), 94
 Children.next (field in core), 94
 Class (struct in pycan), 281
 Class.bases (field in pycan), 282
 Class.decorators (field in pycan), 282
 Class.doc (field in pycan), 282
 Class.keywords (field in pycan), 282

- Class.module (field in pyscan), 281
- Class.mro_complete (field in pyscan), 282
- Class.mro_done (field in pyscan), 282
- Class.mro_order (field in pyscan), 282
- Class.name (field in pyscan), 281
- Class.node (field in pyscan), 282
- Class.outer (field in pyscan), 282
- Class.qualname (field in pyscan), 282
- Class.scope (field in pyscan), 282
- class_attr (procedure in pyscan), 292
- class_directive (procedure in readers/rst), 319
- clean (procedure in project), 278
- clear_cache (procedure in jinja), 244
- cli (package), 86
- clone_message (procedure in gettext), 161
- close (procedure in graphdog), 172
 - (procedure in readers/rst), 319
- code_caption (procedure in core), 127
- code_text (procedure in readers/rst), 319
- Color (struct in highlight), 175
- Color.hex (field in highlight), 175
- Color.set (field in highlight), 175
- Compact (struct in i18n), 223
- Compact.mode (field in i18n), 223
- Compact.name (field in i18n), 223
- compile_file (procedure in typst), 348
- compile_plural (procedure in gettext), 161
- compile_text (procedure in typst), 348
- Config (struct in autodoc), 83
 - (struct in odindoc), 249
 - (struct in project), 260
 - (struct in readers/sphinx), 332
- Config.add_function_parentheses (field in project), 262
 - (field in readers/sphinx), 333
- Config.add_module_names (field in project), 262
 - (field in readers/sphinx), 333
- Config.author (field in project), 260
- Config.autoclass_content (field in autodoc), 83
 - (field in project), 264
- Config.autodoc (field in readers/sphinx), 334
- Config.autodoc_class_signature (field in autodoc), 83
 - (field in project), 264
- Config.autodoc_default_options (field in autodoc), 83
 - (field in project), 265
- Config.autodoc_docstring_signature (field in autodoc), 83
 - (field in project), 265
- Config.autodoc_inherit_docstrings (field in autodoc), 83
 - (field in project), 265
- Config.autodoc_member_order (field in autodoc), 83
 - (field in project), 265
- Config.autodoc_mock_imports (field in autodoc), 83
 - (field in project), 265
- Config.autodoc_preserve_defaults (field in autodoc), 83
 - (field in project), 265
- Config.autodoc_source_paths (field in project), 264
- Config.autodoc_type_aliases (field in autodoc), 83
 - (field in project), 265
- Config.autodoc_typehints (field in autodoc), 83
 - (field in project), 265
- Config.autodoc_typehints_description_target (field in autodoc), 83
 - (field in project), 265
- Config.autodoc_typehints_format (field in autodoc), 83
 - (field in project), 265
- Config.autodoc_use_type_comments (field in autodoc), 84
 - (field in project), 265
- Config.autodoc_warningiserror (field in project), 265
- Config.autosectionlabel_maxdepth (field in project), 263
- Config.autosectionlabel_prefix_document (field in project), 263
- Config.build_dir (field in project), 260
- Config.copyright (field in project), 260
- Config.default_options (field in odindoc), 249
- Config.default_role (field in project), 261
 - (field in readers/sphinx), 332
- Config.directive_aliases (field in project), 263
 - (field in readers/sphinx), 334
- Config.docname (field in readers/sphinx), 332
- Config.exclude_patterns (field in project), 261
- Config.extensions (field in project), 261
- Config.extlinks (field in project), 263
 - (field in readers/sphinx), 332
- Config.extlinks_detect_hardcoded_links (field in project), 263
- Config.figure_language_filename (field in project), 270
- Config.gettext_additional_targets (field in project), 270
- Config.gettext_allow_fuzzy_translations (field in project), 270
- Config.gettext_auto_build (field in project), 270
- Config.gettext_compact (field in project), 270
- Config.gettext_exclude_patterns (field in project), 270
- Config.gettext_language_team (field in project), 270
- Config.gettext_last_translator (field in project), 270
- Config.gettext_location (field in project), 270
- Config.gettext_uuid (field in project), 270
- Config.graphviz_dot (field in project), 268
- Config.graphviz_dot_args (field in project), 268

- Config.graphviz_output_format (field in project), 268
- Config.highlight_language (field in project), 262
- (field in readers/sphinx), 332
- Config.html_additional_pages (field in project), 267
- Config.html_baseurl (field in project), 267
- Config.html_context (field in project), 267
- Config.html_copy_source (field in project), 267
- Config.html_css_files (field in project), 266
- Config.html_domain_indices (field in project), 267
- Config.html_extra_path (field in project), 266
- Config.html_favicon (field in project), 266
- Config.html_file_suffix (field in project), 267
- Config.html_js_files (field in project), 266
- Config.html_last_updated_fmt (field in project), 266
- Config.html_link_suffix (field in project), 267
- Config.html_logo (field in project), 266
- Config.html_permalinks (field in project), 266
- Config.html_permalinks_icon (field in project), 267
- Config.html_secnumber_suffix (field in project), 267
- Config.html_short_title (field in project), 266
- Config.html_short_title_derived (field in project), 270
- Config.html_show_copyright (field in project), 267
- Config.html_show_sourcelink (field in project), 267
- Config.html_show_sphinx (field in project), 267
- Config.html_sourcelink_suffix (field in project), 267
- Config.html_split_index (field in project), 267
- Config.html_static_path (field in project), 266
- Config.html_theme (field in project), 266
- Config.html_theme_options (field in project), 266
- Config.html_title (field in project), 266
- Config.html_title_derived (field in project), 270
- Config.html_use_index (field in project), 267
- Config.include_patterns (field in project), 261
- Config.include_roots (field in project), 261
- Config.intersphinx (field in readers/sphinx), 333
- Config.intersphinx_cache_limit (field in project), 264
- Config.intersphinx_disabled_reftypes (field in project), 264
- Config.intersphinx_mapping (field in project), 264
- Config.intersphinx_resolve_self (field in project), 264
- (field in readers/sphinx), 333
- Config.intersphinx_timeout (field in project), 264
- Config.keep_warnings (field in project), 263
- Config.labels (field in readers/sphinx), 333
- Config.language (field in project), 261
- Config.locale_dirs (field in project), 269
- Config.manpages_url (field in project), 263
- (field in readers/sphinx), 333
- Config.math_eqref_format (field in project), 262
- Config.math_number_all (field in project), 262
- (field in readers/sphinx), 332
- Config.math_numfig (field in project), 262
- Config.mathjax3_config (field in project), 264
- Config.mathjax4_config (field in project), 264
- Config.mathjax_config_path (field in project), 264
- Config.mathjax_display (field in project), 264
- Config.mathjax_inline (field in project), 264
- Config.mathjax_options (field in project), 264
- Config.mathjax_path (field in project), 263
- Config.member_order (field in odindoc), 249
- Config.modindex_common_prefix (field in project), 267
- Config.myst_amsmath_enable (field in project), 269
- Config.myst_commonmark_only (field in project), 269
- Config.myst_disable_syntax (field in project), 269
- Config.myst_dmath_allow_digits (field in project), 269
- Config.myst_dmath_allow_labels (field in project), 268
- Config.myst_dmath_allow_space (field in project), 269
- Config.myst_dmath_double_inline (field in project), 269
- Config.myst_dmath_enable (field in project), 269
- Config.myst_enable_extensions (field in project), 268
- Config.myst_footnote_transition (field in project), 269
- Config.myst_heading_anchors (field in project), 268

- Config.myst_html_meta (field in project), 269
- Config.myst_linkify_fuzzy_links (field in project), 269
- Config.myst_sub_delimiters (field in project), 269
- Config.myst_substitutions (field in project), 269
- Config.myst_url_schemes (field in project), 269
- Config.myst_words_per_minute (field in project), 269
- Config.nitpick_ignore (field in project), 262
- Config.nitpick_ignore_regex (field in project), 262
- Config.nitpick_patterns (field in project), 262
- Config.nitpicky (field in project), 262
- Config.numfig (field in project), 262 (field in readers/sphinx), 332
- Config.numfig_format (field in project), 262
- Config.numfig_secnum_depth (field in project), 262
- Config.object_types (field in project), 263 (field in readers/sphinx), 334
- Config.odin_autoapi_add_toctree_entry (field in project), 266
- Config.odin_autoapi_dirs (field in project), 265
- Config.odin_autoapi_generate_api_docs (field in project), 266
- Config.odin_autoapi_member_order (field in project), 266
- Config.odin_autoapi_options (field in project), 266
- Config.odin_autoapi_root (field in project), 265
- Config.odin_autodoc (field in readers/sphinx), 334
- Config.odin_collections (field in project), 265
- Config.option_emphasise_placeholders (field in project), 263 (field in readers/sphinx), 333
- Config.pdf_documents (field in project), 267
- Config.pdf_font_paths (field in project), 268
- Config.pdf_fonts (field in project), 268
- Config.pdf_logo (field in project), 268
- Config.pdf_packages (field in project), 268
- Config.pdf_paper_size (field in project), 268
- Config.pdf_preamble (field in project), 268
- Config.pdf_show_urls (field in project), 268
- Config.pdf_toplevel_sectioning (field in project), 268
- Config.primary_domain (field in project), 261 (field in readers/sphinx), 332
- Config.project (field in project), 260
- Config.pygments_dark_style (field in project), 262
- Config.pygments_style (field in project), 262
- Config.python_display_short_literal_types (field in autodoc), 84 (field in project), 265
- Config.release (field in project), 261 (field in readers/sphinx), 333
- Config.root_doc (field in project), 261
- Config.rst_epilog (field in project), 261 (field in readers/sphinx), 332
- Config.rst_prolog (field in project), 261 (field in readers/sphinx), 332
- Config.show_authors (field in project), 262 (field in readers/sphinx), 332
- Config.smartquotes (field in project), 263
- Config.source_dir (field in project), 260
- Config.source_suffix (field in project), 261
- Config.strip_signature_backslash (field in autodoc), 84 (field in project), 265
- Config.suppress_warnings (field in project), 263
- Config.tab_width (field in autodoc), 84
- Config.templates_path (field in project), 261
- Config.toc_object_entries (field in project), 262 (field in readers/sphinx), 333
- Config.toc_object_entries_show_parents (field in project), 263 (field in readers/sphinx), 333
- Config.today (field in project), 261 (field in readers/sphinx), 333
- Config.today_fmt (field in project), 261
- Config.todo_include_todos (field in project), 263 (field in readers/sphinx), 332
- Config.translation_progress_classes (field in project), 270
- Config.trim_doctest_flags (field in project), 263 (field in readers/sphinx), 333
- Config.trim_footnote_reference_space (field in project), 263
- Config.typehint_fields (field in autodoc), 84
- Config.typst (field in project), 268
- Config.using (field in readers/sphinx), 332
- Config.version (field in project), 261 (field in readers/sphinx), 333
- CONFIG_FILE (constant in project), 280
- confined (procedure in host), 210
- Confinement (enum in host), 199
- confinement (procedure in host), 210
- Confinement.Inside (enumerator in host), 199
- Confinement.Outside (enumerator in host), 199
- Confinement.Through_Link (enumerator in host), 199
- CONFORMANCE (constant in readers/commonmark), 307 (constant in readers/rst), 324

- (constant in readers/sphinx), 337
- (constant in render/html), 337
- (constant in render/text), 338
- (constant in render/typst), 339
- Conformance (enum in core), 94
- Conformance.Conformant (enumerator in core), 94
- Conformance.Partial (enumerator in core), 94
- Conformance.Planned (enumerator in core), 94
- containing_root (procedure in host), 211
- Content (enum in core), 94
- Content.Blocks (enumerator in core), 94
- Content.Inlines (enumerator in core), 94
- Content.Nothing (enumerator in core), 94
- Content.Structure (enumerator in core), 94
- content_error (procedure in core), 127
- content_model (procedure in core), 127
- Content_Policy (enum in core), 94
- Content_Policy.Allow (enumerator in core), 95
- Content_Policy.Omit (enumerator in core), 94
- Content_Policy.Refuse (enumerator in core), 94
- Content_Rule (enum in readers/rst), 307
- Content_Rule.None (enumerator in readers/rst), 307
- Content_Rule.Optional (enumerator in readers/rst), 307
- Content_Rule.Required (enumerator in readers/rst), 307
- contents_scope (procedure in core), 127
- Context (struct in graphdog), 168
- Context.gvc (field in graphdog), 168
- context_of (procedure in readers/commonmark), 305
- convert (procedure in core), 127
- convert_artifact (procedure in host), 211
- convert_text (procedure in core), 127
- copy_payload (procedure in core), 128
- copy_subtree (procedure in core), 128
- core (package), 88
- CORE_CAPACITY (constant in core), 141
- CORE_DEPTH (constant in core), 141
- CORE_NODES (constant in core), 141
- CORE_STACK (constant in core), 142
- CORE_UNHANDLED_KIND (constant in core), 142
- CORE_WORK (constant in core), 142
- count_tokens (procedure in highlight), 190
- CRASH_EXIT (constant in host), 221
- crash_on_assertion (procedure in host), 211
- create_environment (procedure in jinja), 244
- curl (foreign import in fetch), 145
- current (procedure in core), 128
- current_kind (procedure in core), 128
- Cursor (struct in treesitter), 339
- Cursor.ctx (field in treesitter), 339
- Cursor.id (field in treesitter), 339
- Cursor.tree (field in treesitter), 339
- cursor_delete (procedure in treesitter), 341

- cursor_field_name (procedure in treesitter), 341
- cursor_first_child (procedure in treesitter), 341
- cursor_new (procedure in treesitter), 341
- cursor_next_sibling (procedure in treesitter), 341
- cursor_node (procedure in treesitter), 341
- cursor_parent (procedure in treesitter), 342
- custom (procedure in readers/rst), 319
- custom_attribute (procedure in core), 128
- cut (procedure in readers/rst), 319
- Cycler (struct in jinja), 235
- Cycler.items (field in jinja), 235
- Cycler.pos (field in jinja), 235

D

- Decl (struct in odindoc), 249
- Decl.attributes (field in odindoc), 249
- Decl.availability (field in odindoc), 250
- Decl.docs (field in odindoc), 249
- Decl.file (field in odindoc), 250
- Decl.foreign_lib (field in odindoc), 249
- Decl.kind (field in odindoc), 249
- Decl.line (field in odindoc), 250
- Decl.members (field in odindoc), 249
- Decl.name (field in odindoc), 249
- Decl.order (field in odindoc), 250
- Decl.private (field in odindoc), 249
- Decl.signature (field in odindoc), 249
- Decl.targets (field in odindoc), 249
- Decl.variant (field in odindoc), 250
- decode_last_rune (procedure in core), 128
- decode_rune (procedure in core), 128
- dedent (procedure in readers/rst), 319
- deeper_than (procedure in treesitter), 342
- Def (struct in pyscan), 282
- Def.annotation (field in pyscan), 283
- Def.class (field in pyscan), 283
- Def.function (field in pyscan), 283
- Def.kind (field in pyscan), 282
- Def.line (field in pyscan), 283
- Def.member (field in pyscan), 283
- Def.name (field in pyscan), 282
- Def.node (field in pyscan), 282
- Def.runtime (field in pyscan), 282
- Def.target (field in pyscan), 283
- Def.tuple (field in pyscan), 283
- Def.value (field in pyscan), 283
- Def_Kind (enum in pyscan), 283
- Def_Kind.Annotation (enumerator in pyscan), 283
- Def_Kind.Assign (enumerator in pyscan), 283
- Def_Kind.Class (enumerator in pyscan), 283
- Def_Kind.Function (enumerator in pyscan), 283
- Def_Kind.Import (enumerator in pyscan), 283
- Def_Kind.Import_From (enumerator in pyscan), 284
- Def_Kind.Type_Alias (enumerator in pyscan), 284
- def_object (procedure in pyscan), 292
- DEFAULT_BUDGET (constant in host), 221

- default_config (procedure in autodoc), 86
 - (procedure in readers/sphinx), 336
- DEFAULT_FIGURE_FILENAME (constant in i18n), 231
- default_limits (procedure in core), 128
- default_merge_options (procedure in gettext), 161
- Default_Option (struct in autodoc), 84
- Default_Option.flag (field in autodoc), 84
- Default_Option.name (field in autodoc), 84
- Default_Option.value (field in autodoc), 84
- default_policy (procedure in core), 129
- default_read_options (procedure in core), 129
- default_request (procedure in core), 129
- default_settings (procedure in readers/commonmark), 305
 - (procedure in readers/rst), 319
- default_stack_budget (procedure in host), 211
- DEFAULT_STACK_BYTES (constant in core), 142
- default_write_options (procedure in gettext), 161
- defaults (package), 142
- delete_error (procedure in graphdog), 172
- describe (procedure in host), 211
- describe_all (procedure in host), 211
- destroy (procedure in fetch), 145
 - (procedure in gettext), 161
 - (procedure in pyscan), 292
- destroy_environment (procedure in jinja), 244
- destroy_error (procedure in gettext), 161
 - (procedure in jinja), 244
- destroy_storage (procedure in host), 211
- destroy_template (procedure in i18n), 228
- destroy_translator (procedure in gettext), 161
- diagnostic (procedure in core), 129
- Diagnostic (struct in core), 95
 - (struct in typst), 346
- Diagnostic.args (field in core), 95
- Diagnostic.column (field in typst), 346
- Diagnostic.file (field in typst), 346
- Diagnostic.line (field in typst), 346
- Diagnostic.message (field in typst), 346
- Diagnostic.severity (field in typst), 346
- Diagnostic.span (field in core), 95
- Diagnostic.stdin (field in typst), 346
- Diagnostic.using (field in core), 95
- Diagnostic_Arg (struct in core), 95
- Diagnostic_Arg.int (field in core), 95
- Diagnostic_Arg.kind (field in core), 95
- Diagnostic_Arg.span (field in core), 95
- Diagnostic_Arg.static (field in core), 95
- DIAGNOSTICS_SCHEMA (constant in host), 221
- dict (procedure in jinja), 244
- Dict (struct in jinja), 235
- Dict allocator (field in jinja), 235
- Dict.index (field in jinja), 235
- Dict.keys (field in jinja), 235
- Dict.kind (field in jinja), 235
- Dict.name (field in jinja), 235
- Dict.values (field in jinja), 235
- Dict_Kind (enum in jinja), 235
- Dict_Kind.Dict (enumerator in jinja), 235
- Dict_Kind.Module (enumerator in jinja), 235
- Dict_Kind.Namespace (enumerator in jinja), 235
- Dir_Opt (struct in readers/rst), 308
- Dir_Opt.lines (field in readers/rst), 308
- Dir_Opt.span (field in readers/rst), 308
- Dir_Opt.spec (field in readers/rst), 308
- Dir_Spec (struct in readers/rst), 308
- Dir_Spec.arg (field in readers/rst), 308
- Dir_Spec.content (field in readers/rst), 308
- Dir_Spec.final_ws (field in readers/rst), 308
- Dir_Spec.kind (field in readers/rst), 308
- Dir_Spec.name (field in readers/rst), 308
- Dir_Spec.opts (field in readers/rst), 308
- Dir_Spec.value (field in readers/rst), 308
- Directive (struct in readers/commonmark), 295
 - (struct in readers/rst), 308
- Directive.arg (field in readers/commonmark), 296
 - (field in readers/rst), 308
- Directive.arg_block (field in readers/rst), 309
- Directive.classes (field in readers/commonmark), 296
- Directive.content (field in readers/commonmark), 296
 - (field in readers/rst), 309
- Directive.ext (field in readers/rst), 309
- Directive.name (field in readers/commonmark), 295
- Directive.name_span (field in readers/commonmark), 295
 - (field in readers/rst), 308
- Directive.nopts (field in readers/rst), 308
- Directive.options (field in readers/commonmark), 296
- Directive.opts (field in readers/rst), 308
- Directive.sections (field in readers/commonmark), 296
 - (field in readers/rst), 309
- Directive.source (field in readers/commonmark), 296
- Directive.span (field in readers/commonmark), 296
 - (field in readers/rst), 308
- Directive.spec (field in readers/rst), 308
- Directive.subst (field in readers/rst), 309
- Directive.subst_name (field in readers/rst), 309
- Directive_Alias (struct in readers/sphinx), 334
- Directive_Alias.name (field in readers/sphinx), 334
- Directive_Alias.target (field in readers/sphinx), 334
- Directive_Handler (struct in readers/commonmark), 296
 - (struct in readers/rst), 309
- Directive_Handler.arg (field in readers/rst), 309

- Directive_Handler.content (field in readers/rst), 309
 - Directive_Handler.final_ws (field in readers/rst), 309
 - Directive_Handler.name (field in readers/commonmark), 296
(field in readers/rst), 309
 - Directive_Handler.options (field in readers/rst), 309
 - Directive_Handler.run (field in readers/commonmark), 296
(field in readers/rst), 309
 - directive_node (procedure in readers/rst), 319
 - Directive_Proc (type in readers/commonmark), 296
(type in readers/rst), 309
 - Directive_Result (enum in readers/commonmark), 296
 - Directive_Result.Failed (enumerator in readers/commonmark), 296
 - Directive_Result.Ran (enumerator in readers/commonmark), 296
 - Directive_Result.Unknown (enumerator in readers/commonmark), 296
 - directive_shape (procedure in readers/rst), 319
 - Directive_Shape (struct in readers/commonmark), 297
(struct in readers/rst), 309
 - Directive_Shape.arg (field in readers/rst), 310
 - Directive_Shape.argument (field in readers/commonmark), 297
 - Directive_Shape.content (field in readers/rst), 310
 - Directive_Shape.final_ws (field in readers/rst), 310
 - Directive_Shape.options (field in readers/commonmark), 297
(field in readers/rst), 310
 - DIRECTIVE_UNKNOWN (constant in readers/rst), 324
 - discharge (procedure in host), 212
 - Disk (struct in host), 199
 - Disk.chunks (field in host), 200
 - Disk.dir (field in host), 200
 - Disk.failure (field in host), 200
 - Disk.limit (field in host), 200
 - Disk.mutex (field in host), 200
 - Disk.peak (field in host), 200
 - Disk.refused (field in host), 200
 - Disk.used (field in host), 200
 - disk_allocator (procedure in host), 212
 - DISK_CHUNK_BYTES (constant in host), 221
 - disk_destroy (procedure in host), 212
 - Disk_Failure (enum in host), 200
 - Disk_Failure.Limit (enumerator in host), 200
 - Disk_Failure.None (enumerator in host), 200
 - Disk_Failure.Space (enumerator in host), 200
 - Disk_Failure.System (enumerator in host), 201
 - DISK_MARGIN (constant in host), 221
 - disk_owns (procedure in host), 212
 - display_path (procedure in host), 212
 - Doc (struct in pyscan), 284
 - Doc.ok (field in pyscan), 284
 - Doc.text (field in pyscan), 284
 - document (procedure in autodoc), 86
(procedure in odindoc), 255
 - Document (struct in core), 95
 - Document.bank (field in core), 95
 - Document.generation (field in core), 95
 - Document.revision (field in core), 95
 - Document.ws (field in core), 95
 - Document_Reading (enum in core), 96
 - Document_Reading.Docutils (enumerator in core), 96
 - Document_Reading.Sphinx (enumerator in core), 96
 - document_valid (procedure in core), 129
 - domain_for (procedure in i18n), 228
 - domain_plural (procedure in gettext), 162
 - domain_plural_with_context (procedure in gettext), 162
 - domain_text (procedure in gettext), 162
 - domain_with_context (procedure in gettext), 162
 - dump (procedure in host), 212
 - dump_tree (procedure in core), 129
- ## E
- each_chunk (procedure in host), 212
 - edit_plan (procedure in gds), 149
 - embed_begin (procedure in readers/rst), 319
 - embed_end (procedure in readers/rst), 320
 - EMBEDDED (constant in typst), 349
 - embedded_catalog (procedure in i18n), 228
 - emit (procedure in core), 129
 - emit_byte (procedure in core), 129
 - emit_int (procedure in core), 129
 - emit_mark (procedure in core), 130
 - Emit_Mode (enum in core), 96
 - Emit_Mode.Measure (enumerator in core), 96
 - Emit_Mode.Write (enumerator in core), 96
 - emit_preamble (procedure in render/typst), 339
 - emit_rune (procedure in core), 130
 - Emitter (struct in core), 96
 - Emitter.mapped (field in core), 96
 - Emitter.mode (field in core), 96
 - Emitter.out (field in core), 96
 - Emitter.source_map (field in core), 96
 - Emitter.written (field in core), 96
 - encode_rune (procedure in core), 130
 - end_byte (procedure in treesitter), 342
 - end_node (procedure in core), 130
 - end_point (procedure in treesitter), 342
 - Engine (enum in graphdog), 168
 - Engine.Circo (enumerator in graphdog), 168
 - Engine.Dot (enumerator in graphdog), 168
 - Engine.Fdp (enumerator in graphdog), 168
 - Engine.Neato (enumerator in graphdog), 168

- Engine.Osage (enumerator in graphdog), 169
- Engine.Patchwork (enumerator in graphdog), 169
- Engine.Sfdp (enumerator in graphdog), 168
- Engine.Twopi (enumerator in graphdog), 169
- engine_name (procedure in graphdog), 172
- ENGLISH_LABELS (variable in core), 142
- enter_source (procedure in readers/rst), 320
- Environment (struct in jinja), 236
- Environment allocator (field in jinja), 236
- Environment.arena (field in jinja), 236
- Environment.backing (field in jinja), 236
- Environment.cache (field in jinja), 236
- Environment.filters (field in jinja), 236
- Environment.globals (field in jinja), 236
- Environment.options (field in jinja), 236
- Environment.out_of_memory (field in jinja), 236
- Environment.stack (field in jinja), 236
- Environment.tests (field in jinja), 236
- equal (procedure in treesitter), 342
- Error (enum in fetch), 143
 - (enum in inventory), 231
 - (struct in gettext), 152
 - (struct in graphdog), 169
 - (struct in jinja), 236
- Error.Bad_Compression (enumerator in inventory), 232
- Error.column (field in gettext), 153
 - (field in graphdog), 169
- Error.excerpt (field in gettext), 153
 - (field in jinja), 237
- Error.file (field in gettext), 152
- Error.hint (field in gettext), 153
 - (field in graphdog), 169
 - (field in jinja), 237
- Error.kind (field in gettext), 152
 - (field in graphdog), 169
 - (field in jinja), 236
- Error.line (field in gettext), 152
 - (field in graphdog), 169
 - (field in jinja), 236
- Error.message (field in gettext), 153
 - (field in graphdog), 169
 - (field in jinja), 236
- Error.Missing (enumerator in fetch), 143
- Error.near (field in graphdog), 169
- Error.Network (enumerator in fetch), 143
- Error.None (enumerator in fetch), 143
 - (enumerator in inventory), 231
- Error.Out_Of_Memory (enumerator in inventory), 232
- Error.Status (enumerator in fetch), 144
- Error.template (field in jinja), 236
- Error.Timeout (enumerator in fetch), 144
- Error.Too_Large (enumerator in fetch), 144
- Error.Truncated (enumerator in inventory), 232
- Error.Unknown_Version (enumerator in inventory), 232
- Error.Unsupported (enumerator in fetch), 143
- Error_Kind (enum in gettext), 153
 - (enum in graphdog), 169
 - (enum in jinja), 237
- Error_Kind.Charset (enumerator in gettext), 153
- Error_Kind.Duplicate (enumerator in gettext), 153
- Error_Kind.Empty_Input (enumerator in graphdog), 170
- Error_Kind.Encoding (enumerator in gettext), 153
- Error_Kind.File_Refused (enumerator in graphdog), 170
- Error_Kind.Html_Label (enumerator in graphdog), 169
- Error_Kind.Io (enumerator in gettext), 153
- Error_Kind.Layout (enumerator in graphdog), 170
- Error_Kind.Limit (enumerator in jinja), 237
- Error_Kind.Memory (enumerator in gettext), 153
 - (enumerator in jinja), 237
- Error_Kind.Missing_Engine (enumerator in graphdog), 170
- Error_Kind.Missing_Format (enumerator in graphdog), 170
- Error_Kind.Mo_Format (enumerator in gettext), 153
- Error_Kind.None (enumerator in gettext), 153
 - (enumerator in graphdog), 169
 - (enumerator in jinja), 237
- Error_Kind.Not_Found (enumerator in jinja), 237
- Error_Kind.Not_Initialized (enumerator in graphdog), 170
- Error_Kind.Out_Of_Memory (enumerator in graphdog), 170
- Error_Kind.Plural (enumerator in gettext), 153
- Error_Kind.Render (enumerator in graphdog), 170
- Error_Kind.Runtime (enumerator in jinja), 237
- Error_Kind.Syntax (enumerator in gettext), 153
 - (enumerator in graphdog), 169
 - (enumerator in jinja), 237
- Error_Kind.Type (enumerator in jinja), 237
- Error_Kind.Undefined (enumerator in jinja), 237
- error_title (procedure in graphdog), 172
- ERROR_TITLES (variable in gettext), 167
- escape (procedure in jinja), 244
- escape_string (procedure in gettext), 162
 - (procedure in jinja), 244
- evaluate (procedure in jinja), 244
- excerpt_at (procedure in host), 212
- exchange (procedure in host), 212
- exit_code (procedure in host), 213
- exported (procedure in pycan), 292
- Extension (struct in readers/
 - commonmark), 297
 - (struct in readers/rst), 310
- Extension.begin (field in readers/
 - commonmark), 297

(field in readers/rst), 310
 Extension.directive (field in readers/
 commonmark), 297
 Extension.directives (field in readers/
 commonmark), 297
 (field in readers/rst), 310
 Extension.end (field in readers/
 commonmark), 297
 (field in readers/rst), 310
 Extension.lookup (field in readers/rst), 310
 Extension.prologue (field in readers/rst),
 310
 Extension.resolve (field in readers/
 commonmark), 298
 Extension.role (field in readers/
 commonmark), 297
 (field in readers/rst), 310
 Extension.roles (field in readers/
 commonmark), 297
 (field in readers/rst), 310
 Extension.shape (field in readers/
 commonmark), 297
 Extension.substitution (field in readers/
 rst), 310
 Extension.table (field in readers/
 commonmark), 297
 Extension.user (field in readers/
 commonmark), 298
 (field in readers/rst), 311
 extension_as (procedure in core), 130
 Extlink (struct in readers/sphinx), 334
 Extlink.caption (field in readers/sphinx),
 334
 Extlink.name (field in readers/sphinx), 334
 Extlink.url (field in readers/sphinx), 334
 extra_anchor (procedure in core), 130
 extra_text (procedure in core), 130
 extract (procedure in i18n), 229
 Extract_Options (struct in i18n), 223
 Extract_Options.base (field in i18n), 223
 Extract_Options.docname (field in i18n),
 223
 Extract_Options.location (field in i18n),
 223
 Extract_Options.path (field in i18n), 223
 Extract_Options.targets (field in i18n), 223
 Extract_Options.uuid (field in i18n), 223

F

fail (procedure in jinja), 245
 fetch (package), 143
 field (procedure in treesitter), 342
 Field (struct in gds), 146
 Field.literal (field in gds), 146
 Field.name (field in gds), 146
 Field_Id (type in treesitter), 340
 field_name_for_child (procedure in
 treesitter), 342
 figure_filename (procedure in i18n), 229
 figure_for_language (procedure in i18n),
 229
 File_Access (struct in graphdog), 170
 File_Access.procedure (field in graphdog),
 170
 File_Access.user (field in graphdog), 170

FILE_ATTRIBUTES (constant in
 graphdog), 173
 File_Read (struct in odindoc), 250
 (struct in pyscan), 284
 File_Read.path (field in odindoc), 250
 (field in pyscan), 284
 File_Read.text (field in odindoc), 250
 (field in pyscan), 284
 File_Reader (struct in odindoc), 250
 (struct in pyscan), 284
 File_Reader.list (field in odindoc), 250
 File_Reader.procedure (field in odindoc),
 250
 (field in pyscan), 284
 File_Reader.reserve (field in odindoc), 250
 File_Reader.user (field in odindoc), 250
 (field in pyscan), 284
 Files (struct in host), 201
 Files.canonical (field in host), 201
 Files.ledger (field in host), 201
 Files.max_bytes (field in host), 201
 Files.project (field in host), 201
 Files.roots (field in host), 201
 files_provider (procedure in host), 213
 filesystem_loader (procedure in jinja), 245
 fill_labels (procedure in i18n), 229
 find (procedure in gettext), 162
 (procedure in inventory), 233
 find_catalog_file (procedure in gettext),
 162
 find_def (procedure in pyscan), 292
 find_folded (procedure in inventory), 233
 find_lexer (procedure in highlight), 191
 find_lexer_for_filename (procedure in
 highlight), 191
 find_program (procedure in typst), 348
 find_reader (procedure in core), 130
 find_renderer (procedure in core), 130
 find_role (procedure in readers/rst), 320
 find_style (procedure in highlight), 191
 first_child (procedure in core), 130
 first_problem (procedure in core), 131
 fitting_budget (procedure in host), 213
 Flag_Or_Name (struct in project), 270
 Flag_Or_Name.name (field in project), 270
 Flag_Or_Name.on (field in project), 270
 float (procedure in jinja), 245
 Folder_Entry (struct in host), 201
 Folder_Entry.inode (field in host), 201
 Folder_Entry.name (field in host), 201
 Folder_Entry.type (field in host), 201
 folders_of (procedure in host), 213
 footnote_symbol (procedure in readers/rst/
 inline), 329
 Format (enum in graphdog), 170
 (enum in host), 202
 Format.Commonmark (enumerator in
 host), 202
 Format.Dot (enumerator in graphdog), 171
 Format.Png (enumerator in graphdog), 171
 Format.Rst (enumerator in host), 202
 Format.Svg (enumerator in graphdog), 171
 Format.Typst (enumerator in host), 202
 format_date (procedure in i18n), 229
 format_error (procedure in gettext), 162
 (procedure in graphdog), 172
 (procedure in jinja), 245

format_message (procedure in graphdog), 172
 format_stats (procedure in gettext), 163
 Frame (struct in readers/rst), 311
 Frame.src (field in readers/rst), 311
 Frame.tab (field in readers/rst), 311
 Frame.text (field in readers/rst), 311
 FULL_VERSION (constant in cli), 87
 function (procedure in jinja), 245
 Function (struct in pyscan), 284
 Function.class (field in pyscan), 285
 Function.decorators (field in pyscan), 285
 Function.doc (field in pyscan), 285
 Function.is_async (field in pyscan), 285
 Function.is_lambda (field in pyscan), 285
 Function.module (field in pyscan), 284
 Function.name (field in pyscan), 284
 Function.node (field in pyscan), 284
 Function.params (field in pyscan), 285
 Function.qualname (field in pyscan), 284
 Function.returns (field in pyscan), 285
 Function.type_args (field in pyscan), 285
 Function.type_comment (field in pyscan), 285
 function.type_returns (field in pyscan), 285
 FUZZY_MIN_WORK (constant in gettext), 167
 FUZZY_THRESHOLD (constant in gettext), 167
 FUZZY_WORK_PER_BYTE (constant in gettext), 167

G

gds (package), 145
 get (procedure in fetch), 145
 (procedure in jinja), 245
 get_attr (procedure in pyscan), 292
 get_opt (procedure in readers/rst), 320
 gettext (package), 151
 graphdog (package), 167
 graphviz (foreign import in graphdog), 174
 grow_session (procedure in host), 213
 Guard (struct in host), 202
 Guard.backing (field in host), 202
 Guard.refused (field in host), 202
 Guard.tripped (field in host), 202
 guard_allocator (procedure in host), 213
 guard_problem (procedure in host), 213
 guard_tripped (procedure in host), 214
 guess_lexer (procedure in highlight), 191

H

has_arg (procedure in jinja), 245
 has_children (procedure in core), 131
 has_class (procedure in core), 131
 has_error (procedure in treesitter), 342
 has_errors (procedure in highlight), 191
 has_option (procedure in readers/rst), 320
 head (procedure in readers/rst), 320
 header_field (procedure in gettext), 163
 Header_Field (struct in gettext), 154
 Header_Field.name (field in gettext), 154
 Header_Field.value (field in gettext), 154
 highlight (package), 175
 highlight/guidedoc (package), 195
 highlight_html (procedure in highlight), 191

highlight_html_lines (procedure in highlight), 191
 highlight_typst (procedure in highlight), 191
 highlighter (procedure in highlight/guidedoc), 195
 Highlighter (struct in core), 96
 Highlighter.procedure (field in core), 97
 Highlighter.user (field in core), 97
 Hook_Proc (type in readers/rst), 311
 host (package), 195
 Host (struct in readers/commonmark), 298
 (struct in readers/rst), 311
 Host.b (field in readers/commonmark), 298
 Host.body (field in readers/rst), 311
 Host.ctx (field in readers/commonmark), 298
 Host.disabled (field in readers/commonmark), 298
 Host.ext (field in readers/commonmark), 298
 Host.html_format (field in readers/commonmark), 299
 Host.inline (field in readers/rst), 311
 Host.items (field in readers/rst), 311
 Host.label_tmp (field in readers/commonmark), 299
 Host.labels (field in readers/commonmark), 298
 Host.myst (field in readers/commonmark), 298
 Host.names (field in readers/commonmark), 298
 Host.nrefs (field in readers/commonmark), 298
 Host.refs (field in readers/commonmark), 298
 Host.rel_docs (field in readers/commonmark), 299
 Host.rel_images (field in readers/commonmark), 299
 Host.settings (field in readers/commonmark), 298
 Host.syntax (field in readers/commonmark), 298
 Host.user (field in readers/rst), 311
 Host.using (field in readers/commonmark), 299
 Host_Directive (struct in readers/rst), 312
 Host_Directive.arg (field in readers/rst), 312
 Host_Directive.content (field in readers/rst), 312
 Host_Directive.name (field in readers/rst), 312
 Host_Directive.name_span (field in readers/rst), 312
 Host_Directive.options (field in readers/rst), 312
 Host_Directive.sections (field in readers/rst), 312
 Host_Directive.span (field in readers/rst), 312

host_lines (procedure in readers/rst), 320
 Host_Option (struct in readers/rst), 312
 Host_Option.name (field in readers/rst), 312
 Host_Option.span (field in readers/rst), 312
 Host_Option.value (field in readers/rst), 312
 host_views (procedure in readers/rst), 320
 Hosted_Result (enum in readers/rst), 312
 Hosted_Result.Failed (enumerator in readers/rst), 312
 Hosted_Result.Ran (enumerator in readers/rst), 312
 Hosted_Result.Unknown (enumerator in readers/rst), 312
 Html_Lines (struct in highlight), 175
 Html_Lines.emphasized (field in highlight), 175
 Html_Lines.first (field in highlight), 175
 Html_Lines.user (field in highlight), 175

I

i18n (package), 222
 id_fold (procedure in core), 131
 identity (procedure in host), 214
 Identity (struct in host), 202
 Identity.device (field in host), 202
 Identity.inode (field in host), 202
 Image_Exists (struct in i18n), 223
 Image_Exists.procedure (field in i18n), 223
 Image_Exists.user (field in i18n), 224
 Image_Localization (struct in i18n), 224
 Image_Localization.docname (field in i18n), 224
 Image_Localization.exists (field in i18n), 224
 Image_Localization.format (field in i18n), 224
 Image_Localization.language (field in i18n), 224
 Import (struct in odindoc), 251
 Import.alias (field in odindoc), 251
 Import.id (field in odindoc), 251
 INCLUDE_MISSING (constant in core), 142
 include_paths (procedure in readers/commonmark), 305
 include_source (procedure in core), 131
 indent_of (procedure in readers/rst), 320
 Index (struct in gettext), 154
 Index.built (field in gettext), 154
 Index.entries (field in gettext), 154
 Index.forms (field in gettext), 154
 Index.fuzzy (field in gettext), 154
 Index.pool (field in gettext), 154
 Index.slots (field in gettext), 154
 Index_Entry (struct in gettext), 154
 Index_Entry.count (field in gettext), 154
 Index_Entry.first (field in gettext), 154
 Index_Entry.message (field in gettext), 154
 Index_Part (struct in i18n), 224
 Index_Part.first (field in i18n), 224
 Index_Part.length (field in i18n), 224
 Index_Part.text (field in i18n), 224
 index_parts (procedure in i18n), 229
 index_plan (procedure in gds), 149
 init (procedure in graphdog), 172
 (procedure in odindoc), 255
 (procedure in pyscan), 292
 init_workspace (procedure in core), 131
 init_workspace_from_bytes (procedure in core), 131
 init_workspace_with_plan (procedure in core), 131
 inline (procedure in readers/rst), 320
 inline_node (procedure in readers/rst), 321
 Inline_State (struct in readers/rst/inline), 325
 Inline_State.ctx (field in readers/rst/inline), 325
 Inline_State.default_role (field in readers/rst/inline), 325
 Inline_State.hook (field in readers/rst/inline), 325
 Inline_State.options (field in readers/rst/inline), 325
 Inline_State.roles (field in readers/rst/inline), 325
 Inline_State.roles_used (field in readers/rst/inline), 325
 Inline_State.source (field in readers/rst/inline), 325
 install_crash_reporter (procedure in host), 214
 integer (procedure in jinja), 246
 interrupted_problem (procedure in host), 214
 Intersphinx_Project (struct in project), 271
 Intersphinx_Project.locations (field in project), 271
 Intersphinx_Project.name (field in project), 271
 Intersphinx_Project.uri (field in project), 271
 Intl_Action (enum in project), 271
 Intl_Action.Create (enumerator in project), 271
 Intl_Action.Not_Changed (enumerator in project), 271
 Intl_Action.Update (enumerator in project), 271
 Intl_File (struct in project), 271
 Intl_File.action (field in project), 271
 Intl_File.path (field in project), 271
 Intl_File.stats (field in project), 271
 Intl_Options (struct in project), 271
 Intl_Options.languages (field in project), 272
 Intl_Options.locale_dir (field in project), 271
 Intl_Options.obsolete (field in project), 272
 Intl_Options.pot_dir (field in project), 271
 Intl_Options.width (field in project), 272
 intl_stat (procedure in project), 278
 intl_update (procedure in project), 278
 INVALID_NODE (type in core), 97
 inventory (package), 231
 Inventory (struct in inventory), 232
 Inventory.by_type (field in inventory), 232
 Inventory.folded (field in inventory), 232

Inventory.items (field in inventory), 232
 Inventory.project (field in inventory), 232
 Inventory.version (field in inventory), 232
 io_problem (procedure in host), 214
 is_ascii_alnum (procedure in core), 132
 is_ascii_digit (procedure in core), 132
 is_ascii_punct (procedure in core), 132
 is_ascii_space (procedure in core), 132
 is_builtin (procedure in pyscan), 293
 is_builtin_exception (procedure in
 pyscan), 293
 is_error (procedure in treesitter), 342
 is_exception (procedure in pyscan), 293
 is_extra (procedure in treesitter), 342
 is_indented (procedure in readers/rst), 321
 is_markup (procedure in jinja), 246
 is_missing (procedure in treesitter), 343
 is_named (procedure in treesitter), 343
 is_none (procedure in jinja), 246
 is_null (procedure in treesitter), 343
 is_stage_name (procedure in host), 214
 is_undefined (procedure in jinja), 246
 is_unicode_punctuation (procedure in
 core), 132
 is_unicode_whitespace (procedure in
 core), 132
 ISSUE_TRACKER (constant in host), 221
 Item (struct in inventory), 232
 Item.display (field in inventory), 233
 Item.name (field in inventory), 232
 Item.priority (field in inventory), 233
 Item.project (field in inventory), 232
 Item.type (field in inventory), 232
 Item.uri (field in inventory), 233
 Item.version (field in inventory), 232
 Iterator (struct in treesitter), 340
 Iterator.count (field in treesitter), 340
 Iterator.index (field in treesitter), 340
 Iterator.named (field in treesitter), 340
 Iterator.node (field in treesitter), 340

J

jinja (package), 234
 join (procedure in project), 278
 join_lines (procedure in readers/rst), 321

K

kept_message (procedure in core), 132
 kernel32 (foreign import in host), 222
 keys (procedure in jinja), 246
 Keyword (struct in jinja), 237
 (struct in pyscan), 285
 Keyword.name (field in jinja), 237
 (field in pyscan), 285
 Keyword.value (field in jinja), 237
 (field in pyscan), 285
 Kind (enum in odindoc), 251
 kind (procedure in core), 132
 (procedure in treesitter), 343
 Kind.Bit_Field (enumerator in odindoc),
 251
 Kind.Bit_Set (enumerator in odindoc), 251
 Kind.Const (enumerator in odindoc), 251
 Kind.Enum (enumerator in odindoc), 251
 Kind.Foreign_Import (enumerator in
 odindoc), 251
 Kind.Proc (enumerator in odindoc), 251

Kind.Proc_Group (enumerator in odindoc),
 251
 Kind.Struct (enumerator in odindoc), 251
 Kind.Type (enumerator in odindoc), 251
 Kind.Union (enumerator in odindoc), 251
 Kind.Var (enumerator in odindoc), 251
 kind_class (procedure in highlight), 191
 kind_is (procedure in highlight), 192
 kind_name (procedure in core), 132
 (procedure in highlight), 192
 kind_parent (procedure in highlight), 192

L

Label (enum in core), 97
 label (procedure in core), 132
 Label.Arguments (enumerator in core), 99
 Label.Attention (enumerator in core), 97
 Label.Availability (enumerator in core), 99
 Label.Caution (enumerator in core), 97
 Label.Code_Author (enumerator in core),
 98
 Label.Contents (enumerator in core), 97
 Label.Danger (enumerator in core), 97
 Label.Error (enumerator in core), 97
 Label.Hint (enumerator in core), 97
 Label.Important (enumerator in core), 97
 Label.Link_Code (enumerator in core), 98
 Label.Link_Definition (enumerator in
 core), 98
 Label.Link_Equation (enumerator in core),
 98
 Label.Link_Heading (enumerator in core),
 98
 Label.Link_Image (enumerator in core), 98
 Label.Link_Table (enumerator in core), 98
 Label.Link_Term (enumerator in core), 98
 Label.Module_Author (enumerator in
 core), 98
 Label.Note (enumerator in core), 97
 Label.Parameters (enumerator in core), 98
 Label.Raisees (enumerator in core), 98
 Label.Return_Type (enumerator in core),
 98
 Label.Return_Values (enumerator in core),
 98
 Label>Returns (enumerator in core), 98
 Label.Section_Author (enumerator in
 core), 98
 Label.See_Also (enumerator in core), 97
 Label.Template_Parameters (enumerator in
 core), 99
 Label.Throws (enumerator in core), 99
 Label.Tip (enumerator in core), 97
 Label.TODO (enumerator in core), 97
 Label.Variables (enumerator in core), 98
 Label.Version_Added (enumerator in core),
 97
 Label.Version_Changed (enumerator in
 core), 98
 Label.Version_Deprecated (enumerator in
 core), 98
 Label.Version_Removed (enumerator in
 core), 98
 Label.Warning (enumerator in core), 97
 LABELS (constant in gds), 151
 Labels (type in core), 99

- Language (struct in treesitter), 340
- language_candidates (procedure in gettext), 163
- leave_source (procedure in readers/rst), 321
- Ledger (struct in host), 203
- Ledger.budget (field in host), 203
- Ledger.held (field in host), 203
- Ledger.refused (field in host), 203
- length (procedure in jinja), 246
- Lexer (enum in highlight), 175
- Lexer.Bash (enumerator in highlight), 177
- Lexer.C (enumerator in highlight), 176
- Lexer.Console (enumerator in highlight), 177
- Lexer.Cpp (enumerator in highlight), 176
- Lexer.Csharp (enumerator in highlight), 176
- Lexer.Css (enumerator in highlight), 177
- Lexer.Default (enumerator in highlight), 177
- Lexer.Diff (enumerator in highlight), 177
- Lexer.Dockerfile (enumerator in highlight), 177
- Lexer.Dot (enumerator in highlight), 177
- Lexer.Go (enumerator in highlight), 176
- Lexer.Guess (enumerator in highlight), 177
- Lexer.Html (enumerator in highlight), 177
- Lexer.Ini (enumerator in highlight), 176
- Lexer.Java (enumerator in highlight), 176
- Lexer.Javascript (enumerator in highlight), 176
- Lexer.Json (enumerator in highlight), 176
- Lexer.Kotlin (enumerator in highlight), 176
- Lexer.Latex (enumerator in highlight), 177
- Lexer.Makefile (enumerator in highlight), 177
- Lexer.Markdown (enumerator in highlight), 177
- Lexer.Odin (enumerator in highlight), 176
- Lexer.Powershell (enumerator in highlight), 177
- Lexer.Pycon (enumerator in highlight), 176
- Lexer.Pytb (enumerator in highlight), 176
- Lexer.Python (enumerator in highlight), 175
- Lexer.Rst (enumerator in highlight), 177
- Lexer.Rust (enumerator in highlight), 176
- Lexer.Sql (enumerator in highlight), 177
- Lexer.Swift (enumerator in highlight), 176
- Lexer.Text (enumerator in highlight), 175
- Lexer.Toml (enumerator in highlight), 176
- Lexer.Typescript (enumerator in highlight), 176
- Lexer.Typst (enumerator in highlight), 177
- Lexer.Xml (enumerator in highlight), 176
- Lexer.Yaml (enumerator in highlight), 176
- Lexer.Zig (enumerator in highlight), 176
- lexer_alias (procedure in highlight), 192
- lexer_info (procedure in highlight), 192
- Lexer_Info (struct in highlight), 177
- Lexer_Info.aliases (field in highlight), 178
- Lexer_Info.filenames (field in highlight), 178
- Lexer_Info.mimetypes (field in highlight), 178
- Lexer_Info.name (field in highlight), 177
- Lexer_Info.spec (field in highlight), 178
- Lexer_Info.step (field in highlight), 178
- lexer_name (procedure in highlight), 192
- LICENSE_NOTICE (constant in cli), 87
- Limits (struct in core), 99
- Limits.max_depth (field in core), 99
- Limits.max_expansion (field in core), 99
- Limits.max_include_depth (field in core), 99
- Limits.max_included_bytes (field in core), 99
- Limits.max_nodes (field in core), 99
- Limits.max_work (field in core), 99
- Limits.stack_bytes (field in core), 99
- Line (struct in readers/rst), 312
 - (type in readers/commonmark), 299
 - (type in readers/rst/inline), 325
- Line.col (field in readers/rst), 313
- Line.offset (field in readers/rst), 313
- Line.raw (field in readers/rst), 313
- Line.text (field in readers/rst), 313
- line_bounds (procedure in core), 133
- line_column (procedure in core), 133
- line_span (procedure in readers/rst), 321
- Line_View (struct in core), 99
- Line_View.offset (field in core), 100
- Line_View.text (field in core), 99
- lines_span (procedure in readers/rst), 321
- link_file (procedure in host), 214
- link_info (procedure in core), 133
- Link_Info (struct in core), 100
- Link_Info.destination (field in core), 100
- Link_Info.kind (field in core), 100
- Link_Info.name (field in core), 100
- Link_Info.resolved (field in core), 100
- Link_Info.title (field in core), 100
- Link_Kind (enum in core), 100
- Link_Kind.Email (enumerator in core), 100
- Link_Kind.External (enumerator in core), 100
- Link_Kind.Internal (enumerator in core), 100
- Link_Kind.Project (enumerator in core), 100
- Link_Kind.Unresolved (enumerator in core), 100
- list (procedure in jinja), 246
- List (struct in jinja), 238
- List allocator (field in jinja), 238
- List.fields (field in jinja), 238
- List.items (field in jinja), 238
- List.tuple (field in jinja), 238
- list_folder (procedure in host), 215
- list_info (procedure in core), 133
- List_Info (struct in core), 100
- List_Info.delimiter (field in core), 101
- List_Info.start (field in core), 101
- List_Info.style (field in core), 100
- list_items (procedure in readers/commonmark), 305
- list_packages (procedure in odindoc), 255
- List_Style (enum in core), 101
- List_Style.Arabic (enumerator in core), 101
- List_Style.Bullet (enumerator in core), 101
- List_Style.Lower_Alpha (enumerator in core), 101

List_Style.Lower_Roman (enumerator in core), 101
 List_Style.Upper_Alpha (enumerator in core), 101
 List_Style.Upper_Roman (enumerator in core), 101
 LIST_STYLE_NAMES (variable in core), 142
 load (procedure in gds), 149
 (procedure in inventory), 233
 (procedure in odindoc), 256
 (procedure in project), 279
 load_mo (procedure in gettext), 163
 load_object (procedure in core), 133
 Load_Options (struct in project), 272
 Load_Options.conf_dir (field in project), 272
 Load_Options.no_config (field in project), 272
 Load_Options.output (field in project), 272
 Load_Options.overrides (field in project), 272
 Load_Options.source (field in project), 272
 load_po (procedure in gettext), 163
 load_template (procedure in jinja), 246
 Loader (struct in jinja), 238
 Loader.destroy (field in jinja), 238
 Loader.procedure (field in jinja), 238
 Loader.user (field in jinja), 238
 Loader_Proc (type in jinja), 238
 locate (procedure in typst), 348
 lock (procedure in gds), 149
 Look (struct in highlight), 178
 Look.background (field in highlight), 178
 Look.bold (field in highlight), 178
 Look.border (field in highlight), 178
 Look.color (field in highlight), 178
 Look.italic (field in highlight), 178
 Look.underline (field in highlight), 178
 lookup_name (procedure in pyscan), 293
 Loop (struct in jinja), 238
 Loop.changed (field in jinja), 239
 Loop.depth0 (field in jinja), 238
 Loop.index0 (field in jinja), 238
 Loop.items (field in jinja), 238
 Loop.last_seen (field in jinja), 238
 Loop.recurse (field in jinja), 239

M

main_stack_bytes (procedure in host), 215
 make_catalog (procedure in gettext), 163
 make_id (procedure in core), 133
 make_storage (procedure in host), 215
 make_template (procedure in i18n), 230
 make_translator (procedure in gettext), 163
 Map_Entry (struct in core), 101
 Map_Entry.offset (field in core), 101
 Map_Entry.span (field in core), 101
 mark (procedure in readers/rst), 321
 Markdown (struct in readers/sphinx), 334
 Markdown.config (field in readers/sphinx), 335
 Markdown.ext (field in readers/sphinx), 335
 Markdown.host (field in readers/sphinx), 335

Markdown.parser (field in readers/sphinx), 335
 Markdown.reader (field in readers/sphinx), 335
 Markdown.settings (field in readers/sphinx), 335
 Markdown.state (field in readers/sphinx), 335
 markdown_settings (procedure in readers/sphinx), 336
 markup (procedure in jinja), 246
 Markup (type in jinja), 239
 Math_Style (enum in core), 101
 Math_Style.Plain (enumerator in core), 101
 Math_Style.Sphinx (enumerator in core), 101
 MAX_OBJECT_TYPES (constant in readers/sphinx), 337
 MAX_RETRIES (constant in host), 222
 MAX_SOURCE_BYTES (constant in host), 222
 Media_Kind (enum in core), 102
 media_kind (procedure in core), 133
 Media_Kind.Audio (enumerator in core), 102
 Media_Kind.Image (enumerator in core), 102
 Media_Kind.Video (enumerator in core), 102
 Member (struct in odindoc), 251
 Member.docs (field in odindoc), 252
 Member.kind (field in odindoc), 251
 Member.name (field in odindoc), 251
 Member.signature (field in odindoc), 251
 Member_Kind (enum in odindoc), 252
 Member_Kind.Bit_Field (enumerator in odindoc), 252
 Member_Kind.Enumerator (enumerator in odindoc), 252
 Member_Kind.Field (enumerator in odindoc), 252
 MEMORY_HINT (constant in host), 222
 memory_loader (procedure in jinja), 246
 merge (procedure in gettext), 164
 Merge_Options (struct in gettext), 155
 Merge_Options.fuzzy_matching (field in gettext), 155
 Merge_Options.keep_obsolete (field in gettext), 155
 Merge_Options.stats (field in gettext), 155
 Merge_Stats (struct in gettext), 155
 Merge_Stats.bounds (field in gettext), 155
 Merge_Stats.comparisons (field in gettext), 155
 Merge_Stats.cut_short (field in gettext), 155
 Merge_Stats.searches (field in gettext), 155
 Merge_Stats.work (field in gettext), 155
 Message (struct in autodoc), 84
 (struct in core), 102
 (struct in gettext), 155
 (struct in odindoc), 252
 Message.code (field in autodoc), 84
 (field in core), 102
 (field in odindoc), 252

- Message.comments (field in gettext), 156
 - Message.ctx (field in gettext), 155
 - Message.flags (field in gettext), 156
 - Message.fuzzy (field in gettext), 156
 - Message.has_ctx (field in gettext), 155
 - Message.has_plural (field in gettext), 156
 - Message.has_previous (field in gettext), 156
 - Message.hint (field in autodoc), 84
 - (field in core), 102
 - (field in odindoc), 252
 - Message.id (field in gettext), 156
 - Message.id_plural (field in gettext), 156
 - Message.line (field in gettext), 156
 - Message.message (field in core), 102
 - Message.notes (field in gettext), 156
 - Message.obsolete (field in gettext), 156
 - Message.previous (field in gettext), 156
 - Message.references (field in gettext), 156
 - Message.severity (field in autodoc), 84
 - (field in core), 102
 - (field in odindoc), 252
 - Message.status (field in core), 102
 - Message.strs (field in gettext), 156
 - Message.text (field in autodoc), 84
 - (field in odindoc), 252
 - Message.title (field in autodoc), 84
 - (field in core), 102
 - (field in odindoc), 252
 - Meta (struct in readers/commonmark), 299
 - Meta.content (field in readers/commonmark), 299
 - Meta.key (field in readers/commonmark), 299
 - Meter (struct in host), 203
 - Meter.backing (field in host), 203
 - Meter.ledger (field in host), 203
 - Meter.prepaid (field in host), 203
 - meter_allocator (procedure in host), 215
 - meter_prepay (procedure in host), 215
 - meter_refund (procedure in host), 215
 - metered_refusal (procedure in host), 215
 - migrate (procedure in project), 279
 - Migrate_Options (struct in project), 272
 - Migrate_Options.force (field in project), 272
 - Migrate_Options.input (field in project), 272
 - Migrate_Options.output (field in project), 272
 - Migrate_Options.year (field in project), 273
 - Migrate_Result (struct in project), 273
 - Migrate_Result.attention (field in project), 273
 - Migrate_Result.converted (field in project), 273
 - Migrate_Result.input (field in project), 273
 - Migrate_Result.notes (field in project), 273
 - Migrate_Result.output (field in project), 273
 - Migrate_Result.toml (field in project), 273
 - module (procedure in pyscan), 293
 - Module (struct in pyscan), 285
 - Module.all (field in pyscan), 286
 - Module.broken (field in pyscan), 286
 - Module.doc (field in pyscan), 286
 - Module.info (field in pyscan), 286
 - Module.kind (field in pyscan), 285
 - Module.name (field in pyscan), 285
 - Module.origin (field in pyscan), 286
 - Module.path (field in pyscan), 286
 - Module.project (field in pyscan), 285
 - Module.root (field in pyscan), 286
 - Module.scope (field in pyscan), 286
 - Module.source (field in pyscan), 286
 - Module.too_deep (field in pyscan), 286
 - Module.tree (field in pyscan), 286
 - module_attr (procedure in pyscan), 293
 - Module_Kind (enum in pyscan), 286
 - Module_Kind.Namespace (enumerator in pyscan), 287
 - Module_Kind.Native (enumerator in pyscan), 287
 - Module_Kind.Package (enumerator in pyscan), 286
 - Module_Kind.Source (enumerator in pyscan), 286
 - module_object (procedure in pyscan), 293
 - mro (procedure in pyscan), 293
 - Mro_Entry (struct in pyscan), 287
 - Mro_Entry.class (field in pyscan), 287
 - Mro_Entry.path (field in pyscan), 287
 - MYST_CONFORMANCE (constant in readers/commonmark), 307
 - myst_options (procedure in defaults), 143
 - myst_reader (procedure in readers/commonmark), 305
 - MYST_VERSION (constant in readers/commonmark), 307
- ## N
- named_child (procedure in treesitter), 343
 - named_child_count (procedure in treesitter), 343
 - Native (type in jinja), 239
 - new_message (procedure in gettext), 164
 - new_plan (procedure in gds), 149
 - next_anchor_id (procedure in core), 133
 - next_attribute (procedure in core), 134
 - next_child (procedure in core), 134
 - (procedure in treesitter), 343
 - next_named_sibling (procedure in treesitter), 343
 - next_number (procedure in gds), 150
 - next_sibling (procedure in treesitter), 343
 - next_token (procedure in highlight), 192
 - nfd_first (procedure in core), 134
 - no_reader_settings (procedure in core), 134
 - node (procedure in core), 134
 - Node (struct in core), 102
 - (struct in treesitter), 340
 - Node.attributes (field in core), 103
 - Node.ctx (field in treesitter), 340
 - Node.flags (field in core), 102
 - Node.id (field in treesitter), 340
 - Node.kind (field in core), 102
 - Node.payload (field in core), 103
 - Node.span (field in core), 103
 - Node.subtree_end (field in core), 102
 - Node.text (field in core), 102
 - Node.tree (field in treesitter), 340
 - Node.value (field in core), 102
 - node_count (procedure in core), 134
 - Node_Flag (enum in core), 103

- Node_Flag.Anonymous (enumerator in core), 103
- Node_Flag.Auto (enumerator in core), 103
- Node_Flag.Generated (enumerator in core), 103
- Node_Flag.Header (enumerator in core), 103
- Node_Flag.Local (enumerator in core), 103
- Node_Flag.Referenced (enumerator in core), 103
- Node_Flag.Symbol (enumerator in core), 103
- Node_Flag.Tight (enumerator in core), 103
- Node_Flags (bit set in core), 103
- Node_Id (type in core), 103
- Node_Kind (enum in core), 104
- Node_Kind.Abbreviation (enumerator in core), 108
- Node_Kind.Admonition (enumerator in core), 106
- Node_Kind.Attribution (enumerator in core), 104
- Node_Kind.Block_Quote (enumerator in core), 104
- Node_Kind.Caption (enumerator in core), 106
- Node_Kind.Cell (enumerator in core), 106
- Node_Kind.Citation (enumerator in core), 106
- Node_Kind.Citation_Reference (enumerator in core), 107
- Node_Kind.Classifier (enumerator in core), 104
- Node_Kind.Code (enumerator in core), 107
- Node_Kind.Code_Block (enumerator in core), 105
- Node_Kind.Comment (enumerator in core), 106
- Node_Kind.Compound (enumerator in core), 106
- Node_Kind.Container (enumerator in core), 106
- Node_Kind.Contents (enumerator in core), 106
- Node_Kind.Definition (enumerator in core), 104
- Node_Kind.Definition_Item (enumerator in core), 104
- Node_Kind.Definition_List (enumerator in core), 104
- Node_Kind.Desc (enumerator in core), 107
- Node_Kind.Desc_Addname (enumerator in core), 108
- Node_Kind.Desc_Annotation (enumerator in core), 108
- Node_Kind.Desc_Content (enumerator in core), 107
- Node_Kind.Desc_Name (enumerator in core), 108
- Node_Kind.Desc_Optional (enumerator in core), 108
- Node_Kind.Desc_Parameter (enumerator in core), 108
- Node_Kind.Desc_Parameterlist (enumerator in core), 108
- Node_Kind.Desc_Returns (enumerator in core), 108
- Node_Kind.Desc_Signature (enumerator in core), 107
- Node_Kind.Directive (enumerator in core), 106
- Node_Kind.Doctest_Block (enumerator in core), 105
- Node_Kind.Document (enumerator in core), 104
- Node_Kind.Emphasis (enumerator in core), 107
- Node_Kind.Field (enumerator in core), 104
- Node_Kind.Field_Body (enumerator in core), 105
- Node_Kind.Field_List (enumerator in core), 104
- Node_Kind.Field_Name (enumerator in core), 104
- Node_Kind.Figure (enumerator in core), 106
- Node_Kind.Footnote (enumerator in core), 106
- Node_Kind.Footnote_Reference (enumerator in core), 107
- Node_Kind.Glossary (enumerator in core), 107
- Node_Kind.Hard_Break (enumerator in core), 108
- Node_Kind.Heading (enumerator in core), 104
- Node_Kind.Hlist (enumerator in core), 107
- Node_Kind.Image (enumerator in core), 107
- Node_Kind.Index (enumerator in core), 107
- Node_Kind.Inline_Target (enumerator in core), 108
- Node_Kind.Legend (enumerator in core), 106
- Node_Kind.Line (enumerator in core), 105
- Node_Kind.Line_Block (enumerator in core), 105
- Node_Kind.Link (enumerator in core), 107
- Node_Kind.List (enumerator in core), 104
- Node_Kind.List_Item (enumerator in core), 104
- Node_Kind.Math (enumerator in core), 108
- Node_Kind.Math_Block (enumerator in core), 105
- Node_Kind.Only (enumerator in core), 107
- Node_Kind.Option (enumerator in core), 105
- Node_Kind.Option_Argument (enumerator in core), 105
- Node_Kind.Option_Description (enumerator in core), 105

Node_Kind.Option_Group (enumerator in core), 105
 Node_Kind.Option_Item (enumerator in core), 105
 Node_Kind.Option_List (enumerator in core), 105
 Node_Kind.Paragraph (enumerator in core), 104
 Node_Kind.Preformatted (enumerator in core), 105
 Node_Kind.Production (enumerator in core), 107
 Node_Kind.Production_List (enumerator in core), 107
 Node_Kind.Raw_Block (enumerator in core), 105
 Node_Kind.Raw_Inline (enumerator in core), 108
 Node_Kind.Reference (enumerator in core), 107
 Node_Kind.Role (enumerator in core), 108
 Node_Kind.Row (enumerator in core), 105
 Node_Kind.Rubric (enumerator in core), 106
 Node_Kind.Section (enumerator in core), 104
 Node_Kind.Sidebar (enumerator in core), 106
 Node_Kind.Soft_Break (enumerator in core), 108
 Node_Kind.Strong (enumerator in core), 107
 Node_Kind.Subscript (enumerator in core), 108
 Node_Kind.Substitution_Definition (enumerator in core), 106
 Node_Kind.Substitution_Reference (enumerator in core), 108
 Node_Kind.Subtitle (enumerator in core), 104
 Node_Kind.Superscript (enumerator in core), 108
 Node_Kind.Table (enumerator in core), 105
 Node_Kind.Table_Body (enumerator in core), 105
 Node_Kind.Table_Head (enumerator in core), 105
 Node_Kind.Target (enumerator in core), 106
 Node_Kind.Term (enumerator in core), 104
 Node_Kind.Text (enumerator in core), 107
 Node_Kind.Thematic_Break (enumerator in core), 105
 Node_Kind.Title (enumerator in core), 104
 Node_Kind.Title_Reference (enumerator in core), 108
 Node_Kind.Toctree (enumerator in core), 106
 Node_Kind.Toctree_Entry (enumerator in core), 106
 Node_Kind.Topic (enumerator in core), 106

Node_Kind.Version_Modified (enumerator in core), 107
 Node_Kind.Xref (enumerator in core), 108
 node_text (procedure in core), 134
 NONE (constant in core), 142
 none (procedure in jinja), 247
 None (struct in jinja), 239
 normalize_label (procedure in core), 134
 normalize_nfd_into (procedure in core), 134
 NOTICES (constant in treesitter), 345
 now (procedure in gds), 150
 number_repr (procedure in pyscan), 293

O

Object (struct in pyscan), 287
 Object.abstract (field in pyscan), 288
 Object.class (field in pyscan), 287
 Object.class_property (field in pyscan), 288
 Object.decorator (field in pyscan), 288
 Object.def (field in pyscan), 288
 Object.doc (field in pyscan), 288
 Object.function (field in pyscan), 287
 Object.kind (field in pyscan), 287
 Object.module (field in pyscan), 287
 Object.owner (field in pyscan), 288
 Object.partial (field in pyscan), 288
 Object.partial_args (field in pyscan), 288
 Object.partial_method (field in pyscan), 288
 Object.path (field in pyscan), 287
 Object.tuple (field in pyscan), 287
 Object.value (field in pyscan), 287
 Object.wrapper (field in pyscan), 287
 Object_Kind (enum in pyscan), 288
 Object_Kind.Builtin (enumerator in pyscan), 289
 Object_Kind.Class (enumerator in pyscan), 288
 Object_Kind.External (enumerator in pyscan), 289
 Object_Kind.Function (enumerator in pyscan), 288
 Object_Kind.Module (enumerator in pyscan), 288
 Object_Kind.None (enumerator in pyscan), 288
 Object_Kind.Value (enumerator in pyscan), 289
 Object_Type (struct in readers/sphinx), 335
 Object_Type.directive (field in readers/sphinx), 335
 Object_Type.display (field in readers/sphinx), 335
 Object_Type.first_word (field in readers/sphinx), 335
 Object_Type.index (field in readers/sphinx), 335
 Object_Type.program (field in readers/sphinx), 336
 Object_Type.role (field in readers/sphinx), 335
 Object_Type.target (field in readers/sphinx), 335
 odindoc (package), 248

- offset_in (procedure in core), 135
- offset_of (procedure in typst), 349
- On_Refusal (type in host), 203
- open (procedure in readers/rst), 321
- Opt_Spec (struct in readers/rst), 313
- Opt_Spec.choices (field in readers/rst), 313
- Opt_Spec.name (field in readers/rst), 313
- Opt_Spec.type (field in readers/rst), 313
- Opt_Type (enum in readers/rst), 313
- Opt_Type.Char (enumerator in readers/rst), 314
- Opt_Type.Choice (enumerator in readers/rst), 314
- Opt_Type.Class (enumerator in readers/rst), 314
- Opt_Type.Encoding (enumerator in readers/rst), 314
- Opt_Type.Figwidth (enumerator in readers/rst), 314
- Opt_Type.Flag (enumerator in readers/rst), 313
- Opt_Type.Int (enumerator in readers/rst), 313
- Opt_Type.Length (enumerator in readers/rst), 314
- Opt_Type.Length_Percent (enumerator in readers/rst), 314
- Opt_Type.Name (enumerator in readers/rst), 314
- Opt_Type.Nonneg_Int (enumerator in readers/rst), 313
- Opt_Type.Opt_Int (enumerator in readers/rst), 313
- Opt_Type.Path (enumerator in readers/rst), 314
- Opt_Type.Percentage (enumerator in readers/rst), 314
- Opt_Type.Pos_Int (enumerator in readers/rst), 313
- Opt_Type.Text (enumerator in readers/rst), 313
- Opt_Type.Text_Required (enumerator in readers/rst), 313
- Opt_Type.Widths (enumerator in readers/rst), 314
- opt_value (procedure in readers/rst), 321
- Option (struct in autodoc), 85
 - (struct in odindoc), 252
 - (struct in readers/commonmark), 299
- Option.name (field in autodoc), 85
 - (field in odindoc), 252
 - (field in readers/commonmark), 299
- Option.span (field in readers/commonmark), 299
- Option.value (field in autodoc), 85
 - (field in odindoc), 252
 - (field in readers/commonmark), 299
- option_lines (procedure in readers/rst), 321
- Options (struct in fetch), 144
 - (struct in gds), 146
 - (struct in graphdog), 171
 - (struct in jinja), 239
 - (struct in readers/rst/inline), 325
 - (struct in typst), 347
- Options.attributes (field in graphdog), 171
- Options.author (field in gds), 146
- Options.autoescape (field in jinja), 239
- Options.autoescape_extensions (field in jinja), 239
- Options.character_level (field in readers/rst/inline), 326
- Options.check_only (field in gds), 147
- Options.command (field in gds), 146
- Options.default_substitution (field in readers/rst/inline), 326
- Options.dir (field in gds), 146
- Options.discussion (field in gds), 146
- Options.docinfo (field in readers/rst/inline), 326
- Options.dry_run (field in gds), 147
- Options.embedded_fonts_only (field in typst), 347
- Options.engine (field in graphdog), 171
- Options.evidence (field in gds), 146
- Options.files (field in graphdog), 171
- Options.font_paths (field in typst), 347
- Options.format (field in graphdog), 171
- Options.historical (field in gds), 147
- Options.identifier (field in gds), 146
- Options.keep_trailing_newline (field in jinja), 239
- Options.label (field in gds), 146
- Options.labels (field in gds), 146
- Options.language (field in readers/rst/inline), 326
- Options.loader (field in jinja), 239
- Options.lstrip_blocks (field in jinja), 239
- Options.max_bytes (field in fetch), 144
- Options.max_iterations (field in jinja), 240
- Options.max_nesting (field in jinja), 240
- Options.max_output (field in jinja), 240
- Options.max_recursion (field in jinja), 239
- Options.number (field in gds), 147
- Options.offline (field in typst), 347
- Options.package_cache (field in typst), 347
- Options.pep_base_url (field in readers/rst/inline), 326
- Options.pep_references (field in readers/rst/inline), 325
- Options.random_seed (field in jinja), 240
- Options.reason (field in gds), 146
- Options.record (field in gds), 146
- Options.render (field in gds), 147
- Options.reserve (field in graphdog), 171
 - (field in typst), 347
- Options.rfc_base_url (field in readers/rst/inline), 326
- Options.rfc_references (field in readers/rst/inline), 326
- Options.rollback (field in gds), 147
- Options.sectsubtitle (field in readers/rst/inline), 326
- Options.slug (field in gds), 146
- Options.smart_quotes (field in readers/rst/inline), 326
- Options.smart_quotes_locales (field in readers/rst/inline), 326
- Options.stack_bytes (field in jinja), 240
- Options.state (field in gds), 146
- Options.tab_width (field in readers/rst/inline), 326

Options.target (field in gds), 147
 Options.timeout_ms (field in fetch), 144
 Options.to (field in gds), 146
 Options.trim_blocks (field in jinja), 239
 Options.trim_footnote_reference_space
 (field in readers/rst/inline), 326
 Options.undefined (field in jinja), 239
 Options.user_agent (field in fetch), 144
 Options.warnings (field in graphdog), 171
 OS (type in odindoc), 252
 out_of_memory (procedure in host), 216
 Outcome (struct in typst), 347
 Outcome.detail (field in typst), 347
 Outcome.diagnostics (field in typst), 347
 Outcome.exit_code (field in typst), 347
 Outcome.launch_error (field in typst), 347
 Outcome.ok (field in typst), 347
 Output (struct in highlight), 178
 Output.buf (field in highlight), 178
 Output.written (field in highlight), 178
 output_fits (procedure in highlight), 192
 output_string (procedure in highlight), 192
 output_text (procedure in host), 216

P

Package (struct in odindoc), 252
 Package.decls (field in odindoc), 253
 Package.dir (field in odindoc), 253
 Package.doc (field in odindoc), 253
 Package.errors (field in odindoc), 253
 Package.files (field in odindoc), 253
 Package.id (field in odindoc), 253
 Package.imports (field in odindoc), 253
 Package.name (field in odindoc), 253
 Package.platform (field in odindoc), 253
 Package.synopsis (field in odindoc), 253
 Package.targets (field in odindoc), 253
 Package.too_deep (field in odindoc), 253
 Package.too_large (field in odindoc), 253
 Package_Ref (struct in odindoc), 253
 Package_Ref.dir (field in odindoc), 253
 Package_Ref.id (field in odindoc), 253
 Param (struct in pyscan), 289
 Param.annotation (field in pyscan), 289
 Param.default (field in pyscan), 289
 Param.kind (field in pyscan), 289
 Param.name (field in pyscan), 289
 Param.type_comment (field in pyscan), 289
 Param_Kind (enum in pyscan), 289
 Param_Kind.Keyword_Only (enumerator
 in pyscan), 289
 Param_Kind.Positional_Only (enumerator
 in pyscan), 289
 Param_Kind.Positional_Or_Keyword
 (enum in pyscan), 289
 Param_Kind.Var_Keyword (enumerator in
 pyscan), 289
 Param_Kind.Var_Positional (enumerator in
 pyscan), 289
 parent (procedure in treesitter), 343
 parent_of (procedure in core), 135
 parse (procedure in treesitter), 343
 PARSE_BASE_BYTES (constant in
 odindoc), 256
 parse_blocks (procedure in readers/rst),
 322
 parse_body (procedure in readers/
 commonmark), 306
 (procedure in readers/rst), 322
 parse_bound (procedure in odindoc), 256
 PARSE_BYTES_PER_BYTE (constant in
 odindoc), 256
 parse_engine (procedure in graphdog), 173
 parse_expression (procedure in pyscan),
 294
 parse_format (procedure in graphdog), 173
 parse_inline (procedure in readers/
 commonmark), 306
 (procedure in readers/rst/inline), 329
 parse_mo (procedure in gettext), 164
 parse_plural_forms (procedure in gettext),
 164
 parse_po (procedure in gettext), 164
 parse_record (procedure in gds), 150
 parse_source (procedure in readers/rst),
 322
 parse_target (procedure in i18n), 230
 parse_template (procedure in jinja), 247
 parse_titled (procedure in readers/rst), 322
 Parser (struct in readers/rst), 314
 (struct in treesitter), 340
 Parser.b (field in readers/rst), 314
 Parser.ctx (field in readers/rst), 314
 Parser.depth (field in readers/rst), 314
 Parser.ext (field in readers/rst), 315
 Parser.has_pending (field in readers/rst),
 315
 Parser.host (field in readers/rst), 316
 Parser.hosted (field in readers/rst), 316
 Parser.in_sidebar (field in readers/rst), 315
 Parser.inl (field in readers/rst), 314
 Parser.level_base (field in readers/rst), 315
 Parser.pending (field in readers/rst), 315
 Parser.pending_span (field in readers/rst),
 315
 Parser.section_base (field in readers/rst),
 315
 Parser.sections (field in readers/rst), 315
 Parser.sections_used (field in readers/rst),
 315
 Parser.src (field in readers/rst), 314
 Parser.stack (field in readers/rst), 315
 Parser.styles (field in readers/rst), 315
 Parser.styles_used (field in readers/rst),
 315
 Parser.tab (field in readers/rst), 314
 Parser.table_extra (field in readers/rst), 316
 Parser.text (field in readers/rst), 314
 Parser.top_count (field in readers/rst), 315
 Parser.top_transition (field in readers/rst),
 315
 Parser.transition_span (field in readers/
 rst), 315
 parser_delete (procedure in treesitter), 344
 parser_new (procedure in treesitter), 344
 Pass (struct in core), 109
 Pass.name (field in core), 109
 Pass.procedure (field in core), 109
 Pass.user (field in core), 109
 Pass_Outcome (enum in core), 109
 Pass_Outcome.Failed (enumerator in core),
 109

- Pass_Outcome.Rebuilt (enumerator in core), 109
- Pass_Outcome.Unchanged (enumerator in core), 109
- PATH_BYTES (constant in host), 222
- path_of (procedure in readers/rst), 322
- path_type (procedure in host), 216
- Pdf_Document (struct in project), 273
- Pdf_Document.author (field in project), 273
- Pdf_Document.file (field in project), 273
- Pdf_Document.root (field in project), 273
- Pdf_Document.title (field in project), 273
- pinned (procedure in typst), 349
- PINNED_VERSION (constant in typst), 349
- place_titled (procedure in readers/rst), 322
- plain_text (procedure in core), 135
- Plan (struct in gds), 147
- Plan.changes (field in gds), 147
- Plan.summary (field in gds), 147
- plan_bytes (procedure in core), 135
- plan_session (procedure in host), 216
- plural (procedure group in gettext), 166
- plural_forms_for (procedure in gettext), 164
- plural_index (procedure in gettext), 165
- Plural_Node (struct in gettext), 156
- Plural_Node.a (field in gettext), 156
- Plural_Node.b (field in gettext), 157
- Plural_Node.c (field in gettext), 157
- Plural_Node.op (field in gettext), 156
- Plural_Node.value (field in gettext), 157
- Plural_Op (enum in gettext), 157
- Plural_Op.Add (enumerator in gettext), 157
- Plural_Op.And (enumerator in gettext), 157
- Plural_Op.Choose (enumerator in gettext), 158
- Plural_Op.Div (enumerator in gettext), 157
- Plural_Op.Equal (enumerator in gettext), 157
- Plural_Op.Greater (enumerator in gettext), 157
- Plural_Op.Greater_Equal (enumerator in gettext), 157
- Plural_Op.Less (enumerator in gettext), 157
- Plural_Op.Less_Equal (enumerator in gettext), 157
- Plural_Op.Mod (enumerator in gettext), 157
- Plural_Op.Mul (enumerator in gettext), 157
- Plural_Op.N (enumerator in gettext), 157
- Plural_Op.Not (enumerator in gettext), 157
- Plural_Op.Not_Equal (enumerator in gettext), 157
- Plural_Op.Number (enumerator in gettext), 157
- Plural_Op.Or (enumerator in gettext), 157
- Plural_Op.Sub (enumerator in gettext), 157
- Plural_Problem (struct in gettext), 158
- Plural_Problem.column (field in gettext), 158
- Plural_Problem.message (field in gettext), 158
- Plural_Rule (struct in gettext), 158
- Plural_Rule.count (field in gettext), 158
- Plural_Rule.nodes (field in gettext), 158
- Plural_Rule.nplurals (field in gettext), 158
- Plural_Rule.root (field in gettext), 158
- plural_with_context (procedure group in gettext), 166
- Point (struct in treesitter), 340
- Point.column (field in treesitter), 340
- Point.row (field in treesitter), 340
- Policy (struct in core), 109
- Policy.includes (field in core), 109
- Policy.keep_documents_with_errors (field in core), 109
- Policy.math (field in core), 109
- Policy.raw (field in core), 109
- Policy.strict (field in core), 109
- Policy.unsafe_urls (field in core), 109
- pool_classes (procedure in readers/rst), 322
- pool_lines (procedure in readers/rst), 322
- pool_lower (procedure in readers/rst), 323
- pool_of (procedure in gettext), 165
- pool_string (procedure in core), 135
- pop (procedure in core), 135
- Pot_Header (struct in i18n), 224
- Pot_Header.copyright (field in i18n), 224
- Pot_Header.creation_date (field in i18n), 225
- Pot_Header.language_team (field in i18n), 224
- Pot_Header.last_translator (field in i18n), 224
- Pot_Header.project (field in i18n), 224
- Pot_Header.version (field in i18n), 224
- prepare_docstring (procedure in pyscan), 294
- prepare_session (procedure in host), 216
- prev_named_sibling (procedure in treesitter), 344
- prev_sibling (procedure in treesitter), 344
- Previous (struct in gettext), 158
- Previous.ctx (field in gettext), 158
- Previous.has_ctx (field in gettext), 158
- Previous.has_plural (field in gettext), 158
- Previous.id (field in gettext), 158
- Previous.id_plural (field in gettext), 158
- Private (enum in odindoc), 254
- Private.File (enumerator in odindoc), 254
- Private.Package (enumerator in odindoc), 254
- Private.Public (enumerator in odindoc), 254
- problem (procedure in gds), 150
- Problem (struct in gds), 147
 - (struct in highlight), 179
 - (struct in host), 204
- Problem.code (field in gds), 147
 - (field in host), 204
- Problem.column (field in highlight), 179
- Problem.excerpt (field in highlight), 179
- Problem.exit (field in gds), 147
 - (field in host), 204
- Problem.hint (field in gds), 147
 - (field in host), 204
- Problem.kind (field in highlight), 179

Problem.lexer (field in highlight), 179
 Problem.line (field in highlight), 179
 Problem.message (field in gds), 147
 (field in host), 204
 Problem.name (field in highlight), 179
 Problem.near (field in highlight), 179
 Problem.path (field in gds), 147
 (field in host), 204
 Problem.suggestion (field in highlight), 179
 Problem.title (field in gds), 147
 (field in host), 204
 Problem_Kind (enum in highlight), 179
 Problem_Kind.Lexing_Error (enumerator
 in highlight), 179
 Problem_Kind.None (enumerator in
 highlight), 179
 Problem_Kind.Unknown_Language
 (enumerator in highlight), 179
 Problem_Kind.Unknown_Style
 (enumerator in highlight), 179
 Progress (enum in i18n), 225
 Progress.Translated (enumerator in i18n),
 225
 Progress.Untranslated (enumerator in
 i18n), 225
 Progress_Classes (bit set in i18n), 225
 project (package), 256
 Project (struct in odindoc), 254
 (struct in project), 274
 (struct in pyscan), 289
 Project.config (field in project), 274
 Project.files (field in odindoc), 254
 (field in pyscan), 290
 Project.memory (field in project), 274
 Project.modules (field in pyscan), 290
 Project.mro_depth (field in pyscan), 290
 Project.owner (field in project), 274
 Project.packages (field in odindoc), 254
 Project.parser (field in pyscan), 290
 Project.reader (field in odindoc), 254
 (field in pyscan), 290
 Project.registry (field in project), 274
 Project.resolving (field in pyscan), 290
 Project.roots (field in odindoc), 254
 (field in pyscan), 290
 Project.settings (field in project), 274
 Project.stack_base (field in pyscan), 290
 Project.steps (field in pyscan), 290
 Project.trees (field in pyscan), 290
 Project.warnings (field in project), 274
 publish (procedure in host), 216
 push (procedure in core), 135
 push_lines (procedure in readers/rst), 323
 put (procedure in highlight), 192
 put_byte (procedure in highlight), 192
 put_html (procedure in highlight), 193
 pyscan (package), 280
 python (procedure in treesitter), 344
 python_space (procedure in readers/rst/
 inline), 329
 PYTHON_VERSION (constant in
 treesitter), 345

Q

quickstart (procedure in project), 279
 Quickstart_Options (struct in project), 275

Quickstart_Options.author (field in
 project), 275
 Quickstart_Options.date (field in project),
 275
 Quickstart_Options.dir (field in project),
 275
 Quickstart_Options.language (field in
 project), 275
 Quickstart_Options.project (field in
 project), 275
 Quickstart_Options.release (field in
 project), 275
 Quickstart_Options.root_doc (field in
 project), 275
 Quickstart_Options.separate (field in
 project), 275
 Quickstart_Options.suffix (field in project),
 275
 Quickstart_Options.version (field in
 project), 275
 Quickstart_Options.year (field in project),
 275

R

read (procedure in host), 216
 (procedure in readers/commonmark),
 306
 (procedure in readers/rst), 323
 (procedure in readers/sphinx), 336
 Read_Context (struct in core), 110
 Read_Context.b (field in core), 110
 Read_Context.included (field in core), 110
 Read_Context.limits (field in core), 110
 Read_Context.options (field in core), 110
 Read_Context.policy (field in core), 110
 Read_Context.reports (field in core), 110
 Read_Context.resources (field in core), 110
 Read_Context.scratch (field in core), 110
 Read_Context.source (field in core), 110
 Read_Context.ws (field in core), 110
 read_document (procedure in core), 135
 read_extension (procedure in core), 135
 Read_Extension (struct in core), 110
 Read_Extension.data (field in core), 110
 Read_Extension.id (field in core), 111
 read_file (procedure in host), 217
 read_inline (procedure in readers/
 commonmark), 306
 (procedure in readers/rst), 323
 (procedure in readers/sphinx), 336
 Read_Inline_Proc (type in core), 111
 read_kept (procedure in host), 217
 read_myst (procedure in readers/
 commonmark), 306
 read_myst_inline (procedure in readers/
 commonmark), 306
 Read_Options (struct in core), 111
 Read_Options.extension (field in core), 111
 Read_Options.promote_title (field in core),
 111
 Read_Options.tab_width (field in core),
 111
 Read_Options.today (field in core), 111
 read_owned (procedure in host), 217
 Read_Proc (type in core), 111

- Read_Result (struct in core), 111
- Read_Result.document (field in core), 111
- Read_Result.using (field in core), 111
- read_source (procedure in host), 217
- reader (procedure in readers/
 - commonmark), 306
 - (procedure in readers/rst), 323
 - (procedure in readers/sphinx), 337
- Reader (struct in core), 111
- Reader.conformance (field in core), 112
- Reader.id (field in core), 111
- Reader.name (field in core), 111
- Reader.read (field in core), 112
- Reader.read_inline (field in core), 112
- Reader.version (field in core), 111
- reader_settings (procedure in core), 136
- readers/commonmark (package), 295
- readers/rst (package), 307
- readers/rst/inline (package), 325
- readers/sphinx (package), 330
- Record (struct in gds), 147
- Record.authors (field in gds), 148
- Record.created (field in gds), 148
- Record.discussion (field in gds), 148
- Record.fields (field in gds), 148
- Record.historical (field in gds), 148
- Record.kind (field in gds), 148
- Record.labels (field in gds), 148
- Record.number (field in gds), 147
- Record.path (field in gds), 148
- Record.slug (field in gds), 147
- Record.state (field in gds), 148
- Record.text (field in gds), 148
- Record.title (field in gds), 148
- Record.updated (field in gds), 148
- recover (procedure in gds), 150
- reference (procedure in pyscan), 294
- Reference_Style (enum in gettext), 159
- Reference_Style.None (enumerator in
 - gettext), 159
- Reference_Style.One_Per_Line
 - (enumerator in gettext), 159
- Reference_Style.Wrapped (enumerator in
 - gettext), 159
- Refusal (struct in host), 204
- Refusal.disk (field in host), 204
- Refusal.limit (field in host), 204
- Refusal.need (field in host), 204
- Refusal.place (field in host), 204
- Refusal.used (field in host), 204
- Refusal_Choice (enum in host), 204
- Refusal_Choice.Disk (enumerator in host),
 - 205
- Refusal_Choice.Raise (enumerator in host),
 - 205
- Refusal_Choice.Stop (enumerator in host),
 - 204
- Refusal_Decision (struct in host), 205
- Refusal_Decision.choice (field in host), 205
- Refusal_Decision.limit (field in host), 205
- Region (enum in core), 112
- Region.Attributes (enumerator in core),
 - 112
- Region.Cells (enumerator in core), 112
- Region.Links (enumerator in core), 112
- Region.Lists (enumerator in core), 112
- Region.Nodes (enumerator in core), 112
- Region.None (enumerator in core), 112
- Region.Output (enumerator in core), 112
- Region.Reports (enumerator in core), 112
- Region.Scratch (enumerator in core), 112
- Region.Second_Bank (enumerator in core),
 - 112
- Region.Source_Map (enumerator in core),
 - 112
- Region.Sources (enumerator in core), 112
- Region.Text (enumerator in core), 112
- registry (procedure in defaults), 143
- Registry (struct in core), 112
- Registry.readers (field in core), 112
- Registry.renderers (field in core), 113
- release (procedure in project), 279
 - (procedure in readers/rst), 323
- remove_path (procedure in host), 217
- remove_stages (procedure in host), 217
- rename_path (procedure in host), 218
- render (procedure in graphdog), 173
 - (procedure in jinja), 247
 - (procedure in render/html), 337
 - (procedure in render/text), 338
 - (procedure in render/typst), 339
- render/html (package), 337
- render/text (package), 338
- render/typst (package), 338
- Render_Context (struct in core), 113
- Render_Context.doc (field in core), 113
- Render_Context.limits (field in core), 113
- Render_Context.options (field in core), 113
- Render_Context.out (field in core), 113
- Render_Context.policy (field in core), 113
- Render_Context.reports (field in core), 113
- Render_Context.scratch (field in core), 113
- Render_Context.view (field in core), 113
- render_document (procedure in core), 136
- render_json (procedure in host), 218
- Render_Options (struct in core), 113
- Render_Options.anchor_prefix (field in
 - core), 114
- Render_Options.highlighter (field in core),
 - 114
- Render_Options.labels (field in core), 114
- Render_Options.language (field in core),
 - 113
- Render_Options.math (field in core), 114
- Render_Options.math_display (field in
 - core), 114
- Render_Options.math_inline (field in
 - core), 114
- Render_Options.permalink_icon (field in
 - core), 114
- Render_Options.permalinks (field in core),
 - 114
- Render_Options.reference_profile (field in
 - core), 113
- Render_Options.secnumber_suffix (field in
 - core), 114
- Render_Options.section_numbers (field in
 - core), 113
- Render_Options.standalone (field in core),
 - 113
- Render_Options.stylesheet (field in core),
 - 113
- Render_Options.title (field in core), 113
- render_position (procedure in jinja), 247

- render_problem_json (procedure in host), 218
- render_problem_text (procedure in host), 218
- Render_Proc (type in core), 114
- render_source (procedure in jinja), 247
- render_text (procedure in host), 218
- renderer (procedure in render/html), 337
 - (procedure in render/text), 338
 - (procedure in render/typst), 339
- Renderer (struct in core), 114
- Renderer.conformance (field in core), 115
- Renderer.id (field in core), 114
- Renderer.media_type (field in core), 114
- Renderer.name (field in core), 114
- Renderer.render (field in core), 115
- Renderer.version (field in core), 114
- replace_field (procedure in gds), 150
- report (procedure in core), 136
 - (procedure in readers/rst), 323
- Report (struct in host), 205
- Report.category (field in host), 206
- Report.code (field in host), 205
- Report.column (field in host), 205
- Report.end_column (field in host), 205
- Report.end_line (field in host), 205
- Report.excerpt (field in host), 205
- Report.hint (field in host), 205
- Report.line (field in host), 205
- Report.marker (field in host), 206
- Report.message (field in host), 205
- Report.path (field in host), 205
- Report.schema (field in host), 205
- Report.severity (field in host), 205
- Report.source (field in host), 206
- Report.title (field in host), 205
- report_message (procedure in core), 136
- Reports (struct in core), 115
- Reports.errors (field in core), 115
- Reports.fatal (field in core), 115
- Reports.has_primary (field in core), 115
- Reports.items (field in core), 115
- Reports.muted (field in core), 115
- Reports.primary (field in core), 115
- Reports.suppressed (field in core), 115
- Reports.truncated (field in core), 115
- Reports.used (field in core), 115
- Reports.warnings (field in core), 115
- repr (procedure in jinja), 247
- repr_bytes (procedure in pyscan), 294
- repr_str (procedure in pyscan), 294
- Request (struct in autodoc), 85
 - (struct in core), 115
 - (struct in odindoc), 254
- Request.argument (field in autodoc), 85
 - (field in odindoc), 254
- Request.cancel (field in core), 116
- Request.class (field in autodoc), 85
- Request.content (field in autodoc), 85
- Request.current (field in odindoc), 254
- Request.directive (field in autodoc), 85
 - (field in odindoc), 254
- Request.limits (field in core), 116
- Request.module (field in autodoc), 85
- Request.options (field in autodoc), 85
 - (field in odindoc), 254
- Request.passes (field in core), 116
- Request.policy (field in core), 116
- Request.read (field in core), 115
- Request.reader (field in core), 115
- Request.render (field in core), 115
- Request.renderer (field in core), 115
- Request.resources (field in core), 116
- Request.source (field in core), 115
- Request.source_map (field in core), 116
- Reserve (struct in graphdog), 171
 - (struct in typst), 348
- Reserve.procedure (field in graphdog), 171
 - (field in typst), 348
- Reserve.user (field in graphdog), 171
 - (field in typst), 348
- reset_workspace (procedure in core), 136
- resize_output (procedure in host), 218
- resize_source_map (procedure in host), 218
- resolve (procedure in gds), 150
 - (procedure in highlight), 193
 - (procedure in readers/rst/inline), 330
- resolve_link (procedure in core), 136
- resolve_style (procedure in highlight), 193
- Resource_Provider (struct in core), 116
- Resource_Provider.procedure (field in core), 116
- Resource_Provider.user (field in core), 116
- Resource_Status (enum in core), 116
- Resource_Status.Limit (enumerator in core), 116
- Resource_Status.Missing (enumerator in core), 116
- Resource_Status.Ok (enumerator in core), 116
- Resource_Status.Refused (enumerator in core), 116
- Result (struct in autodoc), 85
 - (struct in core), 116
 - (struct in fetch), 144
 - (struct in odindoc), 254
- Result.data (field in fetch), 144
- Result.error (field in fetch), 144
- Result.lines (field in autodoc), 85
 - (field in odindoc), 254
- Result.mapped (field in core), 117
- Result.message (field in fetch), 144
- Result.messages (field in autodoc), 85
 - (field in odindoc), 254
- Result.need (field in core), 117
- Result.primary (field in core), 117
- Result.reports (field in core), 117
- Result.status (field in core), 117
 - (field in fetch), 144
- Result.suppressed (field in core), 117
- Result.truncated (field in core), 117
- Result.url (field in fetch), 144
- Result.written (field in core), 117
- Role (struct in readers/commonmark), 299
 - (struct in readers/rst/inline), 326
- Role.base (field in readers/rst/inline), 326
- Role.classes (field in readers/rst/inline), 326
- Role.content (field in readers/commonmark), 300
- Role.format (field in readers/rst/inline), 326
- Role.language (field in readers/rst/inline), 327
- Role.lines (field in readers/commonmark), 300

- Role.name (field in readers/commonmark), 300
- Role.name (field in readers/rst/inline), 326
- Role.source (field in readers/commonmark), 300
- Role.span (field in readers/commonmark), 300
- Role_Base (enum in readers/rst/inline), 327
- Role_Base.Abbreviation (enumerator in readers/rst/inline), 327
- Role_Base.Acronym (enumerator in readers/rst/inline), 327
- Role_Base.Code (enumerator in readers/rst/inline), 327
- Role_Base.Custom (enumerator in readers/rst/inline), 327
- Role_Base.Emphasis (enumerator in readers/rst/inline), 327
- Role_Base.Literal (enumerator in readers/rst/inline), 327
- Role_Base.Math (enumerator in readers/rst/inline), 327
- Role_Base.Pep_Reference (enumerator in readers/rst/inline), 327
- Role_Base.Raw (enumerator in readers/rst/inline), 327
- Role_Base.Rfc_Reference (enumerator in readers/rst/inline), 327
- Role_Base.Strong (enumerator in readers/rst/inline), 327
- Role_Base.Subscript (enumerator in readers/rst/inline), 327
- Role_Base.Superscript (enumerator in readers/rst/inline), 327
- Role_Base.Title_Reference (enumerator in readers/rst/inline), 327
- Role_Call (struct in readers/rst/inline), 327
- Role_Call.b (field in readers/rst/inline), 328
- Role_Call.content (field in readers/rst/inline), 328
- Role_Call.marked (field in readers/rst/inline), 328
- Role_Call.name (field in readers/rst/inline), 328
- Role_Call.span (field in readers/rst/inline), 328
- Role_Call.state (field in readers/rst/inline), 327
- Role_Call.text (field in readers/rst/inline), 328
- Role_Handler (struct in readers/commonmark), 300
- Role_Handler (struct in readers/rst), 316
- Role_Handler.name (field in readers/commonmark), 300
- Role_Handler.name (field in readers/rst), 316
- Role_Handler.run (field in readers/commonmark), 300
- Role_Handler.run (field in readers/rst), 316
- Role_Hook (struct in readers/rst/inline), 328
- Role_Hook.procedure (field in readers/rst/inline), 328
- Role_Hook.user (field in readers/rst/inline), 328
- Role_Proc (type in readers/commonmark), 300
- Role_Proc (type in readers/rst), 316
- root (procedure in treesitter), 344
- Root (struct in odindoc), 255
- ROOT (type in core), 110
- Root.collection (field in odindoc), 255
- Root.dir (field in odindoc), 255
- rst_options (procedure in defaults), 143
- Rule (enum in readers/commonmark), 300
- Rule.Autolink (enumerator in readers/commonmark), 301
- Rule.Backticks (enumerator in readers/commonmark), 301
- Rule.Blockquote (enumerator in readers/commonmark), 300
- Rule.Code (enumerator in readers/commonmark), 300
- Rule.Emphasis (enumerator in readers/commonmark), 301
- Rule.Entity (enumerator in readers/commonmark), 301
- Rule.Escape (enumerator in readers/commonmark), 301
- Rule.Fence (enumerator in readers/commonmark), 300
- Rule.Footnote_Def (enumerator in readers/commonmark), 302
- Rule.Footnote_Ref (enumerator in readers/commonmark), 302
- Rule.Front_Matter (enumerator in readers/commonmark), 301
- Rule.Heading (enumerator in readers/commonmark), 301
- Rule.Hr (enumerator in readers/commonmark), 300
- Rule.Html_Block (enumerator in readers/commonmark), 301
- Rule.Html_Inline (enumerator in readers/commonmark), 301
- Rule.Image (enumerator in readers/commonmark), 301
- Rule.Lheading (enumerator in readers/commonmark), 301
- Rule.Link (enumerator in readers/commonmark), 301
- Rule.List (enumerator in readers/commonmark), 300
- Rule.Math_Block (enumerator in readers/commonmark), 301
- Rule.Math_Inline (enumerator in readers/commonmark), 301
- Rule.Myst_Block_Break (enumerator in readers/commonmark), 301
- Rule.Myst_Directive (enumerator in readers/commonmark), 302
- Rule.Myst_Line_Comment (enumerator in readers/commonmark), 301

Rule.Myst_Role (enumerator in readers/
commonmark), 301
Rule.Myst_Target (enumerator in readers/
commonmark), 301
Rule.Newline (enumerator in readers/
commonmark), 301
Rule.Reference (enumerator in readers/
commonmark), 300
Rule.Table (enumerator in readers/
commonmark), 300
run (procedure in cli), 87
(procedure in host), 218
run_hosted (procedure in readers/rst), 324
run_workers (procedure in host), 219
RUNTIME_VERSION (constant in
treesitter), 345

S

same_contents (procedure in host), 219
save_po (procedure in gettext), 165
Scanner (struct in highlight), 180
Scanner.code (field in highlight), 180
Scanner.count (field in highlight), 180
Scanner.head (field in highlight), 180
Scanner.inner (field in highlight), 180
Scanner.limit (field in highlight), 180
Scanner.nested (field in highlight), 180
Scanner.pos (field in highlight), 180
Scanner.queue (field in highlight), 180
Scanner.region (field in highlight), 180
Scanner.region_start (field in highlight),
180
Scanner.state (field in highlight), 180
scanner_init (procedure in highlight), 193
SCHEMA_VERSION (constant in core), 142
Scope (struct in pyscan), 290
Scope.defs (field in pyscan), 290
Scratch (struct in core), 117
Scratch.bytes (field in core), 117
Scratch.high (field in core), 117
Scratch.used (field in core), 117
scratch_buffer (procedure in core), 136
scratch_mark (procedure in core), 136
Scratch_Mark (type in core), 117
scratch_release (procedure in core), 137
scratch_take (procedure in core), 137
serve (procedure in project), 279
Server (struct in project), 275
Server.project (field in project), 275
Server.rebuild (field in project), 276
Server.stamp (field in project), 276
Server.user (field in project), 276
Session (struct in autodoc), 85
(struct in host), 206
Session.allocated (field in host), 207
Session allocator (field in host), 206
Session.denied (field in host), 207
Session.disk (field in host), 207
Session.held (field in host), 207
Session.inputs (field in host), 207
Session.kept (field in host), 207
Session.limit (field in host), 207
Session.map_size (field in host), 206
Session.merged (field in autodoc), 86
Session.output (field in host), 206
Session.own (field in host), 207
Session.project (field in autodoc), 86
Session.ready (field in host), 206
Session.refused (field in host), 207
Session.result (field in host), 206
Session.retries (field in host), 206
Session.shared (field in host), 207
Session.sizes (field in host), 206
Session.source_map (field in host), 206
Session.storage (field in host), 206
Session.ws (field in host), 206
session_destroy (procedure in host), 219
session_init (procedure in autodoc), 86
set (procedure in jinja), 247
set_cell (procedure in core), 137
set_header (procedure in i18n), 230
set_header_field (procedure in gettext),
165
set_key (procedure in jinja), 247
set_kind (procedure in core), 137
set_link (procedure in core), 137
set_list (procedure in core), 137
set_span_end (procedure in core), 137
set_text (procedure in core), 137
set_value (procedure in core), 137
Settings (struct in readers/commonmark),
302
(struct in readers/rst), 316
Settings.character_level_inline_markup
(field in readers/rst), 317
Settings.commonmark_only (field in
readers/commonmark), 302
Settings.disable_syntax (field in readers/
commonmark), 302
Settings.dmath_allow_digits (field in
readers/commonmark), 303
Settings.dmath_allow_labels (field in
readers/commonmark), 303
Settings.dmath_allow_space (field in
readers/commonmark), 303
Settings.dmath_double_inline (field in
readers/commonmark), 303
Settings.docinfo (field in readers/rst), 317
Settings.docname (field in readers/
commonmark), 304
Settings.enable_extensions (field in
readers/commonmark), 302
Settings.env_config (field in readers/
commonmark), 304
Settings.extension (field in readers/
commonmark), 302
(field in readers/rst), 316
Settings.footnote_transition (field in
readers/commonmark), 303
Settings.heading_anchors (field in readers/
commonmark), 303
Settings.heading_slug_func (field in
readers/commonmark), 303
Settings.host (field in readers/rst), 316
Settings.html_meta (field in readers/
commonmark), 303
Settings.language (field in readers/rst), 317
Settings.linkify_fuzzy_links (field in
readers/commonmark), 302
Settings.pep_base_url (field in readers/rst),
317

- Settings.pep_references (field in readers/rst), 316
- Settings.rfc_base_url (field in readers/rst), 317
- Settings.rfc_references (field in readers/rst), 317
- Settings.sectsubtitle (field in readers/rst), 317
- Settings.smart_quotes (field in readers/rst), 317
- Settings.smart_quotes_locales (field in readers/rst), 317
- Settings.sub_delimiters (field in readers/commonmark), 303
- Settings.substitutions (field in readers/commonmark), 303
- Settings.tab_width (field in readers/rst), 317
- Settings.today (field in readers/commonmark), 304
- Settings.trim_footnote_reference_space (field in readers/rst), 317
- Settings.url_schemes (field in readers/commonmark), 302
- Settings.words_per_minute (field in readers/commonmark), 304
- settle (procedure in host), 219
- Severity (enum in autodoc), 86
 - (enum in core), 117
 - (enum in odindoc), 255
- Severity.Error (enumerator in autodoc), 86
 - (enumerator in core), 117
 - (enumerator in odindoc), 255
- Severity.Note (enumerator in autodoc), 86
 - (enumerator in core), 117
 - (enumerator in odindoc), 255
- Severity.Warning (enumerator in autodoc), 86
 - (enumerator in core), 117
 - (enumerator in odindoc), 255
- similarity (procedure in gettext), 165
- Sizes (struct in host), 207
- Sizes.attributes (field in host), 207
- Sizes.cells (field in host), 208
- Sizes.links (field in host), 208
- Sizes.lists (field in host), 207
- Sizes.nodes (field in host), 207
- Sizes.output (field in host), 208
- Sizes.reports (field in host), 208
- Sizes.scratch (field in host), 208
- Sizes.second_bank (field in host), 208
- Sizes.sources (field in host), 208
- Sizes.text (field in host), 208
- sizes_for (procedure in host), 219
- slab_plan (procedure in core), 138
- Slab_Plan (struct in core), 118
- Slab_Plan.attributes (field in core), 118
- Slab_Plan.cells (field in core), 118
- Slab_Plan.links (field in core), 118
- Slab_Plan.lists (field in core), 118
- Slab_Plan.nodes (field in core), 118
- Slab_Plan.output (field in core), 118
- Slab_Plan.reports (field in core), 118
- Slab_Plan.scratch (field in core), 118
- Slab_Plan.second_bank (field in core), 118
- Slab_Plan.sources (field in core), 118
- Slab_Plan.text (field in core), 118
- Slot (struct in pyscan), 290
- Slot.doc (field in pyscan), 290
- Slot.name (field in pyscan), 290
- slots (procedure in pyscan), 294
- Smart_Quotes (enum in readers/rst/inline), 328
 - (type in readers/rst), 317
- Smart_Quotes.Alt (enumerator in readers/rst/inline), 328
- Smart_Quotes.Off (enumerator in readers/rst/inline), 328
- Smart_Quotes.On (enumerator in readers/rst/inline), 328
- Snapshot (struct in core), 118
- Snapshot.attributes (field in core), 118
- Snapshot.cells (field in core), 118
- Snapshot.links (field in core), 118
- Snapshot.lists (field in core), 118
- Snapshot.nodes (field in core), 118
- Snapshot.text (field in core), 118
- Snapshot.text_used (field in core), 118
- Snapshot_Storage (struct in core), 119
- Snapshot_Storage.attributes (field in core), 119
- Snapshot_Storage.cells (field in core), 119
- Snapshot_Storage.links (field in core), 119
- Snapshot_Storage.lists (field in core), 119
- Snapshot_Storage.nodes (field in core), 119
- Snapshot_Storage.text (field in core), 119
- sort_by_reference (procedure in gettext), 165
- sort_items (procedure in inventory), 233
- Source (struct in core), 119
- Source.name (field in core), 119
- Source.text (field in core), 119
- source_entry (procedure in core), 138
- Source_Entry (struct in core), 119
- Source_Entry.included_at (field in core), 119
- Source_Entry.parent (field in core), 119
- Source_Entry.use (field in core), 119
- Source_Entry.using (field in core), 119
- Source_Id (type in core), 119
- source_of (procedure in core), 138
- Source_Span (struct in core), 119
- Source_Span.end (field in core), 120
- Source_Span.source (field in core), 120
- Source_Span.start (field in core), 120
- source_text (procedure in core), 138
- Source_Use (enum in core), 120
- Source_Use.Literal (enumerator in core), 120
- Source_Use.Parsed (enumerator in core), 120
- span_of (procedure in readers/rst), 324
- spend (procedure in core), 138
- sphinx_options (procedure in defaults), 143
- spill_disk (procedure in host), 219
- split_label (procedure in core), 138
- split_lines (procedure in pyscan), 294
- Stack (struct in core), 120
- Stack.items (field in core), 120
- Stack.used (field in core), 120
- stack_budget_limit (procedure in host), 220

- stack_diagnostic (procedure in core), 138
- stack_exhausted (procedure in core), 138
- stack_guard (procedure in core), 139
- Stack_Guard (struct in core), 120
- Stack_Guard.base (field in core), 120
- Stack_Guard.limit (field in core), 120
- STACK_RESERVE (constant in core), 142
- stack_slice (procedure in core), 139
- STAGE_PREFIX (constant in host), 222
- standard_role (procedure in readers/rst), 324
- start_byte (procedure in treesitter), 344
- start_point (procedure in treesitter), 344
- State (struct in jinja), 240
- State.autoscape (field in jinja), 240
- State.caller (field in jinja), 241
- State.ctx (field in jinja), 240
- State.current (field in jinja), 240
- State.depth (field in jinja), 240
- State.env (field in jinja), 240
- State.err (field in jinja), 240
- State.failed (field in jinja), 240
- State.guard (field in jinja), 241
- State.iterations (field in jinja), 240
- State.line (field in jinja), 240
- State.out (field in jinja), 240
- State.rng (field in jinja), 241
- State.scratch (field in jinja), 241
- State.stored (field in jinja), 241
- STATES (constant in gds), 151
- STATIC (constant in graphdog), 173
 - (constant in treesitter), 345
- stats (procedure in gettext), 165
- Stats (struct in gettext), 159
- Stats.fuzzy (field in gettext), 159
- Stats.obsolete (field in gettext), 159
- Stats.translated (field in gettext), 159
- Stats.untranslated (field in gettext), 159
- Status (enum in core), 120
- Status.Cancelled (enumerator in core), 121
- Status.Capacity (enumerator in core), 120
- Status.Internal (enumerator in core), 121
- Status.Invalid_Input (enumerator in core), 120
- Status.Limit (enumerator in core), 121
- Status.Ok (enumerator in core), 120
- Status.Policy (enumerator in core), 120
- Status.Unsupported (enumerator in core), 120
- Storage (struct in core), 121
- Storage.banks (field in core), 121
- Storage.reports (field in core), 121
- Storage.scratch (field in core), 121
- Storage.sources (field in core), 121
- str (procedure in jinja), 248
- str_value (procedure in pycan), 294
- String_Kind (enum in pycan), 291
- String_Kind.Bytes (enumerator in pycan), 291
- String_Kind.Formatted (enumerator in pycan), 291
- String_Kind.Str (enumerator in pycan), 291
- string_value (procedure in pycan), 295
- Style (enum in highlight), 180
- Style.Abap (enumerator in highlight), 180
- Style.Algol (enumerator in highlight), 180
- Style.Algol_Nu (enumerator in highlight), 180
- Style.Arduino (enumerator in highlight), 181
- Style.Autumn (enumerator in highlight), 181
- Style.Borland (enumerator in highlight), 181
- Style.Bw (enumerator in highlight), 181
- Style.Coffee (enumerator in highlight), 181
- Style.Colorful (enumerator in highlight), 181
- Style.Default (enumerator in highlight), 181
- Style.Dracula (enumerator in highlight), 181
- Style.Emacs (enumerator in highlight), 181
- Style.Friendly (enumerator in highlight), 181
- Style.Friendly_Grayscale (enumerator in highlight), 181
- Style.Fruity (enumerator in highlight), 181
- Style.Github_Dark (enumerator in highlight), 181
- Style.Gruvbox_Dark (enumerator in highlight), 181
- Style.Gruvbox_Light (enumerator in highlight), 181
- Style.Igor (enumerator in highlight), 181
- Style.Inkpot (enumerator in highlight), 181
- Style.Lightbulb (enumerator in highlight), 181
- Style.Lilypond (enumerator in highlight), 181
- Style.Lovelace (enumerator in highlight), 182
- Style.Manni (enumerator in highlight), 182
- Style.Material (enumerator in highlight), 182
- Style.Monokai (enumerator in highlight), 182
- Style.Murphy (enumerator in highlight), 182
- Style.Native (enumerator in highlight), 182
- Style.None (enumerator in highlight), 182
- Style.Nord (enumerator in highlight), 182
- Style.Nord_Darker (enumerator in highlight), 182
- Style.One_Dark (enumerator in highlight), 182
- Style.Paraiso_Dark (enumerator in highlight), 182
- Style.Paraiso_Light (enumerator in highlight), 182
- Style.Pastie (enumerator in highlight), 182
- Style.Perldoc (enumerator in highlight), 182
- Style.Rainbow_Dash (enumerator in highlight), 182
- Style.Rrt (enumerator in highlight), 182
- Style.Sas (enumerator in highlight), 182
- Style.Solarized_Dark (enumerator in highlight), 182

- Style.Solarized_Light (enumerator in highlight), 182
 - Style.Sphinx (enumerator in highlight), 183
 - Style.Staroffice (enumerator in highlight), 183
 - Style.Stata_Dark (enumerator in highlight), 183
 - Style.Stata_Light (enumerator in highlight), 183
 - Style.Tango (enumerator in highlight), 183
 - Style.Trac (enumerator in highlight), 183
 - Style.Vim (enumerator in highlight), 183
 - Style.Vs (enumerator in highlight), 183
 - Style.Xcode (enumerator in highlight), 183
 - Style.Zenburn (enumerator in highlight), 183
 - Style_Info (struct in highlight), 183
 - Style_Info.background (field in highlight), 183
 - Style_Info.base (field in highlight), 184
 - Style_Info.dark (field in highlight), 183
 - Style_Info.highlight (field in highlight), 183
 - Style_Info.line_number (field in highlight), 183
 - Style_Info.line_number_background (field in highlight), 183
 - Style_Info.name (field in highlight), 183
 - Style_Info.rules (field in highlight), 184
 - Style_Info.special_background (field in highlight), 184
 - Style_Info.special_line_number (field in highlight), 183
 - style_name (procedure in highlight), 193
 - Style_Rule (struct in highlight), 184
 - Style_Rule.definition (field in highlight), 184
 - Style_Rule.kind (field in highlight), 184
 - STYLES (variable in highlight), 194
 - Substitution (struct in readers/commonmark), 304
 - Substitution.name (field in readers/commonmark), 304
 - Substitution.value (field in readers/commonmark), 304
 - Substitution_Hook (struct in readers/rst/inline), 329
 - Substitution_Hook.procedure (field in readers/rst/inline), 329
 - Substitution_Hook.user (field in readers/rst/inline), 329
 - subtree_end (procedure in core), 139
 - suggest_budget (procedure in host), 220
 - suggest_lexer (procedure in highlight), 193
 - symbol (procedure in treesitter), 344
 - Symbol (struct in core), 121
 - Symbol (type in treesitter), 340
 - Symbol.hash (field in core), 121
 - Symbol.key (field in core), 121
 - Symbol.used (field in core), 121
 - Symbol.value (field in core), 121
 - symbol_get (procedure in core), 139
 - symbol_put (procedure in core), 139
 - symbol_set (procedure in core), 139
 - Symbol_Status (enum in core), 121
 - Symbol_Status.Duplicate (enumerator in core), 121
 - Symbol_Status.Full (enumerator in core), 121
 - Symbol_Status.Ok (enumerator in core), 121
 - Symbol_Status.Probe_Limit (enumerator in core), 122
 - Symbol_Table (struct in core), 122
 - Symbol_Table.count (field in core), 122
 - Symbol_Table.max_probe (field in core), 122
 - Symbol_Table.slots (field in core), 122
 - symbol_table_init (procedure in core), 139
 - sync_all (procedure in host), 220
 - sync_each (procedure in host), 220
 - Syntax_Extension (enum in readers/commonmark), 304
 - Syntax_Extension.Amsmath (enumerator in readers/commonmark), 304
 - Syntax_Extension.Colon_Fence (enumerator in readers/commonmark), 304
 - Syntax_Extension.Deflist (enumerator in readers/commonmark), 304
 - Syntax_Extension.Dollarmath (enumerator in readers/commonmark), 304
 - Syntax_Extension.Fieldlist (enumerator in readers/commonmark), 304
 - Syntax_Extension.Html_Admonition (enumerator in readers/commonmark), 304
 - Syntax_Extension.Html_Image (enumerator in readers/commonmark), 304
 - Syntax_Extension.Linkify (enumerator in readers/commonmark), 305
 - Syntax_Extension.Replacements (enumerator in readers/commonmark), 305
 - Syntax_Extension.Smartquotes (enumerator in readers/commonmark), 305
 - Syntax_Extension.Substitution (enumerator in readers/commonmark), 305
 - Syntax_Extension.Tasklist (enumerator in readers/commonmark), 305
 - syntax_extension_named (procedure in readers/commonmark), 307
 - Syntax_Extensions (bit set in readers/commonmark), 305
 - syntax_rule (procedure in readers/commonmark), 307
- ## T
- Table (struct in core), 122
 - Table.rows (field in core), 122
 - Table.used (field in core), 122
 - table_decorate (procedure in readers/rst), 324
 - Table_Extra (struct in readers/rst), 318

- Table_Extra.align (field in readers/rst), 318
- Table_Extra.auto_widths (field in readers/rst), 318
- Table_Extra.classes (field in readers/rst), 318
- Table_Extra.found (field in readers/rst), 318
- Table_Extra.name (field in readers/rst), 318
- Table_Extra.title (field in readers/rst), 318
- Table_Extra.width (field in readers/rst), 318
- Table_Extra.width_class (field in readers/rst), 318
- Table_Extra.widths (field in readers/rst), 318
- take (procedure in readers/rst), 324
- Target (enum in host), 208
 - (enum in i18n), 225
 - (enum in typst), 348
- Target.Doctest_Block (enumerator in i18n), 225
- Target.Html (enumerator in host), 208
 - (enumerator in typst), 348
- Target.Image (enumerator in i18n), 225
- Target.Index (enumerator in i18n), 225
- Target.Literal_Block (enumerator in i18n), 225
- Target.Pdf (enumerator in host), 208
 - (enumerator in typst), 348
- Target.Raw (enumerator in i18n), 225
- Targets (bit set in i18n), 225
 - (type in odindoc), 255
- Template (struct in i18n), 225
 - (struct in jinja), 241
- Template.catalog (field in i18n), 226
- Template.compiled (field in jinja), 241
- Template.env (field in jinja), 241
- Template.name (field in jinja), 241
- Template.seen (field in i18n), 226
- Template.uuids (field in i18n), 226
- text (procedure group in gettext), 166
 - (procedure in core), 139
 - (procedure in pycscan), 295
 - (procedure in treesitter), 344
- Text (struct in core), 122
- Text.length (field in core), 122
- Text.start (field in core), 122
- text_close (procedure in core), 139
- text_domain (procedure in gettext), 165
- text_open (procedure in core), 140
- text_push (procedure in core), 140
- text_push_byte (procedure in core), 140
- text_push_rune (procedure in core), 140
- text_rewind (procedure in core), 140
- thread_stack_bytes (procedure in host), 220
- to_string (procedure in jinja), 248
- Toc_Parents (enum in readers/sphinx), 336
- Toc_Parents.All (enumerator in readers/sphinx), 336
- Toc_Parents.Domain (enumerator in readers/sphinx), 336
- Toc_Parents.Hide (enumerator in readers/sphinx), 336
- today (procedure in gds), 151
- Token (struct in highlight), 184
- Token.end (field in highlight), 184
- Token.kind (field in highlight), 184
- Token.start (field in highlight), 184
- Token_Kind (enum in highlight), 184
- Token_Kind.Chord_Modifier (enumerator in highlight), 190
- Token_Kind.Comment (enumerator in highlight), 187
- Token_Kind.Comment_Hashbang (enumerator in highlight), 188
- Token_Kind.Comment_Multiline (enumerator in highlight), 188
- Token_Kind.Comment_PreProc (enumerator in highlight), 189
- Token_Kind.Comment_Preproc (enumerator in highlight), 188
- Token_Kind.Comment_Preproc_File (enumerator in highlight), 188
- Token_Kind.Comment_Single (enumerator in highlight), 188
- Token_Kind.Comment_Special (enumerator in highlight), 188
- Token_Kind.Error (enumerator in highlight), 184
- Token_Kind.Escape (enumerator in highlight), 184
- Token_Kind.Generic (enumerator in highlight), 188
- Token_Kind.Generic_Deleted (enumerator in highlight), 188
- Token_Kind.Generic_Emph (enumerator in highlight), 188
- Token_Kind.Generic_Emph_Strong (enumerator in highlight), 188
- Token_Kind.Generic_Error (enumerator in highlight), 188
- Token_Kind.Generic_Heading (enumerator in highlight), 188
- Token_Kind.Generic_Inserted (enumerator in highlight), 188
- Token_Kind.Generic_Output (enumerator in highlight), 188
- Token_Kind.Generic_Prompt (enumerator in highlight), 188
- Token_Kind.Generic_Strong (enumerator in highlight), 188
- Token_Kind.Generic_Subheading (enumerator in highlight), 188
- Token_Kind.Generic_Traceback (enumerator in highlight), 188
- Token_Kind.Generic_Underline (enumerator in highlight), 189
- Token_Kind.Keyword (enumerator in highlight), 185
- Token_Kind.Keyword_Constant (enumerator in highlight), 185
- Token_Kind.Keyword_Declaration (enumerator in highlight), 185
- Token_Kind.Keyword_Namespace (enumerator in highlight), 185
- Token_Kind.Keyword_Pseudo (enumerator in highlight), 185

- Token_Kind.Keyword_Reserved
(enumerator in highlight), 185
- Token_Kind.Keyword_Type (enumerator
in highlight), 185
- Token_Kind.Literal (enumerator in
highlight), 186
- Token_Kind.Literal_Date (enumerator in
highlight), 186
- Token_Kind.Literal_Scalar_Plain
(enumerator in highlight), 189
- Token_Kind.Name (enumerator in
highlight), 185
- Token_Kind.Name_Attribute (enumerator
in highlight), 185
- Token_Kind.Name_Backslash_Reference
(enumerator in highlight), 189
- Token_Kind.Name_Builtin (enumerator in
highlight), 185
- Token_Kind.Name_Builtin_Articulation
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Clef
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Context
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Context_Property
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Dynamic
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Grob
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Grob_Property
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Header_Variable
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Markup_Command
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Music_Command
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Music_Function
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Paper_Variable
(enumerator in highlight), 190
- Token_Kind.Name_Builtin_Pseudo
(enumerator in highlight), 185
- Token_Kind.Name_Builtin_Repeat_Type
(enumerator in highlight), 190
- Token_Kind.Name_Builtin_Scale
(enumerator in highlight), 190
- Token_Kind.Name_Builtin_Scheme_Builtin
(enumerator in highlight), 190
- Token_Kind.Name_Builtin_Scheme_Function
(enumerator in highlight), 190
- Token_Kind.Name_Builtin_Translator
(enumerator in highlight), 190
- Token_Kind.Name_Class (enumerator in
highlight), 185
- Token_Kind.Name_Constant (enumerator
in highlight), 185
- Token_Kind.Name_Decorator (enumerator
in highlight), 185
- Token_Kind.Name_Entity (enumerator in
highlight), 185
- Token_Kind.Name_Exception (enumerator
in highlight), 185
- Token_Kind.Name_Function (enumerator
in highlight), 185
- Token_Kind.Name_Function_Magic
(enumerator in highlight), 185
- Token_Kind.Name_Label (enumerator in
highlight), 186
- Token_Kind.Name_Lvalue (enumerator in
highlight), 189
- Token_Kind.Name_Namespace
(enumerator in highlight), 186
- Token_Kind.Name_Other (enumerator in
highlight), 186
- Token_Kind.Name_Property (enumerator
in highlight), 186
- Token_Kind.Name_Tag (enumerator in
highlight), 186
- Token_Kind.Name_Variable (enumerator
in highlight), 186
- Token_Kind.Name_Variable_Class
(enumerator in highlight), 186
- Token_Kind.Name_Variable_Global
(enumerator in highlight), 186
- Token_Kind.Name_Variable_Instance
(enumerator in highlight), 186
- Token_Kind.Name_Variable_Magic
(enumerator in highlight), 186
- Token_Kind.Number (enumerator in
highlight), 187
- Token_Kind.Number_Bin (enumerator in
highlight), 187
- Token_Kind.Number_Float (enumerator in
highlight), 187
- Token_Kind.Number_Hex (enumerator in
highlight), 187
- Token_Kind.Number_Integer (enumerator
in highlight), 187
- Token_Kind.Number_Integer_Long
(enumerator in highlight), 187
- Token_Kind.Number_Oct (enumerator in
highlight), 187
- Token_Kind.Operator (enumerator in
highlight), 187
- Token_Kind.Operator_Word (enumerator
in highlight), 187
- Token_Kind.Other (enumerator in
highlight), 185
- Token_Kind.Pitch (enumerator in
highlight), 190
- Token_Kind.Punctuation (enumerator in
highlight), 187
- Token_Kind.Punctuation_Indicator
(enumerator in highlight), 189
- Token_Kind.Punctuation_Marker
(enumerator in highlight), 187
- Token_Kind.String (enumerator in
highlight), 186

- Token_Kind.String_Affix (enumerator in highlight), 186
 - Token_Kind.String_Backtick (enumerator in highlight), 186
 - Token_Kind.String_Char (enumerator in highlight), 186
 - Token_Kind.String_Delimiter (enumerator in highlight), 186
 - Token_Kind.String_Doc (enumerator in highlight), 186
 - Token_Kind.String_Double (enumerator in highlight), 186
 - Token_Kind.String_Escape (enumerator in highlight), 187
 - Token_Kind.String_Heredoc (enumerator in highlight), 187
 - Token_Kind.String_Interpol (enumerator in highlight), 187
 - Token_Kind.String_Other (enumerator in highlight), 187
 - Token_Kind.String_Regex (enumerator in highlight), 187
 - Token_Kind.String_Single (enumerator in highlight), 187
 - Token_Kind.String_Symbol (enumerator in highlight), 187
 - Token_Kind.Text (enumerator in highlight), 184
 - Token_Kind.Token_Number (enumerator in highlight), 190
 - Token_Kind.Token_String (enumerator in highlight), 190
 - Token_Kind.Token_String_Escape (enumerator in highlight), 190
 - Token_Kind.Token_String_Symbol (enumerator in highlight), 190
 - Token_Kind.Whitespace (enumerator in highlight), 184
 - token_text (procedure in highlight), 193
 - tokenize (procedure in highlight), 193
 - Toml_Kind (enum in project), 276
 - Toml_Kind.Array (enumerator in project), 276
 - Toml_Kind.Boolean (enumerator in project), 276
 - Toml_Kind.Datetime (enumerator in project), 276
 - Toml_Kind.Float (enumerator in project), 276
 - Toml_Kind.Integer (enumerator in project), 276
 - Toml_Kind.String (enumerator in project), 276
 - Toml_Kind.Table (enumerator in project), 276
 - Toml_Value (struct in project), 276
 - Toml_Value.boolean (field in project), 277
 - Toml_Value.defined (field in project), 277
 - Toml_Value.float (field in project), 277
 - Toml_Value.inline (field in project), 277
 - Toml_Value.integer (field in project), 276
 - Toml_Value.items (field in project), 277
 - Toml_Value.keys (field in project), 277
 - Toml_Value.kind (field in project), 276
 - Toml_Value.line (field in project), 277
 - Toml_Value.text (field in project), 276
 - Toml_Value.values (field in project), 277
 - top (procedure in core), 140
 - transform_document (procedure in core), 140
 - translated (procedure in gettext), 166
 - translation (procedure in gettext), 166
 - Translation (struct in i18n), 226
 - Translation.catalog (field in i18n), 226
 - Translation.images (field in i18n), 227
 - Translation.kept (field in i18n), 227
 - Translation.limits (field in i18n), 226
 - Translation.policy (field in i18n), 226
 - Translation.progress (field in i18n), 226
 - Translation.read (field in i18n), 226
 - Translation.read_inline (field in i18n), 226
 - Translation.source_name (field in i18n), 226
 - Translation.targets (field in i18n), 226
 - Translation.translated (field in i18n), 227
 - translation_pass (procedure in i18n), 230
 - Translator (struct in gettext), 159
 - Translator allocator (field in gettext), 159
 - Translator.bound (field in gettext), 159
 - Translator.domain (field in gettext), 159
 - Translator.language (field in gettext), 159
 - Tree (struct in treesitter), 340
 - tree_delete (procedure in treesitter), 345
 - treesitter (package), 339
 - trim_blank_end (procedure in readers/rst), 324
 - trusted_policy (procedure in core), 140
 - truthy (procedure in jinja), 248
 - try_unparse (procedure in pyscan), 295
 - ts (foreign import in treesitter), 345
 - tuple (procedure in jinja), 248
 - Type_Alias (struct in autodoc), 86
 - Type_Alias.name (field in autodoc), 86
 - Type_Alias.target (field in autodoc), 86
 - type_name (procedure in jinja), 248
 - typst (package), 345
 - typst_highlighter (procedure in highlight/guidedoc), 195
- ## U
- UI_DOMAIN (constant in i18n), 231
 - UI_Problem (struct in i18n), 227
 - UI_Problem.error (field in i18n), 227
 - UI_Problem.path (field in i18n), 227
 - ui_text (procedure in i18n), 230
 - ui_translator (procedure in i18n), 230
 - undefined (procedure in jinja), 248
 - Undefined (struct in jinja), 241
 - Undefined.hint (field in jinja), 241
 - Undefined.name (field in jinja), 241
 - Undefined.owner (field in jinja), 241
 - Undefined_Mode (enum in jinja), 242
 - Undefined_Mode.Chainable (enumerator in jinja), 242
 - Undefined_Mode.Debug (enumerator in jinja), 242
 - Undefined_Mode.Default (enumerator in jinja), 242
 - Undefined_Mode.Strict (enumerator in jinja), 242

unescape (procedure in readers/rst/inline), 330
 Unit (enum in i18n), 227
 Unit.Alt (enumerator in i18n), 227
 Unit.Caption (enumerator in i18n), 228
 Unit.Contents (enumerator in i18n), 228
 Unit.Entry (enumerator in i18n), 228
 Unit.Graphviz (enumerator in i18n), 227
 Unit.Index (enumerator in i18n), 228
 Unit.Inline (enumerator in i18n), 227
 Unit.Literal (enumerator in i18n), 227
 Unit.Meta (enumerator in i18n), 228
 Unit.None (enumerator in i18n), 227
 Unit.Parsed (enumerator in i18n), 227
 unit_of (procedure in i18n), 230
 unload (procedure in project), 279
 unparse (procedure in pyscan), 295
 Unparse_Mode (enum in pyscan), 291
 Unparse_Mode.Python (enumerator in pyscan), 291
 Unparse_Mode.Sphinx (enumerator in pyscan), 291
 Unparse_Mode.Sphinx_Repr (enumerator in pyscan), 291
 unwrap_node (procedure in core), 140

V

validate (procedure in core), 141
 validate_registry (procedure in core), 141
 Value (union in jinja), 242
 values_equal (procedure in jinja), 248
 VERSION (constant in cli), 87
 (constant in core), 142
 (constant in readers/commonmark), 307
 (constant in readers/rst), 324
 (constant in readers/sphinx), 337
 (constant in render/html), 337
 (constant in render/text), 338
 (constant in render/typst), 339
 version (procedure in graphdog), 173
 VERSION_SUFFIX (constant in cli), 87
 version_text (procedure in cli), 87
 view (procedure in core), 141
 View (struct in core), 122
 View.snap (field in core), 122
 View.sources (field in core), 122

W

was_interrupted (procedure in host), 220
 with_context (procedure group in gettext), 166
 Workspace (struct in core), 122
 Workspace.active (field in core), 123
 Workspace.banks (field in core), 123
 Workspace.generation (field in core), 123
 Workspace.keep_errors (field in core), 123
 Workspace.output (field in core), 123
 Workspace.ready (field in core), 123
 Workspace.reports (field in core), 123
 Workspace.revisions (field in core), 123
 Workspace.scratch (field in core), 123
 Workspace.sources (field in core), 123
 Workspace.sources_used (field in core), 123
 workspace_size (procedure in core), 141
 Wrapper (enum in pyscan), 291
 Wrapper.Cached_Property (enumerator in pyscan), 291

Wrapper.Classmethod (enumerator in pyscan), 291
 Wrapper.None (enumerator in pyscan), 291
 Wrapper.Property (enumerator in pyscan), 291
 Wrapper.Staticmethod (enumerator in pyscan), 291
 Wrapper.Unknown (enumerator in pyscan), 292
 write (procedure in inventory), 234
 write_css (procedure in highlight), 194
 write_figure_filename (procedure in i18n), 231
 write_html (procedure in highlight), 194
 write_html_lines (procedure in highlight), 194
 write_id (procedure in core), 141
 write_message (procedure in i18n), 231
 write_mo (procedure in gettext), 166
 write_object (procedure in core), 141
 Write_Options (struct in gettext), 160
 Write_Options.obsolete (field in gettext), 160
 Write_Options.previous (field in gettext), 160
 Write_Options.references (field in gettext), 160
 Write_Options.width (field in gettext), 160
 write_po (procedure in gettext), 166
 write_problem (procedure in highlight), 194
 write_template (procedure in i18n), 231
 write_typst (procedure in highlight), 194
 write_typst_theme (procedure in highlight), 194
 write_unsynced (procedure in host), 220