

Guidedog 설명서

설명서와 전체 Odin API 참조

01 / 소스

의미를 담아 쓴다.

02 / 구조

관계를 명확히 한다.

03 / 게시

화면과 종이에서 읽는다.

차례

따라 하기	3
첫 프로젝트	3
작은 책 만들기	4
주제별 안내	7
문서 작성	7
도메인	12
프로젝트의 동작 방식	16
수학과 증명	18
개념을 설명하는 다이어그램	19
읽기 위한 조판	20
템플릿	21
이미지, 동영상, 오디오, 다운로드	28
국제화	29
다른 프로젝트로 연결	32
객체 타입 선언	32
autodoc으로 Python 문서 작성하기	33
Odin 문서화	36
실용적인 작업	43
Guidedog 빌드와 설치	43
문서 빌드와 게시	44
Sphinx 프로젝트 옮기기	45
코드베이스 문서화	45
이 문서 관리하기	46
참조	49
명령줄 참조	49
설정 참조	53
리더, 렌더러, 빌더	60
오류 메시지	61
Sphinx 호환성	64
내부 구조	69
Guidedoc 엔진	69
알고리즘에 앞서는 불변식	70
완전한 출력 세대 게시	71
메모리와 소유권	72
이 설명서에 관하여	75
빌드 검증 기록	75
Guidedog	77
프로젝트 시작	77

설치	77
소스에서 빌드	78
Guidedog으로 번역	78
문서 읽기와 임베딩	79
기여자와 라이선스	79
API Reference	81
autodoc	82
cli	86
core	88
defaults	142
fetch	143
gds	145
gettext	151
graphdog	167
highlight	174
highlight/guidedoc	194
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
색인	351

한 번 쓰고, 잘 설명하고, 웹과 종이에 게시하세요.

Guidedog은 reStructuredText와 Markdown 소스로 HTML 사이트와 PDF 책을 만드는 문서 생성기입니다. Python 환경이나 별도의 LaTeX 설치가 필요하지 않습니다. 변환 엔진의 이름은 **Guidedoc**입니다. PDF는 Typst로 조판하며, 웹사이트는 Guidedog이 직접 생성합니다.

작은 프로젝트에는 소스 문서, 목차, `conf.toml` 세 가지가 필요합니다. 첫 프로젝트부터 시작하세요. 구체적인 작업은 실용적인 작업, 정확한 규약은 참조를 참고하세요.

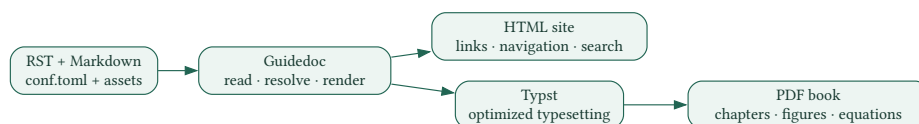


그림 1 하나의 소스 프로젝트, 두 가지 출판물.

따라 하기

순서대로 따라 하세요. 각 단계에서 확인할 수 있는 결과가 나옵니다.

첫 프로젝트

목표는 간단합니다. 한 페이지를 브라우저에서 보고 같은 내용을 PDF로 만듭니다. Guidedog이 필요합니다. PDF에는 내장 컴파일러나 설치된 Typst 0.15.1 이상을 사용할 수 있습니다. [Guidedog 빌드와 설치](#) 을 참고하세요.

macOS에서는 `brew install insanai/guidedog/guidedog` 로 릴리스를 설치하세요. 공식 `tap`이 추가되며 PDF 컴파일러도 함께 설치됩니다.

프로젝트 만들기

`notes`가 아직 없는 디렉터리에서 실행하세요.

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

Guidedog는 설정, 첫 RST 문서, 편집 가능한 템플릿을 작성합니다. 이미지, 오디오, 동영상 폴더도 만듭니다. 기존 프로젝트의 덮어쓰기는 거부합니다.

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

RST 파일에는 내용을, 템플릿에는 표현 방식을 담습니다. `quickstart` 는 `_templates/layout.html` 과 `_templates/book.typ` 을 외부 테마나 캐시가 아닌 프로젝트에 직접 생성합니다. 언제든지 보고 수정할 수 있습니다.

첫 페이지 작성

`notes/index.rst` 를 다음으로 바꾸세요.

```
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.
```

빌드와 확인

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

notes/_build/html/index.html 과 notes/_build/pdf 의 PDF를 여세요. -W는 경고가 있으면 빌드를 실패시키므로 문서 검사에 유지하세요. HTML에는 탐색과 본문이 있으며, PDF에는 책의 머리말·페이지 번호 등도 들어갑니다. 수식은 각 형식에 맞춰 조판합니다. 설정은 [수학과 증명](#)에서 설명합니다.

내용 바꾸기

```
guidedog serve notes
```

명령이 출력한 주소를 여세요. 소스를 수정하고 저장한 뒤 브라우저를 새로 고치세요. Guidedog는 HTML을 제공하기 전에 소스 수정 시각을 확인하고, 변경 되었으면 다시 빌드합니다. 브라우저 새로 고침을 자동으로 보내지는 않습니다. Ctrl+C로 서버를 멈춥니다.

수정 없이 다시 빌드하세요. 변경되지 않은 문서는 재사용하므로, 출력이 최신인지 확인하기 위해 모든 페이지를 다시 읽을 필요가 없습니다. 작은 책 만 들기

작은 책 만들기

사이트는 문서로 이루어진 그래프입니다. 책에는 읽기 순서가 필요합니다. `toctree`는 둘 다 제공합니다.

장 추가

notes/measurement.rst를 만드세요.

```
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.rst의 내용을 다음으로 바꾸세요.

```
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`.
```

문서 이름에서 .rst는 생략합니다. 레이블은 문서 안의 개념에 이름을 붙입니다. 장에는 문서 링크를, 특정 논증에는 레이블을 사용하세요.

다이어그램 추가

다음을 measurement.rst에 넣으세요.

```
.. 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";
   }
```

Guidedog는 링크된 Graphviz 라이브러리를 사용합니다. HTML에는 SVG, PDF에는 이미지가 들어가므로 브라우저용 다이어그램 해석기가 필요하지 않습니다. 캡션은 결론을, 대체 텍스트는 그림을 설명합니다.

책 빌드

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

장은 사이드바와 책의 목차에 나타납니다. 브라우저에서 절 링크를 따라가고 PDF에서도 목적지를 확인하세요.

다음으로 문서 작성을 읽어 보세요. 각 장이 한 주제를 설명할 수 있도록 짧게 유지하세요.

주제별 안내

이 장들은 모델을 설명하고 선택에 필요한 배경을 제공한다.

문서 작성

Guidedog은 reStructuredText (.rst)와 MyST Markdown (.md)을 지원합니다. 두 형식을 공통 문서 트리로 파싱해 HTML 사이트와 PDF 책을 생성합니다.

소스 언어 선택

도메인, 복잡한 표, 확장 가능한 지시문이 필요하면 reStructuredText를 사용하세요. Markdown 구문과 Sphinx 스타일의 지시문 및 역할을 함께 쓰려면 MyST Markdown을 선택하세요.

프로젝트의 Markdown 문서는 MyST로 파싱합니다. 단일 파일을 변환하는 `guidedog convert` 는 MyST를 명시하지 않으면 CommonMark를 사용합니다.

제목과 구조

문서 제목은 페이지를 식별하고 절과 하위 절은 내용을 나눕니다. RST 제목에는 밑줄을 쓰며 같은 윗줄도 추가할 수 있습니다. 줄 길이는 제목보다 짧으면 안 됩니다.

권장하는 제목 기호:

```
Document Title
=====

Section Heading
-----

Subsection
~~~~~

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

Paragraph Heading
.....
```

프로젝트 전체에서 제목 수준을 일관되게 유지하세요. MyST Markdown에서는 #, ##, ### 를 사용합니다.

텍스트 형식과 인라인 마크업

인라인 마크업은 장식보다 의미를 전달하는 데 사용하세요.

표 1 인라인 마크업

스타일	reStructuredText	MyST Markdown
굵게	<code>**strong text**</code>	<code>**strong text**</code>
강조 / 기울임	<i><code>*emphasized text*</code></i>	<i><code>*emphasized text*</code></i>
인라인 코드	<code><code>`inline code`</code></code>	<code><code>`inline code`</code></code>
아래 첨자	<code>:sub:`text`</code>	<code>{sub}`text`</code>
위 첨자	<code>:sup:`text`</code>	<code>{sup}`text`</code>

별표나 백틱을 문자 그대로 쓰려면 `\*not italic\*` 처럼 백슬래시로 이스케이프하세요.

목록

Guidedog은 네 종류의 목록을 지원합니다.

글머리 기호 목록

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

번호 목록

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

정의 목록

정의 목록은 용어와 설명을 짝으로 묶습니다. 용어를 한 줄에 쓰고 바로 다음 줄부터 들여쓴 정의를 적습니다.

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

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

필드 목록

필드 목록은 구조화된 메타데이터와 매개변수 설명에 사용합니다.

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

코드 블록

`code-block` 또는 `sourcecode` 로 구문을 강조한 코드를 표시하세요. Guidedog은 35개 이상의 프로그래밍 언어를 지원합니다.

```
.. 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)
```

코드 블록 옵션:

- `:linenos:`: 코드 옆에 줄 번호를 표시합니다.
- `:caption: Title:` 코드 위나 아래에 캡션을 붙입니다.
- `:emphasize-lines: 1, 3-5:` 지정한 줄을 강조합니다.
- `:name: label:` 코드 블록에 참조 대상을 지정합니다.

외부 소스 파일을 직접 포함하려면 `literalinclude` 를 사용하세요.

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

안내 상자

안내 상자는 주의 사항, 경고, 보충 설명을 강조합니다.

```
.. 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.
```

Guidedog은 `danger`, `error`, `hint`, `attention` 도 지원합니다. 제목을 직접 지정하려면 `admonition` 을 사용하세요.

```
.. admonition:: Design Rationale

    Explains why a particular architecture was chosen.
```

표

Guidedog은 단순 표, 격자 표, 지시문 기반 표를 지원합니다.

단순 표

단순 표는 가로선으로 열의 범위를 표시합니다.

```
=====
A      B      A and B
=====
False  False  False
```

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

격자 표

격자 표는 여러 줄의 셀과 행·열 병합을 지원합니다.

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

목록 표

`list-table` 은 중첩 클머리 기호 목록으로 표를 만듭니다. 넓은 표도 소스에서 읽고 관리하기 쉽습니다.

```
.. 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 표

`csv-table` 은 쉼표로 구분한 값으로 표를 만듭니다.

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

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

이미지와 그림

`image` 와 `figure` 로 이미지, 스크린샷, 도표를 넣습니다.

```
.. 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` 는 이미지에 캡션을 붙이며 참조 대상도 지정할 수 있습니다.

목차 트리로 문서 구성

`toctree` 는 문서 계층, 탐색 메뉴, HTML과 PDF의 읽는 순서를 정의합니다.

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

    installation
    quickstart
    configuration
```

toctree 의 주요 옵션:

- `:maxdepth: N`: 목차에 포함할 제목 깊이입니다.
- `:caption: Title`: 탐색 항목 위에 표시할 범주 제목입니다.
- `:numbered:`: 장과 제목에 번호를 붙입니다.
- `:titlesonly:`: 문서 제목만 표시하고 내부 절은 생략합니다.
- `:hidden:`: 본문에 목록을 표시하지 않고 문서의 읽는 순서만 기록합니다.
- `:glob: tutorials/*` 같은 와일드카드로 문서를 선택할 수 있습니다.

상호 참조와 링크

명시적 대상 레이블은 문서 파일 이름과 별개로 안정적인 참조를 제공합니다.

대상 레이블과 `:ref:`

제목, 표, 그림 바로 앞에 대상 레이블을 놓습니다.

```
.. _storage-model:

Storage model
-----

Caller storage is bounded and measured upfront.
```

프로젝트의 어느 문서에서든 이 대상을 참조할 수 있습니다.

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

`:doc:` 문서 참조

확장자를 생략한 경로로 다른 문서에 연결합니다.

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

`:term:` 용어 참조

glossary 로 용어를 정의합니다.

```
.. 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:` 역할로 용어에 연결합니다.

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

`:download:` 파일 다운로드

파일을 다운로드 디렉터리로 복사하고 링크를 만듭니다.

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

외부 링크

외부 웹 주소에 연결합니다.

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

치환과 포함

재사용할 문구, 기호, 이미지를 정의합니다.

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

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

여러 파일에서 reStructuredText 내용을 공유하려면 다음과 같이 작성합니다.

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

MyST Markdown 작성

MyST Markdown은 펜스 블록으로 같은 지시문과 역할의 의미를 표현합니다.

```
# 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`.
```

도메인

도메인은 특정 프로그래밍 언어나 분야의 객체를 설명하고 상호 참조하는 지시문과 역할의 모음입니다.

Guidedog에는 다음 Sphinx 도메인이 기본 구현되어 있습니다.

- **표준 도메인** (`std`): 명령줄 옵션, 환경 변수, 용어, UI 레이블 등 일반 문서 객체입니다.
- **Python 도메인** (`py`): 모듈, 클래스, 함수, 메서드, 속성, 프로퍼티, 예외입니다.

- **C 도메인** (c): 함수, 타입, 구조체, 공용체, 열거형, 열거자, 변수, 매크로입니다.
- **C++ 도메인** (cpp): 클래스, 콘셉트, 템플릿, 네임스페이스, 메서드, 표현식입니다.
- **JavaScript 도메인** (js): 함수, 메서드, 클래스, 프로퍼티입니다.
- **Odin 도메인** (odin): Odin 패키지, 프로시저, 구조체, 공용체, 열거형입니다. [Odin 문서화](#) 을 참고하세요.

기본 도메인 설정

conf.toml 의 primary_domain 으로 기본 도메인을 설정합니다. 기본값은 "py" 입니다. 설정하면 접두사를 생략할 수 있습니다. 예를 들어 .. py:function:: 대신 .. function:: 을, :py:func:`name` 대신 :func:`name` 을 쓸 수 있습니다.

문서 안에서 현재 도메인을 바꾸려면 다음과 같이 작성합니다.

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

표준 도메인

표준 도메인은 명령줄 도구, 환경 변수, 문서 상호 참조를 설명합니다.

지시문

```
.. program:: mytool

.. option:: -c <config>, --config <config>
    Path to the configuration file.

.. envvar:: GUIDEDOG_THEME
    Specifies the default theme stylesheet.
```

역할

- :doc:`path`: 상대 경로로 프로젝트 문서에 연결합니다.
- :ref:`label`: 명시적 대상 레이블 .. _label: 에 연결합니다.
- :term:`term`: glossary 에 정의한 용어에 연결합니다.
- :option:`--config`: option 에 설명한 명령줄 옵션에 연결합니다.
- :envvar:`VARIABLE`: envvar 에 설명한 환경 변수에 연결합니다.
- :command:`name`: 시스템 명령 이름을 표시합니다.
- :file:`path`: 파일이나 디렉터리 경로를 표시합니다. {variable} 자리표시자도 지원합니다.
- :kbd:`Ctrl+C`: 키보드 입력을 표시합니다.
- :menuselection:`File --> Save As`: 메뉴 선택 순서를 표시합니다.
- :guilabel:`Submit`: 버튼이나 UI 요소를 표시합니다.
- :pep:`8`: Python 개선 제안에 연결합니다.
- :rfc:`7231`: IETF RFC 문서에 연결합니다.

Python 도메인

Python 도메인은 Python 인터페이스와 시그니처를 설명합니다.

지시문

```
.. 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.
```

지원하는 지시문은 `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` 입니다.

역할

- `:py:func:`name``: Python 함수에 연결합니다.
- `:py:class:`name``: Python 클래스에 연결합니다.
- `:py:meth:`name``: Python 클래스 메서드 또는 인스턴스 메서드에 연결합니다.
- `:py:attr:`name``: Python 속성에 연결합니다.
- `:py:data:`name``: Python 모듈 수준 데이터에 연결합니다.
- `:py:exc:`name``: Python 예외 클래스에 연결합니다.
- `:py:mod:`name``: Python 모듈에 연결합니다.

C 도메인

C 도메인은 C 라이브러리와 헤더 파일을 설명합니다.

지시문

```
.. 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.
```

역할

- `:c:func:`name``: C 함수에 연결합니다.
- `:c:member:`name``: C 구조체 또는 공용체 멤버에 연결합니다.
- `:c:data:`name`` / `:c:var:`name``: C 변수에 연결합니다.
- `:c:type:`name``: C typedef 또는 타입에 연결합니다.
- `:c:struct:`name``: C 구조체에 연결합니다.
- `:c:union:`name``: C 공용체에 연결합니다.
- `:c:enum:`name``: C 열거형에 연결합니다.
- `:c:macro:`name``: C 전처리기 매크로에 연결합니다.

C++ 도메인

C++ 도메인은 현대 C++ 구문, 콘셉트, 네임스페이스, 범위를 지원합니다.

지시문

```
.. 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.
```

역할

- `:cpp:class:`name``: C++ 클래스 또는 구조체에 연결합니다.
- `:cpp:func:`name``: C++ 함수 또는 메서드에 연결합니다.
- `:cpp:member:`name`` / `:cpp:var:`name``: 멤버 또는 변수에 연결합니다.
- `:cpp:type:`name``: C++ 타입 별칭 또는 typedef에 연결합니다.
- `:cpp:concept:`name``: C++20 콘셉트에 연결합니다.
- `:cpp:enum:`name``: C++ 열거형에 연결합니다.

JavaScript 도메인

JavaScript 도메인은 브라우저 스크립트, 라이브러리, Node.js 모듈을 설명합니다.

지시문

```
.. js:module:: theme
.. js:class:: ThemeController(options)
    Manages color themes and local storage persistence.
.. js:method:: toggleTheme()
    Toggles between light and dark modes.
```

역할

- `:js:func:`name``: JavaScript 함수에 연결합니다.
- `:js:meth:`name``: JavaScript 메서드에 연결합니다.
- `:js:class:`name``: JavaScript 클래스에 연결합니다.
- `:js:data:`name``: JavaScript 변수에 연결합니다.
- `:js:attr:`name``: JavaScript 객체 속성에 연결합니다.
- `:js:mod:`name``: JavaScript 모듈에 연결합니다.

상호 참조 규칙

도메인 상호 참조는 다음과 같은 수식 방법을 지원합니다.

- **접두사 숨기기**: 대상 앞에 `~` 를 붙이면 이름의 마지막 부분만 표시합니다. `:py:func:`~pipeline.reader.DocumentReader.parse`` 는 `parse()` 로 표시됩니다.
- **현재 범위에서 검색**: 상대 참조는 현재 모듈이나 네임스페이스를 먼저 찾고, 없으면 전체 범위에서 검색합니다.
- **정확한 일치**: 대상 앞에 `.` 을 붙이면 현재 모듈이나 포함 객체부터 검색합니다.

프로젝트의 동작 방식

프로젝트 루트는 `conf.toml` 이 있는 폴더다. `root_doc` 은 읽기 순서의 시작 문서다. Guidedog은 소스 문서를 찾고 읽어 관계를 해결한 뒤 완전한 출력 세대를 게시한다.

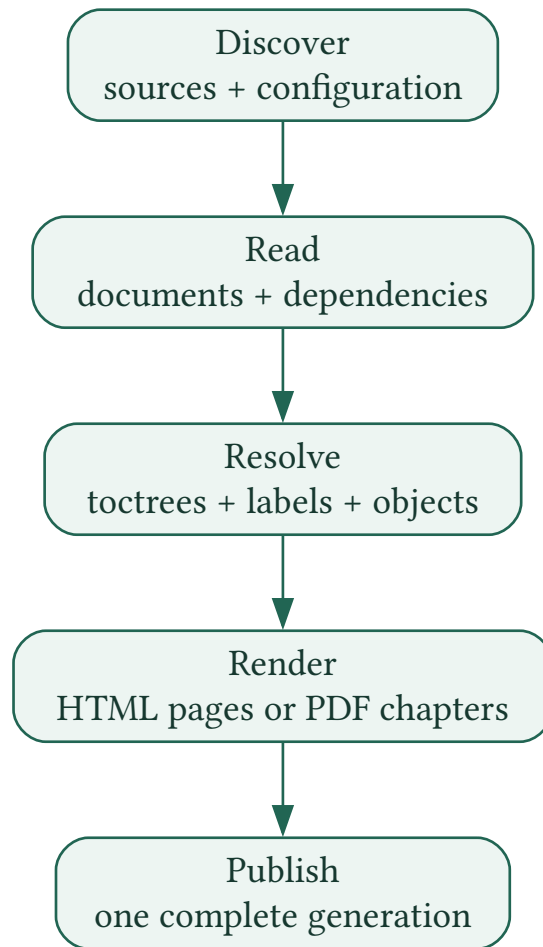


그림 2 빌드는 탐색, 의미 처리, 표현, 게시를 분리한다.

문서와 자원

확장자로 리더를 선택한다. RST와 프로젝트 Markdown은 그래프를 공유한다. 이미지와 다운로드 리소스는 문서가 쓰는 것을 복사한다. 테마 자산은 `html_static_path`에 둔다. 포함 파일은 소스 또는 `include_roots` 안에 있어야 한다.

독립된 프로젝트

설명서와 설계 기록은 독자와 읽기 순서가 달라 별도 프로젝트로 둔다. 각각 설정, 템플릿, 캐시, 출력을 갖는다. Guidedog의 `docs/manual`과 `docs/gds`가 이 구조다.

중분 빌드

Guidedog은 출력별 입력과 조회를 기록한다. 소스가 바뀌면 다시 읽고 의존성이 바뀌면 사용자를 무효화한다. 템플릿 변경은 모든 페이지에 영향을 줄 수 있다. 그대로인 문서는 저장된 객체에서 읽는다.

캐시는 이전 빌드의 증거이며 원본 진실이 아니다. `-E`는 다음 빌드의 저장 환경을 버리고 `-a`는 모든 출력을 쓴다. 서로 다른 문제를 해결한다.

재사용 = 같은 입력 $\wedge$ 같은 조회 결과 $\wedge$ 출력 존재

템플릿의 타임스탬프도 입력이다. 현재 분을 표시하는 페이지는 글이 그대로 여도 바뀔 수 있다. 고정 시간이 필요하면 `SOURCE_DATE_EPOCH`를 쓴다.

게시

빌드는 형제 임시 폴더에 변경을 쓰고 완성한 뒤 파일과 기록을 동기화하고 세대를 바꾼다. 독자는 완전한 이전 또는 새 게시물을 본다. 중단 커밋은 다음 빌드가 처리한다. 설계와 증명 의무는 완전한 출력 세대 게시에 있다.

수학과 증명

관계를 살피기 쉬워질 때 수식을 쓴다. 기호를 먼저 정의하고 필요한 단위를 밝힌다. 수식은 설명을 돕는 것이며 대신하지 않는다.

인라인 수식과 독립 수식

짧은 수식은 `math` 역할, 독립 수식은 `math` 지시문을 쓴다. 두 대상이 모두 이해하는 공통 LaTeX 수학 부분집합을 사용한다.

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

.. math::
   :label: area-square

   A = s^2

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

한 변의 길이가 s 인 정사각형의 넓이는 s^2 이다.

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

관계식은 (1)다. Guidedog은 레이블 수식에 안정적인 대상을 준다. HTML은 MathJax 마크업을, PDF는 지원 LaTeX 부분집합을 Typst 수학으로 변환한다. 미지원 PDF 명령은 보고하며 임의 LaTeX 패키지는 실행하지 않는다.

간단한 증명

계상된 바이트를 U , 한도를 B , 요청량을 n 이라 하자. 모두 음수가 아니고 $U \leq B$ 라 가정한다.

허용 규칙. 다음 조건이 성립할 때만 요청을 허용한다.

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

명제. 허용된 요청은 $U + n \leq B$ 를 유지한다.

증명. (2) 양변에 U 를 더하면 $U + n \leq B$ 다. 구현은 뺄셈 형태를 검사하며 정수 오버플로가 가능한 $U + n$ 을 먼저 계산하지 않는다. 증명과 연산 순서가 일치한다.

경계. 이는 계상된 호스트 공간을 다룬다. 별도 Typst 컴파일러의 내부 할당을 제한하지 않는다. 증명은 밝힌 경계 안에서만 유효하다.

HTML 설정

이 프로젝트는 MathJax 3.2.2와 SVG 글꼴 데이터를 포함하며 수식이 있는 페이지에서만 불러온다. 빌드 뒤 오프라인에서도 읽을 수 있고 그림 스크립트나 웹 글꼴 서비스가 필요 없다.

사용자 프로젝트는 기본적으로 CDN의 MathJax를 사용한다. 로컬 사본을 제공하려면 `_static`에 넣고 다음을 설정한다.

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

웹 수식 조판에는 JavaScript가 필요하며 없으면 소스를 보여 준다. PDF 수식에는 브라우저 JavaScript가 필요 없다. [MathJax 구성 요소 계약](#)을 참고한다.

개념을 설명하는 다이어그램

질문에 맞는 그림을 고른다. 순서도는 선택, 블록도는 책임, 시퀀스는 순서, 그래프는 관계를 설명한다. 글이 더 직접 답하면 글을 쓴다.

순서도

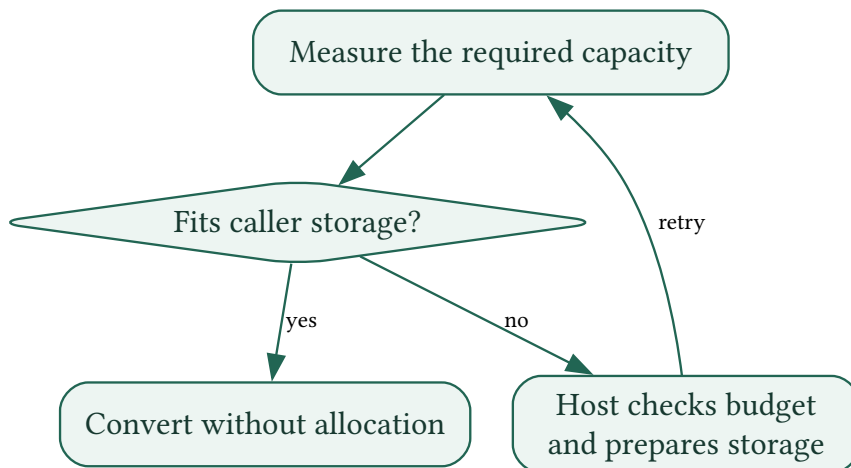


그림 3 저장 공간을 빌리거나 늘리기 전에 용량을 확인한다.

그림은 확장 단계의 소유자를 밝힌다. 변환 코어가 할당한다고 암시하지 않는다. 이 구분은 계약의 일부다.

그래프 작성

```
.. 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, digraph는 내장되어 있다. Guidedog이 Graphviz로 렌더링해 HTML에는 SVG를, PDF에는 그림을 넣는다. 그래프 소스는 같다.

읽기 쉽게 다듬기

노드 레이블은 짧게 하고 긴 규칙은 아래에 설명한다. 주 경로는 한 방향으로 두고 돌아오는 간선에 조건을 쓴다. 색 외에 형태와 텍스트도 쓴다. 인쇄물에는 색이 없을 수 있다.

대체 텍스트와 캡션을 제공하고 좁은 브라우저와 PDF에서 확인한다. 작은 글씨여야 들어가는 그림은 둘로 나눈다. 이미지 크기는 이미지, 동영상, 오디오, 다운로드를 참고한다.

읽기 위한 조판

읽기 전에 페이지 구조가 보여야 한다. 뚜렷한 제목, 차분한 탐색 열, 적절한 본문 너비가 핵심이다. 색은 관계나 동작을 표시하며 의미를 혼자 전달하지 않는다.

웹 페이지

맞춤 레이아웃은 프로젝트 탐색, 본문, 페이지 개요 세 영역이다. 작은 화면은 탐색 서랍을 쓴다. Libertinus Serif와 Sans를 포함하며 제목은 굵은 세리프, 탐색.레이블.컨트롤은 산세리프를 쓴다. 압축 WOFF2를 사이트가 제공하므로 글꼴 서비스 요청은 없다.

기본은 밝은 테마다. 어두운 테마와 시스템 모드도 제공하며 같은 출처에서 선택을 기억한다. 시스템 모드는 운영체제를 따른다. HTML 인쇄는 밝은 색이다. JavaScript가 없어도 내용과 탐색은 사용 가능하며 테마와 향상된 검색은 점진적 향상이다.

영어 본문은 브라우저에서도 문단 전체의 **Knuth-Plass 줄바꿈**을 쓸 수 있다. 작은 스크립트가 글꼴을 측정해 박스, 공백, 하이픈 별점을 만들고 문단 전체의 줄바꿈을 고른다. Frank Liang 패턴 방식이 영어 후보를 제공하며 원래 TeX 패턴과 라이선스를 포함한다.

강조와 원문을 유지하며 본문 너비가 바뀌면 다시 계산한다. 최적화한 한 문단 안에서 복사하면 시각적 줄바꿈이나 추가 하이픈 없는 원문을 얻는다. 인쇄 시 일반 문단으로 복원해 인쇄 레이아웃이 페이지를 나눈다.

링크, 수식, 코드, 상호작용, 좁은 너비, 다른 언어는 브라우저 기본 레이아웃을 쓴다. 적절한 간격 해가 없는 문단도 같다. JavaScript가 없어도 읽을 수 있다. 대체 방식은 언어별 `hyphens: auto`와 `text-wrap: pretty`지만 CSS만으로 Knuth-Plass 최적화를 보장하지 않는다.

PDF 페이지

책 템플릿도 Libertinus를 사용해 양쪽 정렬, 최적 줄바꿈, 언어별 하이픈을 설정한다. 간격, 머리글, 캡션, 코드 블록, 표 선도 제어한다. 글꼴, 하이픈 비용, 문단 너비가 함께 작용한다. 느슨한 문단을 강제 줄바꿈으로 고치는 것은 대개 적절하지 않다.

Typst는 PDF 문단을, 웹 향상은 해당 본문을 최적화한다. 소스·글꼴·시각 계층은 같지만 줄바꿈은 다를 수 있다. Typst 텍스트 계약과 브라우저 하이픈 계약을 참고한다.

디자인 바꾸기

시각 토큰은 `_static/editorial.css`, 구조는 `_templates/layout.html`, 인쇄는 `_templates/book.typ`를 편집한다. 읽기 순서, 초점 표시, 레이블, 충분한 대비를 유지한다.

수정 뒤 두 대상을 빌드하고 코드·표·그림·수식 페이지를 확인한다. 아름다운 표지가 책 전체의 품질을 증명하지 않는다. 전체 템플릿 계약은 템플릿, API는 [API Reference](#)와 같은 PDF 책에 있다.

템플릿

템플릿은 표현 방식을 정하며 본문을 복제하지 않습니다. Guidedog은 HTML에 Jinja를, PDF에 Typst를 사용합니다.

Guidedog 템플릿은 직접 보고 편집할 수 있는 일반 프로젝트 파일입니다. `guidedog quickstart`를 실행하면 실제 사용할 템플릿이 프로젝트에 생성됩니다.

- `_templates/layout.html`은 Jinja로 HTML 사이트의 구조를 정의합니다.
- `_templates/book.typ`은 Typst로 PDF 책의 레이아웃, 타이포그래피, 표지를 정의합니다.
- `_static/guidedog.css`와 `_static/guidedog.js`는 기본 스타일과 브라우저 동작을 제공합니다.

템플릿은 외부 패키지나 바이너리 캐시에 숨겨지지 않고 프로젝트 안에 있습니다. 직접 편집한 뒤 `guidedog build`로 빌드하고 `guidedog serve`로 확인하세요.

HTML 템플릿

Guidedog은 내장 Jinja 엔진으로 루트 페이지 템플릿 `layout.html`을 렌더링합니다. 하나의 큰 파일로 만들 필요 없이 부분 템플릿, 매크로 라이브러리, 단단계 상속 구조로 나눌 수 있습니다.

모듈형 템플릿 구조

`templates_path`에 지정한 디렉터리 아래에서 템플릿을 여러 파일과 하위 디렉터리로 나눌 수 있습니다. 기본값은 `["_templates"]`입니다.

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

Guidedog은 `templates_path` 아래의 모든 파일과 하위 디렉터리를 검색합니다. 포함하거나 가져올 때는 이 디렉터리를 기준으로 한 상대 경로를 사용합니다.

코드 1 `_templates/layout.html`

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

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

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

Jinja 매크로도 별도 파일에 정의하고 필요한 곳에서 가져올 수 있습니다.

코드 2 `_templates/macros/components.html`

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

코드 3 템플릿에서 매크로 사용

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

!layout.html로 테마 상속

기본 테마의 일부만 바꾸려면 전체 페이지를 다시 작성할 필요가 없습니다. Sphinx의 느낌표 구문으로 내장 레이아웃을 상속할 수 있습니다.

코드 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" %}` 또는 `{% extends "basic/layout.html" %}`은 프로젝트의 `_templates/layout.html`을 재귀적으로 읽지 않고 기본 테마의 템플릿을 불러옵니다.

재정의한 블록에서 `{{ super() }}`를 호출하면 부모 내용을 유지하면서 앞뒤에 내용을 추가할 수 있습니다.

레이아웃 블록

내장 `layout.html`은 Sphinx 관례에 대응하는 블록을 정의합니다.

| 블록 | 용도 |
|-------------|---|
| doctype | 문서 형식 선언입니다. 기본값은 <code><!DOCTYPE html></code> 입니다. |
| htmltitle | <code><head></code> 안의 <code><title></code> 요소입니다. |
| linktags | 탐색 및 메타 정보 링크입니다: <code>favicon</code> , <code>index</code> , <code>search</code> , <code>prev</code> , <code>next</code> . |
| css | 스타일시트 링크와 인라인 CSS 루트 변수입니다. |
| scripts | 검색 색인과 대화형 기능을 포함하는 JavaScript 태그입니다. |
| extrahead | 글꼴, 분석 도구, 사용자 메타 태그를 위한 <code><head></code> 끝의 삽입 위치입니다. |
| header | 주 탐색 표시줄 바로 앞의 헤더 영역입니다. |
| relbar1 | 제품 이름, 버전, 검색, 테마 전환이 있는 상단 탐색 표시줄입니다. |
| rootrellink | 관련 링크 앞의 탐색 삽입 위치입니다. |
| relbaritems | 탐색 표시줄에 추가할 사용자 항목입니다. |
| sidebar1 | 왼쪽 탐색 사이드바 컨테이너입니다. |
| sidebartoc | <code>sidebar1</code> 안의 목차 탐색 트리입니다. |
| breadcrumbs | 본문 위에 표시하는 계층형 이동 경로입니다. |
| document | 본문을 감싸는 컨테이너입니다. |
| body | 현재 문서의 HTML 본문인 <code>{{ body }}</code> 입니다. |
| relbar2 | 이전 장과 다음 장으로 이동하는 하단 탐색 영역입니다. |
| footer | 저작권, 마지막 수정 시각, 소스 링크가 있는 페이지 하단입니다. |
| sidebar2 | 현재 페이지의 목차를 표시하는 오른쪽 보조 영역입니다. |

템플릿 찾기

Guidedog은 `conf.toml` 의 `templates_path` 순서대로 검색한 뒤 내장 템플릿을 찾습니다. 템플릿 이름은 해당 디렉터리 기준의 상대 경로이며 `..` 로 바깥에 접근할 수 없습니다.

`templates_path` 의 모든 파일과 하위 디렉터리는 빌드 의존성입니다. 템플릿을 추가, 수정, 삭제하면 `guidedog build` 와 `guidedog serve` 가 감지해 사이트를 다시 렌더링합니다.

페이지 변수

Sphinx에 변수 이름이 있으면 같은 이름을 사용하므로 Sphinx 테마 일부를 옮길 수 있습니다.

| 변수 | 값 |
|-------------------|---|
| body | HTML 형식의 문서. |
| title | 문서 제목. |
| pagename, docname | <code>usage/install</code> 같은 문서 이름이다. 문서가 아닌 페이지에서는 <code>docname</code> 이 비고 <code>pagename</code> 은 Sphinx처럼 <code>genindex</code> , <code>py-modindex</code> , <code>search</code> 또는 템플릿 페이지 이름이다. |
| toc | 목차 트리에서 만든 HTML 탐색. |

| 변수 | 값 |
|--|---|
| <code>outline</code> | 문서 자체의 절을 HTML로 표시합니다. 절이 없으면 비어 있습니다. |
| <code>prev, next</code> | 읽기 순서의 이전·다음 페이지로 <code>url</code> (<code>link</code> 도 가능)과 <code>title</code> 을 갖습니다. 양 끝에서는 해당 페이지가 없습니다. |
| <code>project, version, release, copyright, language</code> | 같은 이름의 설정값. |
| <code>html_title, docstitle</code> | <code>html_title</code> 이며 기본값은 “<project> <release> documentation”입니다. |
| <code>html_short_title, shorttitle</code> | <code>html_short_title</code> . |
| <code>root_doc, master_doc</code> | 루트 문서 이름. |
| <code>path_to_root</code> | 페이지에서 사이트 루트까지의 경로입니다. 예: <code>../</code> . |
| <code>root_url, search_url, genindex_url</code> | 루트 페이지, 검색 페이지, 전체 색인으로 가는 링크. |
| <code>css_files, js_files</code> | <code>url</code> 이 있는 파일 목록입니다. Guidedog 파일 다음에 <code>html_css_files</code> 와 <code>html_js_files</code> 가 옵니다. |
| <code>logo_url, favicon_url</code> | <code>_static</code> 의 <code>html_logo</code> 와 <code>html_favicon</code> 입니다. 설정하지 않으면 비어 있습니다. |
| <code>accent</code> | 테마 색상 <code>html_theme_options.accent</code> . |
| <code>sourcelink_url</code> | 소스를 표시할 때는 <code>_sources</code> 아래의 문서 소스이며, 아니면 비어 있습니다. |
| <code>last_updated</code> | <code>html_last_updated_fmt</code> 를 설정하면 그 형식의 빌드 날짜이며, 아니면 비어 있습니다. |
| <code>show_copyright, show_sphinx, show_guidedog, has_source, show_source</code> | <code>html_show_*</code> 와 <code>html_copy_source</code> 설정. |
| <code>builder, file_suffix</code> | html 같은 빌더 이름과 페이지 확장자. |
| <code>html_context</code> 의 모든 키 | Sphinx가 Python 값을 넘기듯 타입을 유지한다. <code>false</code> 는 <code>{% if %}</code> 에서 거짓이고 숫자는 숫자, 배열은 목록, 테이블은 <code>conf.toml</code> 키 순서의 딕셔너리다. <code>guidedog migrate</code> 는 <code>conf.py</code> 의 <code>html_context</code> 리터럴 항목을 옮긴다. 계산 값은 미정의로 남아 템플릿에서 거짓으로 읽힌다. |

`path_to`는 Sphinx 템플릿 함수다. `path_to("usage/install")`은 문서 페이지, `path_to("_static/logo.svg", 1)`은 사이트 아래 파일의 URL이며 현재 페이지 기준 상대 경로다. `singlehtml`에서는 문서가 단일 페이지의 절이다. 앞의 호출은 그 페이지에서 `#document-usage-install`, 색인·검색에서는 `index.html#document-usage-install`이 된다. `genindex`나 템플릿 페이지 같은 비문서 이름은 모든 빌더에서 사이트 루트 페이지다.

템플릿으로 만드는 페이지

`html_additional_pages`는 Sphinx처럼 템플릿만으로 페이지를 만든다. 키는 페이지 이름, 값은 `templates_path`의 템플릿이다.

코드 5 conf.toml

```
root_doc = "contents"

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

페이지 템플릿은 보통 layout.html 을 상속하고 블록을 채워 다른 페이지와 같은 모습을 만든다.

코드 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 %}
```

비문서 페이지의 변수를 받는다. pagename 은 페이지 이름(index), title 과 body 는 빈 값이다. pathto, toc, toctree(), html_context 는 그대로 작동한다. html, dirhtml, singlehtml 은 루트의 <name>.html(html_file_suffix)로 매번 쓴다. 템플릿이나 templates_path 의 어떤 파일을 바꿔도 모든 페이지를 다시 만든다.

페이지 위치와 Sphinx와의 차이는 다음과 같습니다.

- 같은 이름의 문서 페이지를 대체하며 Sphinx가 마지막에 쓰는 것과 같다. index = "landing.html" 이면 index.html 이 소개 페이지가 되고 목차 항목은 index.rst 가 제공한다. singlehtml 은 단일 페이지를 유지하고 템플릿 페이지를 쓰지 않으며 경고한다.
- dirhtml 은 genindex.html, search.html 옆에 download.html 을 쓴다. Sphinx는 download/index.html 을 쓴다. pathto("download") 는 그 파일을 반환한다.
- 이름은 사이트 루트의 파일 이름이어야 한다. 링크가 루트 기준이므로 "sub/page" 는 경고(build.additional_page)와 함께 거부한다. genindex 처럼 이미 사용한 이름도 거부한다. Sphinx는 이 경우 한 페이지로 다른 페이지를 덮어쓴다.
- guidedog migrate 는 conf.py 의 html_additional_pages 를 옮깁니다.

오류

템플릿 오류는 빌드를 멈추고 파일, 줄 번호, 해당 줄, 수정 제안을 표시합니다.

```
TEMPLATE ERROR                                     template.error
_templates/base.html:1
No filter named 'default'.
Did you mean the filter 'default'?
```

정의하지 않은 변수는 Jinja처럼 아무것도 출력하지 않습니다.

과도한 템플릿도 충돌하지 않고 오류로 멈춘다. 구문(괄호, 태그, 연산자, 필터·elif 사슬)이 100단계, 매크로·포함·재귀 루프가 200단계를 넘거나 재귀 스택이 512 KiB를 넘을 때다. 자신을 포함한 목록은 Python처럼 [...]로 표시한다.

PDF 템플릿

PDF 책은 Typst로 조판한다. `_templates/book.typ`는 `book` 함수를 정의하는 일반 Typst 파일이다. Guidedog은 책 소스를 다음과 같이 쓴다.

```
#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
```

책의 모양을 결정하는 모든 요소가 이 파일에 있습니다. 글꼴, 페이지 크기와 여백, 제목, 표제지, 머리말, 목차입니다.

매개변수

매개변수	값
<code>title</code>	<code>pdf_documents</code> 의 책 <code>title</code> 이며, 없으면 <code>project</code> 입니다.
<code>author</code>	책의 <code>author</code> 이며, 없으면 설정의 <code>author</code> 입니다.
<code>version</code>	<code>release</code> .
<code>date</code>	<code> today </code> 는 설정한 <code>today</code> 값이며, 없으면 <code>today_fmt</code> 형식의 빌드 날짜입니다.
<code>lang</code>	<code>language</code> .
<code>paper</code>	"a4"이며 <code>pdf_paper_size</code> 가 <code>letter</code> 이면 "us-letter"입니다.
<code>numbering</code>	<code>toctree</code> 에 <code>numbered</code> 옵션이 있으면 <code>true</code> 입니다.
<code>logo</code>	<code>pdf_logo</code> 의 경로입니다. 설정하지 않으면 생략합니다.
<code>copyright</code>	판권 페이지의 <code>copyright</code> 이며, 이 매개변수를 받는 템플릿에만 전달합니다.
<code>body</code>	각 장의 내용.

템플릿은 기본값이 있는 매개변수를 추가할 수 있다. `quickstart` 템플릿에는 `accent`, 글꼴 `serif·sans·mono`, 글자 `size`, `page-ref`가 있다. 마지막 값을 `n => [p. #n]` 같은 함수로 설정하면 다른 페이지 참조에 페이지 번호를 붙인다.

모듈형 Typst 템플릿

Typst 템플릿도 한 파일에 넣을 필요가 없습니다. 스타일, 매크로, 표지를 `_templates/` 아래의 여러 `.typ` 파일로 나누고 `#import`와 `#include`로 조합할 수 있습니다.

코드 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
}
```

템플릿에서 쓸 수 있는 것

Typst 패키지

`#import "@preview/cetz:0.4.2"` 등 패키지는 처음 사용할 때 다운로드된다. `pdf_packages = "offline"`이면 디스크에 있는 패키지만 사용한다.

글꼴

`typst` 명령처럼 시스템 글꼴을 쓸 수 있다. `pdf_font_paths`로 폴더를 추가한다. `pdf_fonts = "embedded"`는 Typst 내장 글꼴만 써서 바이트 단위로 재현 가능한 책을 만든다.

프리앰블

`pdf_preamble`은 `#show: book` 다음에 넣을 Typst 파일을 지정합니다. 별도 템플릿 없이 몇 가지 `set.show` 규칙을 추가할 때 적합합니다.

문서 안의 Typst

문서는 `reStructuredText`와 `Markdown`이며 Typst 파일은 문서가 아니다. 책에 Typst 마크업을 넣으려면 `raw` 블록을 쓴다. 웹 페이지에서는 생략된다.

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

`.. only:: pdf`도 일반 콘텐츠를 책에만 남깁니다.

Typst 오류는 템플릿이나 생성된 책 어디서든 파일과 줄 번호를 밝힌다. PDF별 소스는 `_build/pdf/sources/<pdf-filename>.typ`에 있다. `reference-en.pdf`의 경우 `sources/reference-en.pdf.typ`이다. 한 권 빌드에는 `_build/pdf/book-<language>.typ` 확인용 사본도 남긴다. 실패하면 게시하지 않고 오류에 명시된 `_build/.doctrees/failed` 경로에 소스를 보관한다. 여러 책의 실패 소스는 구분한다.

새 템플릿 작성

각 부분의 사용법을 보여 주는 `quickstart` 템플릿부터 시작하세요.

1. 기본 스타일시트가 기대하는 `layout.html` 마크업을 유지하거나 `_static/guidedog.css`도 함께 바꾸세요.

2. 반복 부분을 별도 파일로 옮겨 `include`하거나, `base.html`을 만들고 `layout.html`에서 `extends`하세요.
3. 책은 `book` 함수의 규칙을 바꾸거나 같은 이름과 매개변수의 새 함수를 작성하세요.

편집 중에는 `guidedog serve`로 미리 보세요. 소스를 저장하고 브라우저를 새로 고치면 서버가 변경된 입력을 다시 빌드한 뒤 페이지를 제공합니다.

이미지, 동영상, 오디오, 다운로드

자원은 이를 설명하는 문서 가까이에 두세요. 상대 경로를 쓰거나 `/`로 시작해 소스 루트를 기준으로 지정하세요. Guidedog는 문서에서 쓰는 자원을 복사합니다. 동영상은 웹에서 재생되며 PDF에는 녹화 내용을 표시합니다.

이미지

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

.. figure:: /screenshots/search.png

   Searching the site.
```

페이지는 이미지를 `_images`로 복사하고 책은 포함합니다. PNG, JPEG, GIF, SVG, WebP는 둘 다 지원합니다. 이미지가 없으면 경고하며, 페이지는 대체 텍스트를 남기고 책은 이미지를 제외합니다.

동영상과 오디오

동영상(`.mp4`, `.webm`, `.ogv`, `.mov`, `.m4v`) 또는 오디오(`.mp3`, `.ogg`, `.oga`, `.m4a`, `.opus`, `.wav`, `.flac`, `.aac`)를 지정한 `image`나 `figure`는 미디어를 재생합니다. Docutils의 동영상 처리와 같은 방식입니다.

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

   The two-minute tour.

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

페이지는 브라우저 플레이어와 재생하지 못하는 브라우저용 파일 링크를 표시합니다. 책은 미디어를 재생할 수 없으므로 플레이어 자리의 상자에 녹화·녹음을 표시합니다. 텍스트 출력은 `[video: ...]` 또는 `[audio: ...]`를 씁니다.

다운로드

`download` 역할은 어떤 파일이든 연결하고 `_downloads`로 복사합니다.

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

테마 파일

`_static`에는 테마의 스타일시트, 스크립트, 글꼴, 로고를 두며 사이트의 `_static`으로 그대로 복사합니다. `html_static_path`로 디렉터리를 지정하고,

html_css_files, html_js_files, html_logo, html_favicon으로 파일을 지정합니다. 콘텐츠 미디어는 문서와 함께 두어 같이 이동해도 링크가 유지되게 하세요.

국제화

구조를 유지하며 설명을 번역하세요. Guidedog는 메시지 카탈로그를 추출하고 번역을 병합하여 선택한 언어로 빌드합니다. 원문이 기준이며, 누락된 번역은 내용을 만들어 내지 않고 원문으로 돌아갑니다.

작업 흐름

1. gettext 빌더로 메시지 템플릿을 만드세요.

```
guidedog build gettext
```

_build/gettext에 텍스트 도메인별 템플릿(.pot)을 만든다. 문서의 모든 문단, 제목, 목록 항목, 표 셀, 캡션, 알림, 용어, 필드가 메시지가 된다. 원본 파일과 줄 번호도 함께 기록된다.

2. sphinx-intl update처럼 언어별 카탈로그를 만들거나 갱신하세요.

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

카탈로그는 locales/de/LC_MESSAGES/*.po에 생성된다. 다시 실행하면 msgmerge처럼 새 템플릿을 병합한다. 기존 번역은 유지하고, 원문이 바뀐 번역은 검토하도록 fuzzy로 표시한다. 문서에서 사라진 메시지는 끝에 사용 중단 항목(#~)으로 남겨 두어 나중에 번역을 복원할 수 있다.

3. 각 msgstr에 번역을 넣으세요. 번역이 맞는지 확인한 뒤 #, fuzzy 줄을 지우세요. fuzzy 번역은 사용하지 않습니다.

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

4. sphinx-intl stat처럼 번역 진행 상황을 확인하세요.

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

5. 해당 언어로 빌드하세요.

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

또는 conf.toml에서 language = "de"를 설정하세요.

책 파일 이름에는 언어 접미사가 붙는다. manual.pdf는 manual-de.pdf가 된다. 빌드마다 완전한 출력 세대를 게시하므로 여러 판을 보관할 때는 대상 폴더를 따로 쓴다. 예를 들어 독일어는 guidedog build -b pdf docs build/books/de -D language=de, 영어는 guidedog build -b pdf docs build/books/en -D language=en으로 만든다.

다른 입력과 마찬가지로 카탈로그가 바뀐 문서만 다시 읽습니다. 카탈로그를 추가하거나 지우면 모든 문서를 다시 읽습니다.

guidedog intl은 sphinx-intl의 옵션을 받는다. -p DIR은 템플릿 위치(기본 _build/gettext), 반복 가능한 -l LANG은 언어(기본 프로젝트 언어), -d DIR은

카탈로그 위치(기본 `locale_dirs`의 첫 항목), `-w N`은 줄 너비(기본 76)이다. `--no-obsolete`도 지원한다.

번역 작성

메시지는 마크업을 포함한 단락, 제목, 기타 요소의 원문입니다. `.rst`는 reStructuredText, `.md`는 Markdown입니다. 번역도 같은 문법으로 읽으므로 원문처럼 강조, 링크, 참조를 쓸 수 있습니다.

참조는 그대로 유지해야 합니다. 모든 참조의 표기를 보존하고, 주변 텍스트와 표시 제목만 바꿀 수 있습니다.

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

번역에서 참조가 빠지면 Sphinx처럼 경고(inconsistent references in translated message)를 내고 번역을 사용한다. 원문에 없는 참조를 추가하면 이를 해결할 수 없어 경고와 함께 원문을 유지한다. 리더가 번역의 마크업을 거부할 때도 원문을 유지하고 경고에 해당 번역을 표시한다.

절의 앵커는 언어에 따라 바뀌지 않습니다. 번역된 제목도 원문의 앵커를 유지하므로 내부·외부 링크가 계속 동작합니다.

설정

표준 설정은 Sphinx의 이름과 기본값을 따릅니다. Guidedog는 번역하지 않을 참조 부록을 위한 추출 전용 제외 설정도 제공합니다.

설정	의미
<code>language</code>	빌드할 언어입니다. 예: <code>"de"</code> , <code>"pt_BR"</code> .
<code>locale_dirs</code>	소스 폴더 기준 카탈로그 위치다. 각 위치에 <code>LANGUAGE/LC_MESSAGES/DOMAIN.po</code> (또는 컴파일한 <code>.mo</code>)를 둔다. 기본값은 <code>["locales"]</code> 이다. 여러 위치가 있으면 해당 메시지를 번역하는 첫 카탈로그를 사용한다.
<code>gettext_compact</code>	문서 메시지가 들어갈 카탈로그를 정한다. <code>true</code> (기본값)이면 최상위 문서는 각각 하나를 쓰고 같은 폴더의 문서는 공유한다(<code>guide/usage.rst</code> 는 <code>guide.po</code>). <code>false</code> 이면 문서마다 하나를 만든다. 이름을 지정하면 모든 문서가 그 이름의 카탈로그를 쓴다.
<code>gettext_location</code>	메시지 위치(<code>#: ../../index.rst:12</code>)를 기록합니다. 기본값은 <code>true</code> 입니다.
<code>gettext_uuid</code>	메시지가 나타나는 곳마다 식별자를 기록합니다. 문서, 메시지, 출현 위치로 만들므로 빌드마다 같습니다.
<code>gettext_auto_build</code>	각 <code>.po</code> 를 옆의 <code>.mo</code> 로 컴파일하여 다른 도구에서 쓰게 합니다. 기본값은 <code>true</code> 입니다. Guidedog는 <code>.po</code> 를 직접 읽습니다.
<code>gettext_additional_targets</code>	추가 번역 대상: <code>index</code> (색인 항목), <code>literal-block</code> , <code>doctest-block</code> , <code>raw</code> , <code>image</code> (대체 텍스트).
<code>gettext_allow_fuzzy_translations</code>	<code>uzzy</code> 번역도 사용합니다. 기본값은 <code>false</code> 입니다.

설정	의미
<code>gettext_exclude_patterns</code>	POT 추출에서만 제외하는 문서 경로 패턴입니다. 예: <code>["api/**"]</code> . 기본값은 <code>[]</code> 이며, 문서는 HTML과 PDF에 남습니다.
<code>gettext_last_translator</code> , <code>gettext_language_team</code>	템플릿 헤더의 Last-Translator와 Language-Team.
<code>figure_language_filename</code>	빌드 언어에 맞는 이미지 파일이다. 기본값은 <code>"{root}.{language}{ext}"</code> 이다. <code>language = "de"</code> 이면 파일이 있을 때 <code>img/logo.png</code> 대신 <code>img/logo.de.png</code> 를 쓴다. 필드는 <code>{root}</code> , <code>{path}</code> , <code>{basename}</code> , <code>{ext}</code> , <code>{docpath}</code> , <code>{language}</code> 이다.
<code>translation_progress_class</code>	번역된 요소에는 <code>translated</code> , 나머지는 <code>untranslated</code> 클래스를 붙인다(<code>true</code>). 둘 중 하나만 붙일 수도 있다(<code>"translated"</code> 또는 <code>"untranslated"</code>). 스타일시트로 남은 부분을 표시할 수 있다.

Guidedog의 인터페이스 문구

Guidedog 자체도 알림 제목, "Added in version 2.1", "Fig. 1", "Contents", 레이아웃의 "Previous", "Next", "Navigation", "Search", 검색.색인 페이지, 책의 "Version" 같은 문구를 만든다. 이 문구는 `guidedog` 텍스트 도메인에서 다음 순서로 찾는다.

1. 프로젝트의 `locales/LANGUAGE/LC_MESSAGES/guidedog.po`.
2. Sphinx처럼 프로젝트의 `locales/LANGUAGE/LC_MESSAGES/sphinx.po`.
3. Guidedog 내장 카탈로그는 독일어, 스페인어, 프랑스어, 일본어, 브라질 포르투갈어, 러시아어, 중국어 간체를 지원합니다.

메시지 식별자는 영어다. Sphinx에 같은 문구가 있으면 식별자도 같으므로 프로젝트의 `sphinx.po`를 계속 쓸 수 있다. 자체 `guidedog.po`로 문구를 바꾸거나 언어를 추가할 수 있다.

템플릿은 Sphinx처럼 `_()`, `gettext()`, `ngettext()`로 번역합니다.

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

`conf.toml`에서 지정하지 않으면 `numfig_format`, `html_title`, `html_short_title`는 언어를 따릅니다.

날짜

`|today|`, 책의 날짜, 페이지의 `last_updated`는 Sphinx와 같은 규칙을 쓴다. `today_fmt`와 `html_last_updated_fmt`는 `strftime` 형식이고 월.요일 이름은 `language`를 따른다. `%b %d, %Y`는 영어로 "Sep 29, 2026", 독일어로 "Sept. 29, 2026"이다. 빈 형식은 `_( '%b %d, %Y' )`이며 카탈로그에서 번역할 수 있다. 앞서 나열한 언어의 이름은 Unicode CLDR에서 가져온다. 다른 언어는 Babel이 모르는 언어에 대한 Sphinx처럼 영어를 사용한다.

날짜는 현지 시간대의 날짜다. `SOURCE_DATE_EPOCH`(1970년부터의 초, UTC)을 설정하면 모든 컴퓨터에서 같은 날짜로 재현 가능한 빌드를 할 수 있다.

다른 프로젝트로 연결

`sphinx.ext.intersphinx`를 사용하면 이 프로젝트가 해석하지 못한 참조를 다른 프로젝트의 인벤토리에서 찾습니다. 각 Sphinx 또는 Guidedog 사이트가 게시하는 `objects.inv`입니다. `conf.toml`에 프로젝트를 지정하세요.

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

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

각 항목에는 다른 프로젝트의 페이지 주소와 인벤토리 위치를 적습니다. URL, 소스 폴더 기준의 파일 경로, 또는 주소 아래의 `objects.inv`를 뜻하는 ""를 사용할 수 있습니다. 목록은 순서대로 시도합니다. Sphinx의 `intersphinx_mapping`과 같지만 Python의 `None` 대신 ""를 쓰며, `guidedog migrate`가 변환합니다.

이후 참조에는 새로운 문법이 필요하지 않습니다.

```
: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.
```

레이블과 문서 링크는 다른 프로젝트의 제목을 표시하며, “(in Python v3.14)” 같은 도움말을 붙입니다. Sphinx처럼 `:doc:` 참조에는 프로젝트 이름 (`:doc:python:tutorial/index``)이 필요합니다. `intersphinx_disabled_reftypes`의 기본값이 `["std:doc"]`이기 때문입니다.

빌드 시작 시 인벤토리를 가져와 `_build/.doctrees/_intersphinx_cache_`에 `intersphinx_cache_limit`일 동안 보관합니다. 기본값은 5일이며 음수이면 계속 보관합니다. 로컬 파일은 매번 읽습니다. Sphinx처럼 URL 리디렉션 후의 디렉터리를 링크 기준으로 사용하고 캐시에 기억하므로 캐시에서도 같은 링크를 만듭니다. 예를 들어 `https://werkzeug.palletsprojects.com/objects.inv`는 `/en/stable/`로 이동합니다. `intersphinx_timeout`은 다운로드별 제한 시간(초)입니다. 접근할 수 없는 프로젝트는 경고하며 참조는 해석되지 않은 채로 남습니다.

Guidedog는 모든 페이지, 레이블, 용어, 객체를 담은 `objects.inv`를 기록하므로 다른 프로젝트도 같은 방식으로 연결할 수 있습니다.

객체 타입 선언

표준 도메인에 없는 이름이 필요할 수 있습니다. TOML에서 객체 타입을 선언하면 Guidedog가 Python 확장 없이 참조 대상, 색인 항목, 역할을 만듭니다. 선언은 문법을 나타내며 실행 동작은 아닙니다.

참조 대상: `add_crossref_type`

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

이후 `.. setting:: DEBUG`는 ID가 `setting-DEBUG`인 대상과 색인 항목을 만들고, `:setting: `DEBUG``는 Sphinx처럼 이를 연결합니다. `index`는 Sphinx의

indextemplate에 해당합니다. 첫 콜론 앞에는 종류(single, pair, triple, see, seealso), 뒤에는 항목을 적으며 %s는 이름입니다.

설명: add_object_type

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

.. django-admin:: migrate [app_label]는 일반 객체처럼 시그니처, 본문, ID(django-admin-migrate), 색인 항목으로 명령을 설명합니다. Sphinx는 parse_node 함수로 시그니처를 읽을 수 있지만, 일반적인 처리는 다음 필드로 지정합니다.

name

"whole"(기본값)은 전체 시그니처를 객체 이름으로 씁니다. "first-word"는 첫 공백 앞까지를 씁니다. 예를 들어 pdb의 b(reak) [lineno]는 b(reak)라는 이름을 갖습니다.

display

시그니처의 표시 방식이며 %s가 시그니처입니다. 기본값은 원문 그대로입니다.

program

true이면 이름이 이후 옵션이 속할 프로그램이 됩니다. .. program::와 같습니다.

지시문의 다른 이름

확장은 Sphinx의 지시문을 다른 이름으로 등록할 수 있습니다.

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

도메인이 없는 대상은 표준 지시문이나 기본 도메인의 지시문입니다. py:function 같은 대상은 도메인을 명시합니다.

마이그레이션

guidedog migrate는 conf.py에 나열된 프로젝트 내부 확장 모듈을 찾습니다. add_crossref_type, add_object_type, add_directive 호출을 실행하지 않고 읽어 표를 작성합니다. 선언할 수 없는 parse_node 함수(name, display, program으로 맞추어야 함)와 Python 지시문, 역할, 이벤트 처리기는 보고합니다.

autodoc으로 Python 문서 작성하기

Guidedog는 소스 코드를 읽어 Python 문서를 생성합니다. 패키지를 import하지 않습니다. docstring과 시그니처는 문서 그래프에 포함됩니다. 실행 중에만 생성되는 객체는 명시적으로 설명해야 합니다. 이는 의도적으로 정한 경계입니다.

설정하기

확장을 지정하고 소스 폴더를 기준으로 패키지의 위치를 Guidedog에 알려 주세요. Python에서 `sys.path`를 설정하는 것과 같습니다:

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

`guidedog migrate`는 `sys.path.insert(0, os.path.abspath('../'))`처럼 `conf.py`에서 패키지 경로를 설정하는 줄을 읽고 `autodoc_source_paths`를 작성합니다. 다른 위치에 설치된 패키지는 해당 폴더를 추가해야 찾을 수 있습니다. 예를 들어 의존 패키지의 소스 체크아웃 폴더를 지정할 수 있습니다. Guidedog는 Python의 `site-packages`를 검색하지 않습니다.

이제 지시문을 Sphinx와 같은 방식으로 사용할 수 있습니다:

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

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

Sphinx 9.1에서 지원하는 모든 옵션을 같은 의미로 사용할 수 있습니다: `: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:`. `autodoc_default_options`를 취소하는 `no-` 형식도 지원합니다.

설정

이 `conf.toml` 설정은 Sphinx와 이름, 값, 기본값이 같습니다.

autodoc_source_paths

Guidedog 전용 설정입니다. Python 모듈을 찾을 디렉터리를 소스 디렉터리 기준의 상대 경로로 검색 순서대로 지정합니다. 이 디렉터리 아래의 파일만 읽으며, 밖으로 이어지는 심볼릭 링크는 거부합니다.

autoclass_content

클래스를 설명할 docstring을 지정합니다. `"class"`(기본값), `"init"`, `"both"` 중에서 선택합니다.

autodoc_class_signature

`"mixed"`(기본값) 또는 `"separated"`입니다. 후자는 `__init__`를 별도 메서드로 문서화합니다.

autodoc_default_options

모든 지시문에 적용할 옵션 표입니다. 예를 들어 `members = true` 또는 `member-order = "bysource"`를 지정합니다. 값이 `false`이면 해당 옵션을 제외합니다.

autodoc_docstring_signature

`true`(기본값): docstring의 첫 줄이 `name(args) -> result` 같은 형식이면 시그니처로 사용합니다.

autodoc_inherit_docstrings

true(기본값): docstring이 없는 멤버는 기반 클래스의 docstring을 상속합니다.

autodoc_member_order

"alphabetical"(기본값), "groupwise", "bysource" 중에서 선택합니다.

autodoc_preserve_defaults

false(기본값): 기본값을 repr() 표현으로 표시합니다. true이면 소스에 쓰인 형태를 유지합니다.

autodoc_typehints

"signature"(기본값), "description"(:type: 및 :rtype: 필드 사용), "both", "none" 중에서 선택합니다.

autodoc_typehints_description_target

"all"(기본값), "documented", "documented_params" 중에서 선택합니다.

autodoc_typehints_format

"short"(기본값) 또는 "fully-qualified"입니다.

autodoc_use_type_comments

true(기본값): # type: 주석을 타입 어노테이션으로 취급합니다.

strip_signature_backslash

Sphinx처럼 시그니처의 역슬래시를 두 번 씩입니다.

autodoc_mock_imports, autodoc_type_aliases, autodoc_warningiserror

설정은 허용됩니다. 모듈을 임포트하지 않으므로 모킹이 필요하지 않습니다. 타입 별칭은 적용하지 않습니다.

Guidedog가 Python의 동작을 판단하는 방법

이름은 Python의 바인딩 규칙에 따라 해석합니다. 패키지의 `__init__.py`에 있는 `from .app import Flask`는 `flask.Flask`를 `flask/app.py`에 정의된 클래스로 연결합니다. autodoc은 이를 `:canonical: flask.app.Flask`로 표시합니다. 기반 클래스도 같은 방식으로 찾고, 메서드 결정 순서는 Python과 같은 규칙으로 계산합니다. 상속된 멤버는 기반 클래스의 소스에서 가져옵니다. `if TYPE_CHECKING:` 안에서만 임포트한 이름은 실행 시 존재하지 않으므로, 해당 이름을 쓰는 어노테이션은 Sphinx와 마찬가지로 원문을 유지합니다.

docstring은 Python이 저장할 문자열과 같습니다. 이스케이프를 처리하고 Python 3.13 컴파일러와 같은 규칙으로 들여쓰기를 제거합니다. 속성 설명은 Sphinx 분석기와 같은 위치에서 읽습니다. 대입문 뒤나 바로 위의 `#:` 주석, 그리고 대입문 바로 다음의 문자열 리터럴입니다. 시그니처는 `def`에서 가져옵니다. 클래스는 Python의 탐색 규칙에 따라 메타클래스의 `__call__`, `__new__`, `__init__`를 사용합니다. `dataclass`와 `NamedTuple`은 필드를 사용합니다. `@overload` 변형은 구현의 시그니처를 대신합니다.

Python을 실행하지 않고는 알 수 없는 정보

소스만으로 Python의 동작을 알 수 없으면 Guidedog는 해당 객체의 이름을 밝혀 알립니다. 문서화할 수 없으면 경고를 냅니다. Sphinx보다 적은 정보로 문서화했다면 `-v`에서 볼 수 있는 안내를 남깁니다.

- 컴파일된 모듈(.so, .pyd)에는 읽을 소스가 없으므로 객체를 찾을 수 없습니다.
- 코드를 실행해 얻는 값(`app = Flask(__name__)`, `now = datetime.now()`)에는 `:value:`가 없습니다. 계산이 필요한 기본값은 소스의 표기를 유지합니다.
- 소스 경로 밖의 데코레이터(`@click.command()`)가 만든 함수는 데코레이션 전의 함수를 기준으로 문서화합니다.
- 소스 경로 밖의 패키지에 있는 클래스는 임포트한 경로로 이름을 표시합니다. 이 경로는 실제 정의 모듈과 다를 수 있습니다. 멤버, 생성자 시그니처, 상속할 docstring은 알 수 없습니다.
- 내장 타입의 docstring은 알 수 없습니다. 따라서 자체 docstring 없이 dict의 docstring을 상속할 메서드는 제외합니다.
- autodoc 이벤트(`autodoc-process-docstring`, `autodoc-skip-member` 등)를 처리하는 Python 확장은 실행하지 않습니다.
- 실제 코드 수준을 훨씬 넘는 깊은 중첩은 분석하지 않고 `autodoc.too_deep` 경고를 냅니다. Python 자체도 200단계의 괄호 중첩을 거부합니다. Guidedog는 수백 개 피연산자로 이어진 + 연산도 거부합니다.
- 처리가 계속 늘어날 수 있는 작업에는 한도가 있습니다. 임포트와 별칭을 따라 이름을 찾을 때 한 경로는 48단계, 전체는 100,000단계까지입니다. 타입 별칭과 상수는 10,000단계까지 확장하며, 그 뒤에는 소스 표기를 유지합니다. 클래스의 MRO는 100개 클래스까지 추적합니다. 지시문 하나가 생성하는 지시문은 최대 10,000개이며, 초과하면 `autodoc.limit` 경고를 냅니다. 마지막 한도에 도달하는 것은 자신의 이름으로 자신을 포함하는 클래스입니다.

누락된 패키지의 소스 디렉터리를 `autodoc_source_paths`에 추가하면 이러한 제약 대부분을 해결할 수 있습니다.

다시 빌드하기

autodoc 지시문이 읽은 모든 Python 파일은 페이지의 의존성으로 기록됩니다. 파일을 수정하면 다음 빌드에서 페이지를 다시 읽습니다. 객체를 찾지 못한 페이지는 매번 다시 읽으므로, 소스에 객체가 추가되면 바로 인식합니다.

보안

문서 빌드에서는 프로젝트 코드를 실행하지 않습니다. Guidedog는 Python 소스를 텍스트로 읽고 `include`와 같은 경로 제한을 적용합니다. `autodoc_source_paths` 아래만 읽으며, 밖으로 이어지는 링크는 따라가지 않습니다.

Odin 문서화

Odin 도메인은 패키지, 프로시저, 타입, 멤버에 명명된 대상을 제공한다. Guidedog은 Odin 소스를 파싱해 API 페이지도 생성하지만 컴파일러나 패키지를 실행하지 않는다. 생성된 시그니처에도 유용한 계약이 필요하다.

객체 설명

패키지는 Python 모듈처럼 뒤따르는 내용의 문맥을 정한다.


```

.. 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

```

시그니처는 Odin 선언처럼 한 줄로 쓰고 그대로 표시한다. 아래 예제는 다음과 같이 보인다.

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

```
Circle = 1
```

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

정사각형을 측정합니다.

매개변수: **side** – 변의 길이.

반환값: • **a** – 넓이.
 • **ok** – 종류를 아는지 여부.

지시문과 받는 시그니처:

odin:package와 **odin:currentpackage**

core:fmt 또는 프로젝트 루트 아래 경로인 readers/sphinx를 쓴다. 객체 이름은 패키지 가져오기 경로를 쓴다(core:fmt.println). odin:currentpackage는 대상 없이 문맥만 바꾸고 None은 해제한다. 옵션은 :synopsis:, :platform:, :deprecated:, :no-index:, :imports: 다. 마지막은 alias=package 쌍(gd=core rst=readers/rst)으로 시그니처 별칭을 지정해 gd.Node_Id를 core.Node_Id에 연결한다. :private:는 비공개 타입과 프로시저 그룹의 비공개 멤버(State Rules catalog_text)를 나열한다. 연결할 설명이 없어 텍스트로 표시하며 생성 페이지가 자동으로 목록을 만든다.

odin:procedure (또는 **odin:proc**)

name :: proc(params) -> results에 이름 앞 속성(@(require_results)), proc 앞 #force_inline, 호출 규약(proc "c" (...)), 기본 인수(x := 1, x: int = 1), \$T 다형 인수, .. 가변 인수, using, #c_vararg, 이름 있는 결과, 태그(#optional_ok), where 절을 지원한다. name(params) -> results는 축약형이다.

odin:procgroupp

name :: proc{a, b}: 각 멤버는 해당 프로시저로 연결된다.

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 u32를 쓴다.
 odin:type에는 구별 타입과 별칭(Handle :: distinct uintptr, Callback :: proc(x: int) -> bool)을 쓴다. 타입 내용에는 멤버가 들어간다.

odin:field와 odin:enumerator

타입 안에서는 name: Type에 태그(name: string `json:"n"`), 비트 크기 (low: u8 | 3)를 붙인다. 열거자는 Name 또는 Name = 3이다. 이름은 Type.member다.

odin:const, odin:var, odin:foreign

NAME :: 64 또는 NAME : int : 64; name: Type, name := value 또는
 name: Type = value; 외부 가져오기는 libc "system:c"이다.

모든 객체는 Sphinx의 :no-index:, :no-index-entry:, :no-contents-entry:, :no-typesetting:을 받는다. 추가로 :package: (다른 패키지에서 설명), :private:와 :deprecated: message(시그니처에 없으면 @(private)·@(deprecated="message") 표시), :availability: Windows, Linux(첫 줄에 플랫폼 표시), :foreign: libc(외부 블록 프로시저)가 있다. 여러 시그니처 줄은 대상별로 다른 한 객체를 설명하며 첫 줄만 ID와 색인 항목을 얻는다.

타입 위치의 이름은 시그니처의 패키지.타입에서 먼저 찾고 원래 이름으로 찾아 연결한다. 내장 타입(int, string, rawptr, typeid, any 등), 키워드, 다형 이름(\$T와 이후 T, 외부 타입 포함), 배열 길이, 기본값, 상수, 태그, where 절은 그대로 표시하고 링크하지 않는다.

역할과 ID

:odin:pkg:, :odin:proc:, :odin:type: (구조체·공용체·열거·비트 집합·비트 필드 등), :odin:const:, :odin:var:, :odin:field:, :odin:enumerator:, :odin:obj: (모든 객체)로 링크한다. Python처럼 ~는 마지막 부분만 표시한다 (:odin:proc:~core:fmt.println`는 println). 앞의 .은 이름 끝을 찾는다. 참조의 타입.패키지, 원래 이름, 경로의 :./ 뒤 순서로 찾는다. fmt.println는 core:fmt.println를, sphinx.Config는 readers/sphinx.Config를 찾아 Odin의 마지막 경로 이름 사용과 같다. :imports: 별칭을 우선한다. default-domain:: odin 또는 primary_domain = "odin"이면 odin:을 생략한다.

객체 ID는 odin-과 전체 이름에 Sphinx의 make_id를 적용한 odin-core-fmt.println, odin-readers-sphinx.Config.docname이다. Python처럼 대소문자와 점을 유지하고 경로의 :./는 하이픈으로 바꾼다. URL에서 읽기 쉽고 빌드마다 같으며 마지막 이름이 같은 패키지도 구별한다. 패키지 ID는 odin-package-와 경로다. 색인은 "println (procedure in core:fmt)"처럼 표시한다. objects.inv는 odin 도메인과 타입(odin:procedure, odin:struct, odin:field 등)으로 모든 객체를 나열하며 Sphinx 모듈처럼 패키지를 먼저 둔다.

내용의 정보 필드는 다른 도메인처럼 묶는다. :param name:와 :type name:, 이름 있는 결과의 :result name:, :returns:, :rtype:가 있다.

소스에서 생성

소스 폴더 기준 패키지 위치를 지정한다.

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

odin_autoapi_dirs 아래 패키지는 상대 경로로 이름을 붙인다(src/shapes/round는 shapes/round). 컬렉션은 Odin 가져오기 이름(core:strings)을 쓴다. 생성 지시문은 autodoc처럼 작동한다.

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

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

odin:autopackage는 패키지와 문서 주석을 출력하고 :members:로 전체 또는 지정 선언도 출력한다. odin:autoproc, odin:autoprocgrou, odin:autotype, odin:autoconst, odin:autovar는 현재 패키지 이름 또는 package.name의 선언 하나를 쓴다. 옵션은 :members:, :undoc-members:, :private-members:(기본 제외되는 @private), :exclude-members:, :member-order:(기본 source는 파일 이름과 위치순, alphabetical, groupwise는 "Types", "Procedures", "Procedure groups", "Constants", "Variables", "Foreign imports"로 분류), :no-index: 다.

문서 주석은 선언 바로 위의 // 줄이나 /* */ 블록, 필드.열거자 줄 끝의 주석이다. 일반 텍스트를 다음 규칙으로 reStructuredText로 바꾼다.

- 문단은 문단으로, - item 행 목록은 목록으로 유지된다.
- 빈 줄이나 콜론으로 끝나는 줄(Example:) 뒤의 들여쓴 블록은 code-block:: odin이며 Output: 뒤는 text다. 문단 줄 바로 뒤의 텍스트보다 더 들여쓴 줄은 문단을 이어 간다.
- Inputs: 뒤 - name: text는 :param name: 필드, Returns: 항목은 :result name:(이름이 없으면 :returns:)이 된다. Odin 코어 라이브러리 형식을 따른다.
- NOTE: 또는 WARNING:로 시작하는 줄은 참고 또는 경고가 된다.
- `code`는 리터럴 텍스트다. reStructuredText가 마크업으로 읽는 나머지 문자(*, |, 단어 끝 _ 등)는 이스케이프하므로 주석이 그대로 보이며 경고하지 않는다.

API 페이지

odin_autoapi_dirs를 설정하면 각 패키지에 sphinx-autoapi처럼 읽기 중 페이지를 생성한다. api/shapes에는 odin_autoapi_options의 odin:autopackage:: shapes가, api/index에는 패키지와 요약이 있다. toctree, 검색, 전체 색인, singlehtml, PDF에 포함되는 프로젝트 문서지만 소스 폴더에는 쓰지 않는다. 설정은 다음과 같다.

odin_autoapi_dirs

페이지를 만들 패키지 폴더이며 소스 기준 경로다. 이 폴더와 odin_collections 아래의 파일만 읽고 다른 곳으로 향하는 링크는 거부한다.

odin_collections

core:strings 같은 지시문 인수용 컬렉션 이름과 폴더 표다. 이 패키지에는 페이지를 만들지 않는다.

odin_autoapi_root

페이지 폴더이며 기본값은 "api"다. 같은 이름의 소스 문서가 있으면 그 문서를 유지하고 경고한다.

odin_autoapi_options

기본값은 ["members", "undoc-members"]이며 private-members를 추가할 수 있다.

odin_autoapi_member_order

"source"(기본값), "alphabetical" 또는 "groupwise".

odin_autoapi_add_toctree_entry

true(기본값): api/index를 루트 문서의 첫 toctree에 추가한다.

odin_autoapi_generate_api_docs

true(기본값). false면 지시문을 위해서만 폴더를 읽는다.

문서화하지 않는 패키지의 타입(core.io.Writer)은 링크할 수 없다. -n에서는 nitpick_ignore_regex = ["odin:type", "(core|base):.*"]로 억제한다. Guidedog의 docs/api도 이 방식이다.

플랫폼

Odin과 같은 파일 판단으로 패키지의 빌드 대상을 문서화한다. *_windows.odin, *_linux_arm64.odin, *_amd64.odin은 해당 대상만 포함한다. #+build linux, darwin, #+build !windows(구형 //+build도)가 범위를 좁힌다. 파일의 when ODIN_OS == .Windows가 ODIN_OS·ODIN_ARCH만 비교하면 각 분기와 else를 대상별로 구분한다. *_test.odin과 #+ignore는 제외한다. 전체 공통 선언은 한 번, 일부 대상은 "Availability: Windows"로 표시한다. 다른 시그니처는 각각 대상과 함께 보여 주고 첫 항목만 ID를 갖는다. 일부 대상만 지원하는 패키지는 :platform:로 밝힌다.

다시 빌드하기

읽은 Odin 파일은 페이지의 의존성이다. 소스 수정 뒤에는 해당 패키지를 설명하는 페이지만 다시 읽는다. 패키지 추가·삭제·요약 변경 시 색인을 다시 읽는다. guidedog serve는 HTML 제공 전에 odin_autoapi_dirs와 odin_collections의 수정 시간을 확인한다. 편집 뒤 브라우저를 새로 고친다.

한계

- 소스는 파싱만 하며 컴파일·타입 검사는 하지 않는다. 타입과 상수 값은 그대로 표시하고 100자를 넘는 값은 생략하며 추론하지 않는다. A :: B의 타입·상수 여부는 이름으로 판단한다. 전체 대문자가 아닌 대문자 시작 이름이나 내장 타입이면 타입이다.
- when 조건은 ODIN_OS와 ODIN_ARCH 비교 외에는 평가하지 않고 양쪽 분기를 모두 문서화한다.
- 비공개 타입을 참조한 선언은 텍스트로 표시한다. :private-members:가 없으면 해당 항목이 없기 때문이다. 생성 패키지는 :private:로 나열하므로 -n에서도 보고하지 않는다.
- core:odin/parser는 #optional_ok와 where 절을 함께 쓰는 프로시저를 거부한다. 해당 파일을 보고하고 읽을 수 있었던 부분은 문서화한다.

- Odin 파서는 깊은 중첩에 많은 스택을 쓰며 수백 괄호로 스택이 넘칠 수 있다. 실제 코드보다 훨씬 깊은 파일(괄호·타입 약 50단계, `else` 사슬 약 천 개)은 `odin.autodoc.nesting` 경고로 제외하고 나머지 패키지는 문서화한다. `when`의 `&&`, `||`, `!`는 64단계까지 평가하며 그 이상은 미평가다.

실용적인 작업

각 페이지는 구체적인 작업으로 시작해 명령과 결과를 설명합니다.

Guidedog 빌드와 설치

macOS에서 Homebrew로 설치하기

Apple Silicon 또는 Intel Mac에서 공식 `tap insanai/homebrew-guidedog` 를 사용하세요.

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

Homebrew가 프로세서에 맞는 릴리스 패키지를 선택하고 SHA-256을 확인합니다. 패키지에는 Typst, Graphviz, tree-sitter가 포함되어 있습니다. HTML과 PDF를 만들 때 Odin 컴파일러, Python 환경, 별도의 Typst 설치가 필요하지 않습니다. macOS 12 이상이 필요합니다.

첫 프로젝트에 따라 첫 프로젝트를 만드세요. 설치된 버전을 업데이트하려면 다음을 실행하세요.

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

`brew test insanai/guidedog/guidedog` 를 실행하면 프로젝트를 만들고 HTML과 PDF를 빌드하여 설치된 패키지를 확인할 수 있습니다.

소스에서 빌드하기

`tools/odin-release.txt` 에 지정된 Odin 도구 모음을 사용하세요. 릴리스는 Odin `dev-2026-10` 과 Typst `0.15.1` 을 사용합니다. 공유 라이브러리 빌드에는 Graphviz 개발 라이브러리가 필요합니다.

개발용 빌드

저장소 루트에서 실행하세요.

```
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 빌드는 소스 기반 autodec용으로 버전이 고정된 Python 문법을 제공한다. Graphviz 개발 파일은 시스템 패키지 관리자로 설치한다. Typst 브리지가 없는 빌드는 설치된 typst로 PDF를 만든다. `--typst=/absolute/path/to/typst`로 직접 선택할 수 있다.

내장 컴파일러

네이티브 의존성을 한 번 빌드한 뒤 Odin 실행 파일에 링크하세요.

```
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 브리지는 Rust 코드이며 Guidedoc의 무할당 경계 밖에 있다. 내장 빌드는 별도의 Typst 실행 파일을 띄울 필요를 없애지만 컴파일러 자체의 메모리 비용은 그대로다.

Windows에서는 출력 이름을 `build/guidedog.exe`로 한다. 경로 없이 `guidedog`을 쓰려면 실행 파일을 PATH에 넣는다.

문서 빌드와 게시

프로젝트의 기본 출력 디렉터리로 빌드하세요.

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

사이트는 `docs/manual/_build/html`에 있다. PDF와 생성된 Typst 소스는 `docs/manual/_build/pdf`에 있다. 웹 서버가 HTML 폴더를 정적 파일로 제공할 수 있으므로 운영 환경에 Odin 서버가 필요하지 않다.

출력 디렉터리 선택

Sphinx 방식은 소스와 출력 경로를 명시합니다.

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

출력은 소스 밖에 둔다. 게시가 소스 트리를 대체하는 구조는 Guidedog이 거부한다. 사용자 지정 대상은 `build/.manual-site.guidedog/.doctrees` 같은 안정적인 형제 캐시를 사용하므로 게시 폴더에는 게시물만 들어간다.

미리 보기와 검사

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

dummy 빌더는 프로젝트를 읽고 관계를 해결하지만 게시물을 만들지 않는다. 빠른 의미 검사에 사용한다. 최종 페이지 레이아웃은 검사하지 않는다.

재현 가능한 시각

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


이렇게 빌드 시간을 고정한다. 재현성은 소스, 템플릿, 글꼴, 컴파일러 버전, 리소스에도 달려 있다. 시계만 고정해서는 부족하다.

빌드가 성공한 뒤 전체 폴더를 게시한다. 정상 종료는 필요조건이다. 템플릿을 바꾼 뒤에는 시각적 확인도 필요하다.

Sphinx 프로젝트 옮기기

Guidedog은 Sphinx의 프로젝트 구조와 여러 설정 이름을 따르지만 `conf.py`나 Python 확장은 실행하지 않는다. 이관은 데이터와 실행 가능한 설정을 분리하는 데서 시작한다.

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

`migrate`는 리터럴 설정을 읽어 `conf.toml`에 쓴다. 사람이 선택해야 하는 계산된 값은 보고한다. 값을 알아내려고 프로젝트를 가져오거나 Python을 실행하지 않는다. 동등한 설정으로 취급하기 전에 결과를 검토한다.

세 단계로 진행하기

1. 문서 그래프를 올바르게 구성하세요. 루트, 확장자, 제외 규칙, `include`를 확인합니다.
2. 참조를 올바르게 구성하세요. 도메인, 인벤토리, 사용자 정의 객체 타입을 확인합니다.
3. 표현을 올바르게 구성하세요. 템플릿, 다이어그램, 수식, 두 출력을 확인합니다.

알 수 없는 지시문은 프로젝트의 비공개 확장에 속할 수 있다. HTML 빌드 성공이 그 의미까지 재현되었음을 증명하지는 않는다. 진단을 읽고 지시문 대체, 지원되는 별칭 설정, 제한 사항을 기록한 유지 중 하나를 선택한다.

지원 범위와 CPython, Django, Flask 검증 자료는 [Sphinx 호환성](#)를 참고한다. 선언형 사용자 객체는 [객체 타입 선언](#)를 참고한다.

코드베이스 문서화

공개 이름에는 계약이 필요하다. 입력, 결과, 소유권, 실패 조건을 설명한다. 시그니처만으로는 설명서가 되지 않는다.

Odin

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

Guidedog은 Odin 파서로 소스를 읽는다. 문서화할 패키지를 컴파일하거나 실행하지 않는다. 생성 페이지는 문서 그래프와 객체 목록에 들어간다. 저장소의 별도 API 프로젝트는 `docs/api`이다. 지시문, 시그니처, 옵션은 [Odin 문서화](#)를 참고한다.

Python

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

Guidedog은 Python 소스를 가져오기 없이 파싱하여 import 부작용을 피한다. 따라서 실행 중 생성되는 객체는 빠질 수 있다. 공개 인터페이스에 속한다면 명시적으로 설명한다. 정확한 탐색 규칙은 `autodoc`으로 Python 문서 작성하기를 참고한다.

프로젝트 간 링크

생성한 `objects.inv`를 사이트와 함께 게시한다. 다른 프로젝트가 `intersphinx` 매핑으로 이름을 해결할 수 있다. 재현 가능한 빌드에는 명시적인 매핑을 쓴다. 다른 프로젝트로 연결을 참고한다.

이 문서 관리하기

Guidedog 프로젝트는 `docs/manual`, `docs/api`, `docs/gds` 세 개다. 설명서와 GDS는 `guidedog quickstart`로 시작했다. 생성된 템플릿을 직접 수정한다. 설계 기록을 설명서 목차에 넣지 않는다.

설명서 빌드

저장소 루트에서 실행하세요.

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

설명서 목차에는 전체 Odin API 참고서가 포함된다. PDF 명령은 설명과 선언을 링크로 연결한 한 권의 책을 만든다. 설명서와 별도 API 프로젝트는 모두 `lib/`에서 참고서를 생성하므로 복사한 API 원고를 동기화할 필요가 없다.

API 프로젝트만 빌드하려면 다음을 실행하세요.

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

논의 문서 빌드와 검사

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

GDS CLI는 메타데이터, 번호, 상태, 복구를 관리한다. RST 게시 경로는 프로젝트 엔진을 호출해 전체 HTML 사이트와 기록별 PDF를 만든다. 선택자는 PDF만 제한한다. HTML은 전체 목록을 갖춘 일관된 사이트로 남는다. 선택한 PDF 미리보기는 `_build/selected/<artifact>/pdf`를 쓰며 `_build/pdf`의 전체 목록은 유지한다.

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

일반 프로젝트 명령도 사용할 수 있다. GDS 프로젝트는 자체 `conf.toml` 과 `_build` 폴더를 쓴다. 작성 절차는 `docs/gds/workflow.rst` 를 읽는다.

게시하기 전에

링크와 진단, 넓은 표와 코드 블록을 확인한다. 타이포그래피를 바꾸면 PDF 페이지를 렌더링해 살핀다. HTML은 데스크톱·휴대폰 너비와 밝은·어두운 모드에서 테스트한다. 수식과 그림이 오프라인에서도 작동하는지 확인한다. 엔진을 수정하면 관련 라이브러리와 통합 테스트를 실행한다.

참조

이 페이지는 규약을 설명하며 검색과 조회를 위한 것입니다. 첫 예제가 필요하다면 따라 하기부터 시작하세요.

명령줄 참조

`guidedog` 명령줄 도구로 문서 프로젝트를 관리하고 빌드, 미리 보기, 번역, 변환할 수 있습니다.

기본 구문

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

종료 코드

표 6 종료 상태

종료 코드	조건
0	성공: 모든 작업이 완료되었으며 처리되지 않은 오류가 없습니다.
1	빌드 또는 변환 오류. <code>-w</code> 옵션으로 경고를 오류로 처리한 경우도 포함합니다.
2	명령줄 구문이 잘못되었거나 알 수 없는 플래그 또는 옵션을 사용했습니다.
3	설정된 자원 한도에 도달했거나 메모리를 확보할 수 없습니다.
4	파일 작업이나 외부 컴파일러가 실패했습니다.
5	내부 오류입니다. 진단 메시지에 보고 방법이 나옵니다.
70	충돌이 발생했습니다. 이전에 게시한 출력은 유지됩니다.

명령 개요

표 7 CLI 명령

명령	용도
<code>quickstart [DIR]</code>	문서 프로젝트를 만들고 바로 편집할 수 있는 템플릿과 설정을 생성합니다.
<code>build [BUILDER] [DIR]</code>	지정한 출력 형식으로 문서 프로젝트를 빌드합니다.
<code>serve [DIR]</code>	로컬 HTTP 미리 보기 서버를 시작하고 파일이 바뀌면 자동으로 다시 빌드합니다.
<code>clean [DIR]</code>	생성된 빌드 결과와 캐시를 삭제합니다.
<code>convert INPUT</code>	파일 하나 또는 표준 입력을 직접 변환합니다.
<code>migrate [conf.py]</code>	Sphinx의 <code>conf.py</code> 를 선언형 <code>conf.toml</code> 로 변환합니다.
<code>intl <update stat></code>	PO 번역 카탈로그를 갱신하거나 번역 진행 상황을 표시합니다.
<code>formats</code>	내장 리더와 렌더러, 형식별 적합성 정보를 표시합니다.
<code>gds <COMMAND></code>	Guidedog Discussions 설계 기록과 생명주기를 관리합니다.

guidedog quickstart

프로젝트 디렉터리에 `conf.toml`, `index.rst`, `_templates/` 의 템플릿, `_static/` 의 스타일을 생성합니다.

```
# 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
```

옵션:

- `-q`: 대화형 질문을 생략하고 명령줄에 지정한 값을 사용합니다.
- `-p`, `--project NAME`: 표시할 프로젝트 이름을 지정합니다.
- `-a`, `--author NAME`: 저자 또는 조직 이름을 지정합니다.
- `-v VERSION`: 짧은 버전 번호를 지정합니다. 예: 1.0.
- `-r RELEASE`: 전체 릴리스 식별자를 지정합니다. 예: 1.0.0-rc1.
- `-l LANGUAGE`: 프로젝트 기본 언어를 지정합니다. 기본값은 `en` 입니다.
- `--sep`: 루트에 소스를 두는 대신 `source/` 와 `build/` 디렉터리를 따로 만듭니다.
- `--suffix EXT`: 소스 문서의 기본 확장자를 지정합니다. 기본값은 `.rst` 입니다.

guidedog build

소스 읽기, 목차 트리와 상호 참조 해석, 최종 결과 게시를 처리합니다.

```
# 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
```

지원하는 빌더:

- **html**: 검색과 탐색 기능을 갖춘 HTML 사이트를 생성합니다.
- **dirhtml**: `dir/index.html` 과 같은 디렉터리 형태의 HTML URL을 생성합니다.
- **singlehtml**: 모든 문서를 하나의 HTML 페이지로 합칩니다.
- **pdf**: Typst로 PDF 책을 조판합니다. 별칭은 `latex`, `latexpdf` 입니다.
- **text**: 문서마다 일반 텍스트 파일을 생성합니다.
- **gettext**: 번역할 문자열을 GNU gettext POT 템플릿으로 추출합니다.
- **dummy**: 최종 파일을 쓰지 않고 문서 트리를 파싱하고 참조를 해석합니다. 구문 확인용입니다.

빌드 옵션:

- **-a**: 수정 시각과 관계없이 모든 출력 파일을 씁니다.
- **-E**: 캐시된 상태를 무시하고 환경을 새로 만듭니다.
- **-W**: 경고가 있으면 빌드 실패로 처리합니다.
- **--keep-going**: **-W** 를 사용해도 남은 문서를 계속 처리하여 종료 전에 모든 오류를 보고합니다.
- **-n**: 해결되지 않은 모든 상호 참조에 경고를 냅니다.
- **-j N** / **-j auto**: 문서 파싱에 사용할 스레드 수를 지정합니다. **auto** 는 CPU 코어 수에 맞춥니다.
- **-D name=value**: 이번 실행에서만 `conf.toml` 설정을 덮어씁니다. 예: **-D language=de**.
- **-D table.key=value**: `html_context` 같은 표 설정의 키 하나만 설정하고 나머지 키는 그대로 둡니다. `sphinx-build`와 같습니다.
- **-t TAG**: `only` 지시문에 사용할 조건 태그를 정의합니다.
- **-c DIR**: `conf.toml` 이 있는 디렉터리를 지정합니다.
- **-C**: `conf.toml` 파일을 읽지 않고 빌드합니다.
- **-v**: 상세 모드로 안내 메시지와 소요 시간을 표시합니다.
- **-q**: 경고와 오류만 표시합니다.
- **--color=always|never|auto**: 터미널의 ANSI 색상 출력을 제어합니다.
- **--diagnostics=json**: 프로그램이 읽을 수 있는 JSON 진단을 표준 오류로 출력합니다.
- **--budget=MIB**: 프로젝트 호스트가 보유할 메모리 예산의 상한을 지정합니다. 기본값은 1024 MiB입니다.
- **--memory=ram|disk**: 메모리 예산을 소진했을 때의 처리 방식을 지정합니다.
- **--untrusted**: 원시 임베디드 마크업을 생략하고 네트워크 가져오기를 비활성화합니다. 이는 기능 제한 정책이며 프로세스 샌드박스는 아닙니다.

guidedog serve

로컬 HTTP 서버와 파일 감시기를 시작합니다. 소스 문서, 템플릿, 자산이 바뀌면 다시 빌드하고 브라우저를 새로 고칩니다.

```
# 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
```

옵션:

- `-p, --port PORT`: 로컬 수신 포트입니다. 기본값은 8000 입니다.
- `--host HOST`: 바인딩할 IP 주소입니다. 기본값은 127.0.0.1 입니다.

guidedog clean

빌드 디렉터리와 환경 캐시를 삭제합니다.

```
guidedog clean docs
```

guidedog convert

프로젝트 디렉터리나 설정 없이 파일 또는 표준 입력을 직접 변환합니다.

```
# 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 -
```

옵션:

- `-o, --output FILE`: 출력 파일 경로입니다. `-` 는 표준 출력입니다.
- `--to FORMAT`: 출력 형식은 `html`, `pdf`, `text` 입니다. 파일 이름으로도 추론합니다.
- `--from FORMAT`: 입력 리더는 `rst`, `commonmark`, `myst`, `typst` 입니다. 입력 확장자로도 추론합니다.
- `--force`: 확인 없이 기존 출력 파일을 덮어씁니다.
- `--strict`: 경고가 하나라도 있으면 실패합니다.
- `--raw=allow|omit`: 원시 임베디드 코드 처리 정책입니다. 기본값은 `omit` 입니다.
- `--emit-typst FILE`: 생성한 중간 Typst 소스를 별도 파일에 저장합니다.

guidedog migrate

Sphinx의 Python 설정 파일 `conf.py` 를 선언형 `conf.toml` 로 변환합니다.

```
# 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
```

옵션:

- `-o FILE`: 지정한 경로에 저장합니다. `-` 는 표준 출력입니다.
- `--force`: 기존 `conf.toml` 을 덮어씁니다.

guidedog intl

다국어 문서의 번역 카탈로그를 관리합니다.


```
# 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 옵션:

- `-l`, `--language LANG`: 대상 언어 코드입니다. 여러 언어를 반복 지정할 수 있습니다.
- `-p`, `--pot-dir DIR`: 추출된 POT 메시지 템플릿의 경로를 지정합니다.

guidedog formats

내장 리더, 렌더러, 구문 강조 렉서와 명세 테스트의 적합성 결과를 표시합니다.

```
guidedog formats
```

guidedog gds

Guidedog Discussions 아키텍처 설계 제안(RFC)을 관리합니다.

```
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
```

설정 참조

Guidedog 프로젝트는 루트의 `conf.toml` 로 설정합니다. 설정은 선언형 TOML 데이터이며 임의의 코드를 실행하지 않습니다.

많은 설정 키가 Sphinx의 `conf.py` 변수에 대응하므로 `guidedog migrate conf.py` 로 이전을 도울 수 있습니다.

최소 설정

기본 메타데이터만으로 프로젝트를 시작할 수 있습니다. 생략한 설정에는 기본값을 사용합니다.

```
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"
```

프로젝트 메타데이터

표 8 메타데이터 옵션

키	형식	기본값	설명
<code>project</code>	문자열	<code>"Python"</code>	사람이 읽을 수 있는 프로젝트 이름입니다.
<code>author</code>	문자열	<code>""</code>	저자 또는 조직 이름입니다.
<code>copyright</code>	문자열	<code>""</code>	페이지 하단에 표시할 저작권 문구입니다.
<code>version</code>	문자열	<code>""</code>	짧은 버전 번호입니다. 예: <code>"1.0"</code> .
<code>release</code>	문자열	<code>""</code>	alpha나 beta 식별자를 포함한 전체 버전 문자열입니다. 예: <code>"1.0.0b1"</code> .
<code>language</code>	문자열	<code>"en"</code>	조판, 하이픈 처리, 번역에 사용할 ISO 언어 코드입니다. 예: <code>"de"</code> , <code>"zh_CN"</code> .
<code>today</code>	문자열	<code>""</code>	날짜를 직접 지정합니다. 생략하면 현재 날짜를 <code>today_fmt</code> 으로 표시합니다.
<code>today_fmt</code>	문자열	<code>""</code>	문서 날짜의 형식 문자열입니다.

일반 옵션

표 9 검색 및 파싱 옵션

키	형식	기본값	설명
root_doc	문자열	"index"	최상위 목차로 사용할 문서입니다.
source_suffix	배열	[".rst"]	문서 소스로 인식할 파일 확장자입니다.
exclude_patterns	배열	[]	소스 검색에서 제외할 디렉터리와 파일의 glob 패턴입니다.
include_roots	배열	[]	소스를 포함할 수 있도록 허용한 외부 루트 디렉터리입니다.
templates_path	배열	[]	HTML과 PDF 템플릿 디렉터리입니다. 내장 템플릿보다 먼저 검색합니다.
extensions	배열	[]	활성화할 내장 Sphinx 확장입니다. 예: "sphinx.ext.autodoc".
primary_domain	문자열	"py"	접두사가 없는 지시문과 역할의 기본 도메인입니다. 예: "py", "c", "odin".
highlight_language	문자열	"default"	코드 블록에 언어를 지정하지 않았을 때 사용할 기본 언어입니다.
pygments_style	문자열	""	밝은 테마에 사용할 Pygments 구문 강조 스타일입니다.
pygments_dark_style	문자열	""	어두운 테마에 사용할 구문 강조 스타일입니다.
smartquotes	불리언	true	곧은 따옴표와 대시를 조판용 문장 부호로 바꿉니다.
rst_prolog	문자열	""	각 문서 앞에 붙일 reStructuredText 조각입니다.
rst_epilog	문자열	""	각 문서 끝에 붙일 reStructuredText 조각입니다.

번호와 수식

표 10 번호 옵션

키	형식	기본값	설명
<code>numfig</code>	불리언	<code>false</code>	활성화하면 그림, 표, 코드에 자동으로 번호를 붙입니다.
<code>numfig_secnum_depth</code>	정수	<code>1</code>	그림 번호에 포함할 제목 깊이입니다. <code>1</code> 이면 <code>Fig. 2.1</code> 형식이 됩니다.
<code>numfig_format</code>	테이블	아래 참고	<code>"figure"</code> , <code>"table"</code> , <code>"code-block"</code> 의 번호 접두사 형식입니다.
<code>math_number_all</code>	불리언	<code>false</code>	모든 독립 수식에 자동으로 번호를 붙입니다.
<code>math_eqref_format</code>	문자열	<code>"({number})"</code>	<code>:eq:</code> 수식 참조의 형식 문자열입니다.
<code>mathjax_path</code>	문자열	URL	MathJax JavaScript 파일의 CDN URL 또는 로컬 경로입니다.

`numfig_format` 기본값:

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

진단과 엄격한 검사

표 11 진단 옵션

키	형식	기본값	설명
<code>nitpicky</code>	불리언	<code>false</code>	해결되지 않은 상호 참조와 잘못된 대상을 경고합니다.
<code>nitpick_ignore</code>	배열	<code>[]</code>	엄격한 참조 검사에서 제외할 <code>[type, target]</code> 쌍의 배열입니다.
<code>suppress_warnings</code>	배열	<code>[]</code>	보고하지 않을 경고 범주 코드입니다.
<code>keep_warnings</code>	불리언	<code>false</code>	게시한 문서에 경고를 포함합니다.

HTML 출력 옵션

표 12 HTML 옵션

키	형식	기본값	설명
html_theme	문자열	"guidedog"	HTML 생성에 사용할 테마입니다.
html_theme_options	테이블	{}	HTML 테마에 전달할 키-값 옵션입니다.
html_title	문자열	자동 생성	브라우저 탭에 표시할 제목입니다.
html_short_title	문자열	자동 생성	이동 경로에 사용할 짧은 제목입니다.
html_logo	문자열	""	소스 디렉터리를 기준으로 한 프로젝트 로고 경로입니다.
html_favicon	문자열	""	파비콘 파일 경로입니다.
html_static_path	배열	[]	출력의 _static/ 으로 복사할 디렉터리입니다.
html_extra_path	배열	[]	변환 없이 출력 루트로 복사할 디렉터리입니다.
html_css_files	배열	[]	HTML 페이지에서 불러올 사용자 CSS 파일 이름입니다.
html_js_files	배열	[]	HTML 페이지에서 불러올 사용자 JavaScript 파일 이름입니다.
html_permaLinks	불리언	true	단락과 절에 고정 링크를 붙입니다.
html_permaLinks_icon	문자열	"¶"	고정 링크에 사용할 기호나 텍스트입니다.
html_baseurl	문자열	""	사이트맵과 메타데이터에 사용할 정규 기본 URL입니다.
html_context	테이블	{}	Jinja 템플릿에 전달할 사용자 변수 사전입니다.
html_additional_pages	테이블	{}	추가로 렌더링할 페이지입니다: { "page_name" = . "template.html" }

PDF와 Typst 옵션

표 13 PDF 옵션

키	형식	기본값	설명
<code>pdf_paper_size</code>	문자열	"a4"	용지 크기는 "a4" 또는 "us-letter" 입니다.
<code>pdf_logo</code>	문자열	""	책 표지에 사용할 로고 경로입니다.
<code>pdf_toplevel_sectioning</code>	문자열	"chapter"	책의 최상위 구분에 대응할 절 수준입니다: "chapter" 또는 "part".
<code>pdf_show_urls</code>	문자열	"no"	인쇄본의 URL 표시 방식입니다: "no", "inline", "footnote".
<code>pdf_preamble</code>	문자열	""	생성한 문서 앞부분에 삽입할 원시 Typst 소스입니다.
<code>pdf_font_paths</code>	배열	[]	사용자 OTF/TTF 글꼴을 검색할 추가 디렉터리입니다.
<code>pdf_packages</code>	문자열	"download"	패키지 해석 정책입니다: "download" 또는 "offline".
<code>typst</code>	문자열	"typst"	시스템 Typst CLI의 이름 또는 절대 경로입니다.

PDF 책 정의

[[pdf_documents]] 테이블로 프로젝트에서 만들 책을 하나 이상 정의합니다.

```
[[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 옵션

표 14 MyST Markdown 옵션

키	형식	기본값	설명
<code>myst_enable_extensions</code>	배열	<code>["dollarmath"]</code>	구문 확장입니다: <code>"colon_fence"</code> , <code>"deflist"</code> , <code>"dollarmath"</code> , <code>"fieldlist"</code> , <code>"tasklist"</code> , <code>"substitution"</code> .
<code>myst_heading_anchors</code>	정수	<code>0</code>	제목 앵커를 자동 생성할 깊이입니다. <code>0</code> 이면 비활성화합니다.
<code>myst_substitutions</code>	테이블	<code>{}</code>	Markdown 문서에서 사용할 변수 치환입니다: <code>{key = "value"}</code> .
<code>myst_url_schemes</code>	배열	<code>["http", ...]</code>	외부 링크로 인식할 URI 스킴입니다.
<code>myst_commonmark_only</code>	불리언	<code>false</code>	MyST 확장 없이 CommonMark 구문만 사용합니다.

Intersphinx 옵션

```

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", ""]]

```

전체 절차는 다른 프로젝트로 연결을 참고하세요.

Python autodoc 옵션

표 15 Autodoc 옵션

키	형식	기본값	설명
<code>autodoc_source_paths</code>	배열	<code>[]</code>	정적 분석에서 Python 모듈을 검색할 디렉터리입니다.
<code>autoclass_content</code>	문자열	<code>"class"</code>	클래스 docstring을 가져올 위치입니다: <code>"class"</code> , <code>"init"</code> , <code>"both"</code> .
<code>autodoc_member_order</code>	문자열	<code>"alphabetical"</code>	멤버 정렬 방식입니다: <code>"alphabetical"</code> , <code>"bysource"</code> , <code>"groupwise"</code> .
<code>autodoc_typehints</code>	문자열	<code>"signature"</code>	타입 힌트 위치입니다: <code>"signature"</code> , <code>"description"</code> , <code>"none"</code> .
<code>autodoc_default_options</code>	테이블	<code>{}</code>	모든 <code>auto*</code> 지시문에 적용할 기본 옵션입니다.

전체 검색 규칙은 [autodoc](#)으로 Python 문서 작성하기 을 참고하세요.

Odin API 문서 옵션

표 16 Odin 도메인 옵션

키	형식	기본값	설명
<code>odin_autoapi_dirs</code>	배열	<code>[]</code>	Odin 패키지를 검색할 소스 디렉터리입니다.
<code>odin_autoapi_root</code>	문자열	<code>"api"</code>	생성할 Odin API 문서의 하위 디렉터리입니다.
<code>odin_autoapi_options</code>	배열	<code>["members", "undoc-members"]</code>	포함할 멤버를 거르는 필터입니다.
<code>odin_autoapi_member_order</code>	문자열	<code>"source"</code>	멤버 정렬 방식입니다: <code>"source"</code> 또는 <code>"alphabetical"</code> .

Odin 도메인의 전체 사용법은 [Odin 문서화](#) 을 참고하세요.

국제화 옵션

표 17 i18n 옵션

키	형식	기본값	설명
<code>locale_dirs</code>	배열	<code>["locales"]</code>	<code>gettext</code> 카탈로그를 검색할 디렉터리입니다.
<code>gettext_compact</code>	불리언	<code>true</code>	같은 디렉터리의 문서 메시지를 하나의 카탈로그로 합칩니다.
<code>gettext_uuid</code>	불리언	<code>false</code>	POT 메시지에 안정적인 UUID를 기록합니다.
<code>gettext_auto_build</code>	불리언	<code>true</code>	빌드할 때 <code>.po</code> 를 바이너리 <code>.mo</code> 카탈로그로 컴파일합니다.
<code>figure_language_filename</code>	문자열	<code>"{root}.\n{language}\n{ext}"</code>	언어별 이미지를 선택할 파일 이름 템플릿입니다.

번역 절차는 [국제화](#) 을 참고하세요.

리더, 렌더러, 빌더

리더는 소스 언어를 이해하고 렌더러는 문서 트리 하나를 출력한다. 빌더는 프로젝트를 조율하고 출력을 게시한다. 서로 다른 책임이다.

표 18 소스 처리 경로

입력	리더	경로
프로젝트의 RST	RST와 내장 Sphinx 계층	프로젝트 그래프를 HTML, Typst/PDF, 텍스트, 카탈로그로 만듭니다.
프로젝트의 Markdown	MyST 호환 Markdown 계층	같은 프로젝트 그래프와 빌더를 사용합니다.
convert file.md	기본값은 CommonMark	문서 하나를 HTML 또는 Typst/PDF로 만듭니다.
convert file.typ	네이티브 Typst 어댑터	Guidedoc 의미 트리 밖에서 Typst를 컴파일합니다.

빌더

html

문서별 HTML 파일과 색인, 검색, 인벤토리를 생성합니다.

dirhtml

디렉터리 형식의 문서 URL.

singlehtml

문서별 절과 로컬 참조를 담은 하나의 주 HTML 페이지입니다.

pdf

프로젝트가 정의한 Typst 책을 만듭니다. LaTeX 빌더 별칭도 이 경로를 선택합니다.

text

문서별 일반 텍스트 파일.

gettext

번역용 메시지 템플릿.

dummy

결과물을 게시하지 않고 프로젝트를 읽고 참조를 해석합니다.

지원 범위는 검증의 근거

`guidedog formats`는 실행 파일의 실제 어댑터를 보여 준다. 호환성 페이지에는 고정 테스트 모음과 실제 프로젝트 검증이 기록된다. CommonMark 통과가 전체 MyST 지원을, RST 읽기가 모든 Python 확장 동작을 증명하지는 않는다. Sphinx 호환성을 참고한다.

오류 메시지

진단은 문제의 이름과 위치를 밝히고 유용한 다음 행동을 알려야 합니다. Guidedog는 이 정보를 구조화된 데이터로 보관합니다. 터미널에는 색상과 소스 표시를 더하고, JSON은 터미널 장식 없이 데이터를 유지합니다.

진단 읽기

```

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".

```

진단은 항상 다음 순서의 네 부분으로 구성됩니다.

1. **제목 표시줄.** 제목은 문제 종류를 대문자로 표시합니다. 경고와 안내는 WARNING: 와 NOTE: 로 시작합니다. 오른쪽에는 `rst.substitution.undefined` 같은 진단 코드가 있으며, 검색에 쓸 수 있는 고정된 이름입니다.
2. **위치.** 파일과 줄 번호, 소스 줄, 문제 텍스트 아래의 표시를 보여 줍니다. 누락된 폴더나 옵션처럼 특정 줄과 관계없는 문제는 경로만 표시하거나 위치를 생략합니다.
3. **무슨 일이 일어났는가.** 문제를 설명하고, 필요하면 출력에 미치는 영향도 알립니다. 이 예에서는 아무것도 기록하지 않았다는 뜻입니다.
4. **어떻게 해결하는가.** 컬러 터미널에서 초록색으로 표시하는 구체적인 안내입니다. 해결 방법이 여러이면 가장 적합할 가능성이 높은 방법부터 나열합니다.

색상은 이미 문구로 설명한 내용을 강조할 뿐입니다. `--color=never`는 색상을 끄고, `--color=always`는 터미널이 아닌 출력에서도 색상을 유지합니다.

다음 행동을 안내하는 힌트

Guidedog의 메시지도 이 규칙을 따른다. 힌트는 작업("Add", "Pass", "Set", "Rename", "Run")과 대상을 먼저 밝힌다. 대상은 `--budget=MIB`, `conf.toml`의 `root_doc`, 지시문 옵션 등이다. `tests/hints_test.odin`은 모든 메시지를 읽고 빈 힌트, 메시지 반복, 무엇을 보고할지 없는 "report this"를 거부한다. 수정 자체를 제시하는 제안("Did you mean numfig?")과 작업이 필요 없는 메모의 "Nothing to do:" 및 그 이유도 허용한다.

Guidedog 자체의 오류이면 안내에서 이를 명확히 밝힙니다. 실행한 명령, 문제를 일으킨 입력, `guidedog --version` 출력을 함께 보고하도록 요청합니다.

프로그램용 진단

`--diagnostics=json`은 보고서마다 한 줄의 JSON 객체를 표준 오류에 써서 편집기와 지속적 통합에서 읽을 수 있게 한다.

```

{"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":""}

```

여기서는 줄바꿈했지만 실제 객체는 한 줄이다. `schema`는 필드가 바뀔 때만 변경되는 형식 버전이며 나머지를 읽기 전에 확인한다. 열은 1부터 Unicode 스칼라를 센다. 바이트나 터미널 칸 수가 아니다. `source`는 원본 줄 전체다. 터미널

은 오류 주변 일부를 보여 주며 제어 문자를 이스케이프하고 넓은 문자 폭을 측정해 표시를 맞춘다. `category`는 `suppress_warnings`의 경고 분류이거나 빈 값이다.

메모리가 부족할 때

메모리 부족도 진단으로 보고하며, 충돌하지 않습니다. 두 종류가 있습니다.

host.budget (메모리 예산 도달)

빌드가 보유 가능한 메모리 예산(`--budget=MiB`, 기본 1 GiB)에 도달했다. 보고서는 단계와, 템플릿이라면 메모리를 요청한 줄을 밝힌다. 터미널에서는 예산 확대, 작업 메모리의 디스크 전환, 중단 중 한 번 선택하게 한다. 그 밖의 환경에서는 중단한다. `--budget` 또는 `--memory=disk`로 미리 선택할 수 있다. 힌트도 같은 예산을 숫자로 제시한다.

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

제안 값은 현재 보유량에 거부된 단계 요구량의 네 배를 더한 값(출력과 쓰기용 사본도 보유함)과 기존 예산의 두 배 중 큰 값을 64 MiB 단위로 올림한다. 단계가 가용 메모리에 들어갈 때는 그 순간의 가용량을 넘지 않는다. 들어가지 않으면 이를 밝히고 `--memory=disk`를 먼저 제안한다.

```
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 (메모리 부족)

시스템이 예산 내의 메모리 요청을 거부했다. 명령은 거부된 공간을 쓰지 않고 멈춘다. 빌드는 아무것도 게시하지 않고, GDS 변경은 전부 적용하거나 전혀 적용하지 않으며, `convert`는 완전한 페이지를 쓰거나 아무것도 쓰지 않는다. 메모리를 확보하거나 스레드를 줄여(`-j 1`) 다시 실행한다. 게시 결정 뒤 출력 교체에는 메모리가 필요 없어 할당 거부로 게시가 중간에 멈추지 않는다. 이전 게시 복구를 계획할 메모리가 없으면 그대로 남겨 다음 빌드가 처리하게 한다.

둘 다 상태 3으로 끝납니다.

종료 상태

모든 명령은 다음 상태 중 하나로 끝나므로 스크립트에서 실패 종류를 구분할 수 있습니다.

상 태	의미
0	성공입니다. 출력을 기록했고, 빌드라면 게시했습니다.
1	입력에 문제가 있습니다. 문서 오류, 정책 거부(<code>--raw</code> 없는 raw 콘텐츠), 읽을 수 없는 문서, 또는 <code>-w</code> 나 <code>--strict</code> 로 오류 처리한 경고입니다.
2	명령 사용법이 잘못되었습니다. 알 수 없는 옵션이나 명령, 누락되거나 잘못된 값, <code>conf.toml</code> 또는 <code>-D</code> 의 잘못된 설정입니다.

상 태	의미
3	한도에 도달했습니다. 메모리 예산(--budget), 중첩 깊이(--max-depth), 노드 수(--max-nodes), 스택(--stack-kib), 너무 큰 소스, 시스템 메모리 부족이 해당됩니다.
4	파일이나 외부 프로그램에서 실패했습니다. 읽거나 쓸 수 없는 경로, 읽는 동안 변경된 파일, Typst 시작 또는 완료 실패가 해당됩니다.
5	Guidedog 내부에서 실패했으며, 진단에서 문제 보고를 요청합니다.
70	Guidedog이 충돌하면 GUIDEDOG CRASHED (internal error)와 실행 명령, 오류, guidedog --version 출력을 표시한다. 보고할 곳은 빌드가 지정한 이슈 추적기 또는 빌드 제공자다. 빌드는 마지막에만 게시하므로 이전 출력은 그대로 남는다.
130	Ctrl+C로 명령을 중단했으며, 게시된 파일은 그대로입니다.

어떤 이유로 빌드가 실패하든 이전 출력과 빌드 기록을 유지합니다. 문제를 수정한 뒤 다시 빌드하세요.

Sphinx 호환성

Guidedog는 프로젝트 이전에 도움이 되는 부분에서 Sphinx를 따르며, 기능별로 호환성을 평가합니다. Python 설정이나 임의의 Python 확장은 실행하지 않습니다. 빌드 성공은 빌드가 완료되었다는 근거일 뿐, 완전한 동등성을 증명하지는 않습니다.

프로젝트와 게시

표 20 프로젝트 작업

작업	구현된 동작
프로젝트 구조와 빌드 옵션	Sphinx 방식의 소스·출력 디렉터리와 옵션: <code>-a -E -W -n -D -t -j -b -M -c -C</code> .
설정	익숙한 Sphinx 설정 이름을 TOML 데이터로 작성합니다. 마이그레이션은 리터럴을 변환하고 계산이 필요한 값을 알립니다.
HTML 빌더	html, dirhtml, singlehtml을 지원합니다. 단일 페이지의 앵커에는 문서 이름이 포함됩니다.
다른 빌더	text, gettext, dummy를 지원합니다. pdf는 Typst를 사용하며, latex와 latexpdf도 같은 경로를 선택합니다.
템플릿	Guidedog의 Jinja 구현, 타입을 유지하는 html_context, 레이아웃 상속, 추가 페이지.
증분 빌드	소스, 의존성, 템플릿, 설정, 참조 조회가 바뀌면 관련 출력이 무효화됩니다.
게시	완전한 출력 세대를 게시합니다. 빌드가 실패하면 이전에 게시한 세대를 유지합니다.

`-D`의 불리언 값은 `true`와 `false` 또는 1과 0을 받습니다. `index` 없이 `contents`가 있으면 경고와 함께 `contents`를 루트로 사용할 수 있습니다. 추가 템플릿 페이지는 `dirhtml`에서도 사이트 루트에 놓입니다. 정확한 규약은 [템플릿](#)을 참고하세요.

변경되지 않은 페이지도 진단을 다시 보고하므로 `-w`는 새 빌드와 변경 없는 빌드에서 같은 의미를 갖습니다. 삭제된 문서의 페이지는 제거합니다. 게시는 복구 가능한 저널을 사용하며, 중단된 커밋은 다음 빌드에서 정리합니다. 불변식과 플랫폼 경계는 [완전한 출력 세대 게시](#)을 참고하세요.

`epub`, `man`, `texinfo`, `linkcheck`, `doctest`, `coverage`, `changes` 빌더는 제공하지 않습니다. 사용하면 수정 안내를 표시합니다. 저장소의 `tools/linkcheck`는 별도 검증 도구입니다.

소스 언어

표 21 고정 버전의 적합성 검증

리더	검증 결과와 한계
reStructuredText	코퍼스 소스 98개에서 비교 대상 Docutils 트리와 식별자가 일치합니다. 비교에서 소스 위치와 일부 내부 관리 속성은 제외합니다.
CommonMark 0.31.2	명세 예제 652개가 모두 통과합니다.
MyST 0.16.1 reference	리더 테스트 218개 중 215개, Sphinx 빌드 테스트 10개 중 8개가 일치합니다. 나머지 차이는 기록되어 있습니다.

고정 버전의 테스트이며, 이후 모든 버전과의 호환성을 뜻하지 않습니다. `guidedog formats`로 실행 파일의 리더와 렌더러 검증 정보를 확인하세요. 각 리더 디렉터리에도 적합성 설명이 있습니다.

`rst_prolog`는 시작 부분의 서지 필드 뒤에, `rst_epilog`는 소스 뒤에 삽입합니다. Markdown에는 둘 다 삽입하지 않습니다. `only`와 `ifconfig`는 조건이 참이면 내부 절을 유지합니다. 조건부 절이 주변 절의 수준을 바꾸면 배치가 Sphinx와 다를 수 있습니다. MyST의 `eval-rst` 안에 절 제목이 있으면 오류입니다.

참조와 도메인

Guidedog는 `toctree`, 레이블, 용어집, 절.그림 번호와 `ref`, `doc`, `numref`, `term`, `download`, `any`, `keyword`, `option`, `envvar`, `token` 참조를 구현합니다. 객체는 목차 트리, 사이드바, 페이지 개요에 나타날 수 있습니다.

표준, Python, C, C++, JavaScript, reStructuredText, 수학 도메인이 구현되어 있습니다. Odin 도메인은 Guidedog 전용 확장으로 패키지, 선언, 시그니처, 생성된 API 페이지를 설명합니다. 소스만 사용하는 탐색과 한계는 [Odin 문서화](#)을 참고하세요.

매개변수, 반환값, 예외, 변수 필드는 구조화된 설명을 이룹니다. 정규 이름은 참조와 인벤토리의 별칭이 됩니다. Python 기본값과 어노테이션은 소스 표기를 유지합니다. Sphinx는 Python의 `unparser`로 이를 다시 출력할 수 있습니다. C++ 선언은 Sphinx 호환 식별자 버전과 심볼 인벤토리를 게시합니다. 중첩 깊이 제한은 적용되며, 중첩 괄호는 선형 시간에 파싱합니다.

`numref`는 번호 자리 하나를 치환하며, 번호 없는 대상을 보고합니다. 절 번호는 읽기 순서를 따릅니다. 문서마다 한 번만 번호를 붙이고, 두 번째 번호 목차 트리는 충돌을 보고합니다. 서식 치환과 식별자의 세부 사항은 참조 테스트를 확인하세요.

일반 색인, 모듈 색인, 검색이 내장되어 있습니다. 검색은 단어 접두부를 일치시키며 Sphinx의 영어 어간 추출은 사용하지 않습니다.

선언형 객체 타입으로 자주 쓰이는 Python 확장 등록을 대체할 수 있습니다. `object_types`, `crossref_types`, `directive_aliases`가 데이터를 설명합니다. 사용자 정의 Python `parse_node`는 이름, 표시, 프로그램의 선언형 규칙으로 바꿉니다. 표현할 수 없는 코드는 마이그레이션에서 알립니다. 객체 타입 선언을 참고하세요.

내장 확장의 동작

표 22 확장의 범위

동작	상태
<code>todo</code> , <code>ifconfig</code> , <code>extlinks</code> , <code>autosectionlabel</code>	내장되어 있습니다. 하드코딩한 <code>extlinks</code> 링크에는 역할 사용을 제안할 수 있습니다.
<code>graphviz</code>	링크된 Graphviz가 정적 그림을 생성합니다. 자원 접근 경계가 적용됩니다.
<code>intersphinx</code>	로컬 인벤토리와 가져온 인벤토리를 지원합니다. Windows는 현재 로컬 파일만 지원합니다.
<code>githubpages</code>	정적 사이트 게시 파일을 기본 제공으로 생성합니다.
<code>mathjax</code> 와 <code>imgmath</code>	HTML 수식은 MathJax를 사용합니다. PDF는 지원하는 LaTeX 부분집합을 Typst로 변환합니다.
<code>myst_parser</code>	구현된 프로젝트용 Markdown 계층입니다.
<code>doctest</code> 지시문	내용을 표시합니다. 프로젝트 빌더는 해당 테스트를 실행하지 않습니다.
<code>autodoc</code>	패키지를 임포트하거나 실행하지 않고 Python 소스를 읽습니다.

소스 기반 `autodoc`은 런타임에 생성되는 모든 객체를 찾을 수 없습니다. 컴파일된 모듈, 외부 데코레이터, 계산된 값, 확장 이벤트 후에는 명시적인 한계가 있습니다. Python 3.13과 고정 버전의 Sphinx 문서 생성기 테스트를 기준으로 삼습니다. 전체 설정과 예외는 `autodoc`으로 Python 문서 작성하기를 참고하세요.

`autosummary`, `napoleon`, `viewcode`, 임의의 Python 확장은 이 경로에서 지원하지 않습니다. 설정에 지원하지 않는 확장이 있으면 알립니다. 알 수 없는 지시문이나 역할은 소스 위치와 함께 보고합니다. **생략된 내용은 게시 전에 반드시 검토해야 합니다.** 다른 테마 이름도 허용하지만 Guidedog의 테마 구현을 사용합니다.

신뢰와 자원 제한

일반 빌드는 프로젝트의 raw HTML·Typst, 템플릿, 네트워크 인벤토리를 허용합니다. 로컬 읽기는 소스 디렉터리와 명시적으로 공유한 루트로 제한합니다. 심볼릭 링크로 이 경계를 넓힐 수 없습니다. 템플릿, 정적 파일, include, 이미지, 글꼴, API 소스에도 같은 규칙이 적용됩니다.

Graphviz 이미지 자원은 문서를 기준으로 해석하며 경로 경계를 적용합니다. 지원되는 이미지 데이터는 그림에 포함합니다. `imagepath`와 `fontpath`는 경계를 넓히지 않습니다. Typst는 읽기 가능한 디렉터리를 담은 전용 루트에서 실행되어 책 템플릿과 프리앰블을 선언된 자원으로 제한합니다.

`--untrusted`는 읽기 범위를 소스 폴더로 좁힙니다. raw 콘텐츠, 안전하지 않은 URL, 외부 자원, 프로젝트 Typst 템플릿과 프리앰블을 진단과 함께 제외합니다. 인벤토리나 Typst 패키지를 가져오지 않습니다. 자원 읽기를 요청하는 그래프는 경고와 함께 코드로 표시합니다. 이 모드는 네이티브 컴파일러의 메모리나 실행 시간을 제한하지 않습니다.

호스트 예산의 기본값은 1 GiB입니다. 할당 전에 용량을 예약합니다. 페이지 작업 공간은 사용 후 해제하지만, 유지하는 프로젝트 그래프와 카탈로그는 프로젝트가 커질수록 메모리를 사용합니다. 읽기 워커가 작업 수락을 조율합니다. 요청을 거부하면 실패한 작업과 수정 옵션을 알립니다.

`--memory=ram`은 관리 예산 한도에서 멈춥니다. `--memory=disk`는 관리되는 작업 저장 공간에 메모리 매핑 임시 파일을 허용합니다. 유지되는 호스트 저장 공간은 RAM 예산에 계속 포함됩니다. `--disk-budget`는 매핑 저장 공간을 제한하며 디스크 여유도 남깁니다. 비대화형 명령은 응답을 기다리지 않습니다. 알려진 머신 여유 용량은 `GUIDEDOG_AVAILABLE_MIB`로 지정할 수 있습니다.

`--max-depth`, `--max-nodes`, `--stack-kib`는 문서 처리를 제한합니다. 한도를 바꾸면 읽기와 출력 캐시가 무효화됩니다. 깊이와 스택 용량은 맞아야 합니다. 네이티브 Graphviz, tree-sitter, Typst의 할당은 호스트 예산 밖에 있습니다. 엄격한 한도가 필요하면 운영체제 제한을 사용하세요. 오류 메시지와 메모리와 소유권을 참고하세요.

검증과 실제 프로젝트

`tools/sphinxdiff`는 고정된 Sphinx 테스트 루트 42개에서 페이지, 식별자, 본문 링크, 인벤토리, 번호, 진단을 비교합니다. 저장된 결과와 README는 의도적인 차이를 설명합니다. 정확한 소스 위치, 안정적인 읽기 순서의 번호, 대상 형식 전용 raw 콘텐츠를 생략할 때의 명시적 보고가 그 예입니다.

2026년 10월 1일 원격 검증에서 CPython, Django, Flask의 새 빌드와 변경 없는 HTML·PDF 빌드를 실행했습니다. 12번의 빌드가 모두 성공했고 진단과 링크 결과가 기준과 일치했습니다. Linux 테스트 1,039개와 대상 AddressSanitizer 테스트 94개도 통과했습니다.

이 프로젝트에는 지원하지 않는 Python 확장 구문이 포함되어 있으며, 그 진단도 검증 근거의 일부입니다. 종료 상태 0은 모든 비공개 지시문이 재현되었음을 뜻하지 않습니다. 게시에 진단이 없어야 한다면 `-w`를 사용하세요.

과거 측정값과 소스 리비전은 `docs/manual/evidence/manuals.md`에 있습니다. 최신 원격 검토 보고서는 `build/review-remote-20261001/report.txt`입니다. 이 측정값은 기록된 작업 부하를 설명하며, 보편적인 속도나 메모리 한계가 아닙니다.

내부 구조

이 장들은 구현의 근거와 현재 계약을 설명한다. GDS 기록은 별도 프로젝트에 결정과 검토 상태를 보존한다.

Guidedoc 엔진

Guidedoc은 변환 코어이며 호출자의 저장 공간을 빌린다. 소스를 읽고 의미 트리를 만든 뒤 패스를 실행하고 출력 크기를 측정하여 바이트를 내보낸다. 파일을 열거나 산출물을 게시하지 않는다.

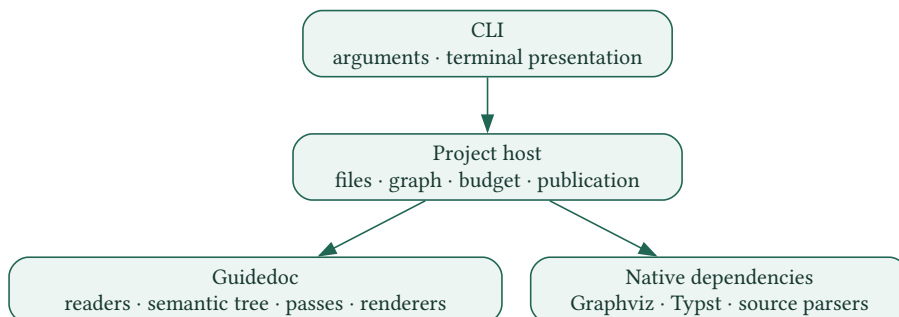


그림 4 외부 작업은 호스트가 맡고 문서의 의미는 Guidedoc이 맡는다.

경계

코어 변환은 할당하지 않는다. 호출자가 노드 열, 텍스트 저장 공간, 진단, 출력 용량을 제공한다. 호스트는 버퍼 준비 중 할당할 수 있다. 외부 컴파일러는 자체 규칙을 따른다. 보장의 범위는 변환 경계이며 프로세스 전체가 아니다.

트리

노드는 타입이 있는 ID를 쓴다. 자주 접근하는 열에는 종류와 부분 트리 범위를, 나머지 표에는 노드별 페이로드를 저장한다. 평탄한 전위 표현으로 힙 포인터 사슬을 따라가지 않고 부분 트리 전체를 건너뛸 수 있다.

노드 i 의 배타적 부분 트리 끝을 e_i 라 하자. 부분 트리는 $[i, e_i)$ 를 차지한다. 리프는 $e_i = i + 1$ 이며 부모는 자손 뒤에서 끝난다. 구간 증명은 알고리즘에 앞서는 불변식을 참고한다.

API

코어 API는 명시적이다. 저장 공간 초기화, 변환, 결과 확인, 초기화를 수행한다. 레지스트리는 리더.패스.렌더러 설명자를 담은 값이다. 실행 중 플러그인 탐색이나 전역 등록 부작용은 없다. 적은 수의 동작으로 다른 Odin 프로그램에서 쉽게 사용하게 한다.

소유권이 핵심 규칙이다. 작업 공간의 뷰는 초기화나 재사용 시 만료된다. 호스트가 보관하는 결과는 세션보다 오래 살아야 하는 텍스트를 소유해야 한다. 정확한 시그니처는 `core`에 있으며 별도 `docs/api`와 같은 Odin 소스에서 생성한다. 설명서 PDF에는 내부 링크로 같은 책에 포함된다. 설계 기록은 `docs/gds`의 GDS 0003이다.

알고리즘에 앞서는 불변식

유용한 불변식은 알고리즘 실행 중 유지되는 사실과 종료 후 호출자가 의존할 사실을 밝힌다. 반복문보다 먼저 작성한다.

전위 순회에서의 서브트리 구간

트리를 전위 순회로 출력한다고 가정한다. 노드는 자식보다 먼저 출력되고 각 부분 트리는 연속한다. e_i 를 노드 i 와 모든 자손 직후의 첫 인덱스라 하자.

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

명제. 인덱스를 e_i 로 옮기면 부분 트리를 건너뛸 수 있다.

증명. 연속성에 따라 모든 자손은 e_i 앞에 있다. 배타적 끝의 정의에 따라 이후 노드는 e_i 이상에 있다. 따라서 그곳으로 이동하면 정확히 부분 트리만 건너뛴다. 렌더러가 가정에 의존하기 전에 검증기가 확인한다.

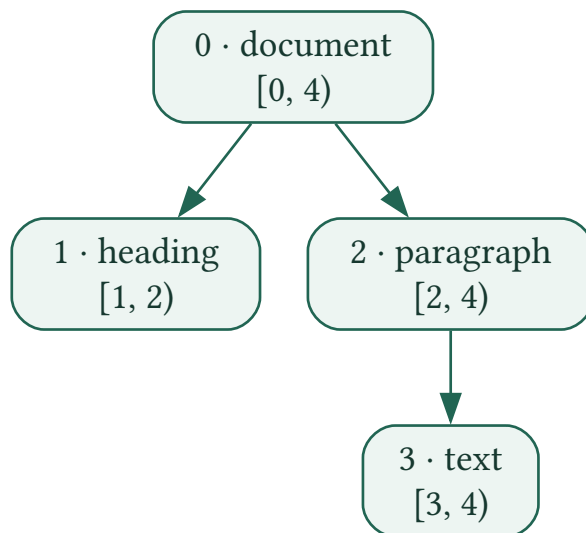


그림 5 부모의 구간은 자식의 구간을 포함합니다.

전체 스캔은 각 노드를 한 번 방문하므로 순회 비용은 $O(N)$ 이다. 부분 트리 건너뛰기는 인덱스 갱신 한 번이면 된다. 페이로드 작업의 비용은 여전히 발생한다. 상수 시간 건너뛰기와 상수 시간 렌더링을 혼동하지 않는다.

출력 전에 측정하기

렌더러는 먼저 바이트 수 m 을 측정한다. 호출자의 출력 용량을 c 라 하자.

$$m \leq c \implies \text{출력을 시작할 수 있다.}$$

측정과 출력 순회는 같은 규칙을 쓴다. 성공하면 정확히 m 바이트를 써야 한다. 실패 시 유효한 부분 산출물을 반환하지 않는다. 게시를 호스트가 맡으므로 불완전한 바이트가 이전 출력을 대체할 수 없다.

캐시 유효성

캐시 키는 결과를 바꿀 수 있는 모든 데이터를 포함해야 한다. 소스만으로는 부족하다. 포함 파일, 템플릿, 설정, 참조 조회, 리소스, 어댑터 버전도 출력에 영향을 준다. 기록된 의존성이 일치할 때만 재사용한다. 키가 같아도 출력이 없으면 재사용할 수 없다.

테스트는 코어 경계에 실패하는 할당자를 설치하고 출력 내용을 비교한다. import도 감사한다. 결과 확인 없이 할당 수만 세면 거부 때문에 문서 일부가 조용히 사라지는 문제를 놓칠 수 있다.

완전한 출력 세대 게시

독자는 절반만 갱신된 사이트를 보면 안 된다. 이는 게시 속성이며 파싱과 렌더링과는 별개다.

불변식

게시된 출력 트리를 P 라 하자. 준비 중 P 는 변하지 않는다. 다음 트리 Q 는 커밋 결정 전에 완성된다. 커밋 저널은 출력과 관련 기록을 지정한다.

준비는 실패할 수 있다. 호스트는 미완성 세대를 버린다. **확정된 커밋은 중단될 수 있다.** 다음 빌드가 저널에 따라 완료하거나 복원한다. 임시 폴더 이름으로 추측하지 않는다.

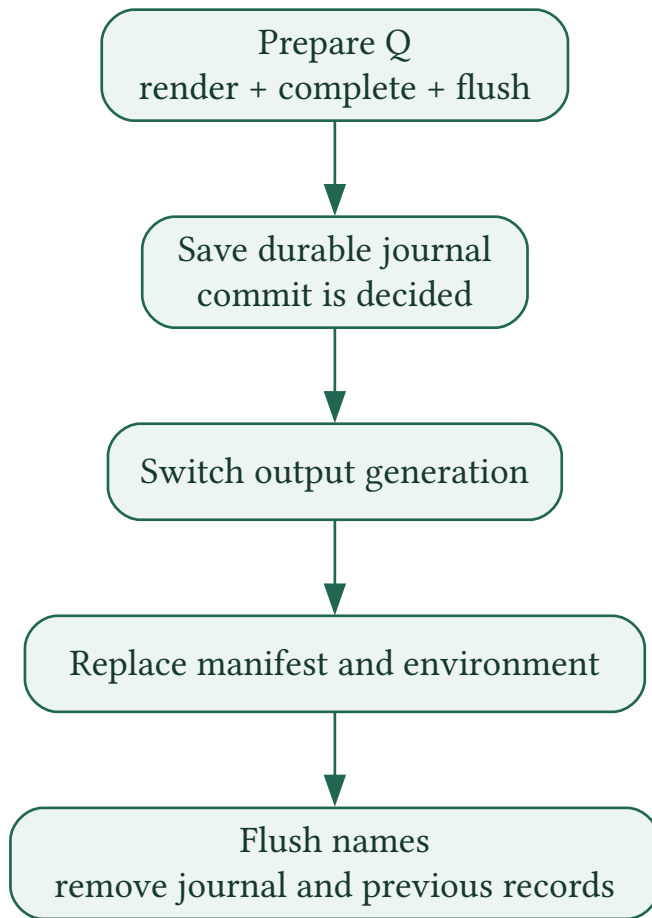


그림 6 영속 저널은 준비 단계와 확정된 커밋을 구분한다.

캐시가 출력 밖에 있는 이유

폴더 교환은 게시 폴더 안의 모든 것을 옮긴다. 내부 저널도 자신이 설명하는 작업 중 이동하여 복구 시 위치가 불안정해진다. Guidedog은 게시 트리 밖의 안정적인 폴더에 기록을 둔다. 사용자 지정 출력에 형제 캐시를 쓰는 이유다.

지속성과 가시성

파일 내용 동기화와 디렉터리 이름 동기화는 다르다. 알고리즘은 저널을 지우기 전에 둘 다 처리한다. 플랫폼마다 폴더 교환 방식이 다르며 복구 프로토콜은 이를 고려한다. 구현은 `lib/project/commit.odin`과 `lib/project/publish.odin`에 있다.

이는 프로젝트 게시를 보호한다. 외부 호스팅 서비스와의 분산 트랜잭션은 아니다. 성공한 빌드를 서버로 복사하는 작업은 배포 도구가 맡는다.

메모리와 소유권

예산은 회계 계약이다. 계상된 비용의 수명이 그 비용이 담당하는 저장 공간의 수명과 같아야 의미가 있다.

사용량은 저장 공간을 따라간다

할당 전에 바이트를 예약하고 해제 뒤에 비용을 반환한다. 버퍼 교체 중에는 새 버퍼와 기존 버퍼가 공존할 수 있으므로 새 할당 전체를 계상한다. 증가분만 계산하면 최대 사용량을 과소평가한다.

교체 시 최대 사용량 = 그 밖의 사용 중 저장 공간 + 기존 버퍼 + 새 버퍼 .

세션 소유 입력은 세션이 회수한다. 호출자 소유 입력은 계속 호출자가 책임진다. `read_owned`와 `read_kept`가 이를 명시한다. 더 짧게 읽어도 이미 할당한 공간은 줄지 않는다.

알려진 0은 미상과 다르다

메모리 탐색은 여유가 없음을 확인하거나 여유를 알아내지 못할 수 있다. 둘은 다른 상태다. 정책은 모두 0으로 나타내지 않고 별도의 확인 여부 플래그를 둔다. Linux에서는 보이는 상위 계층의 읽을 수 있는 `cgroup` 제한도 적용한다. 호스트 전체의 가용량만으로 컨테이너 메모리를 판단하면 잘못될 수 있다.

진단은 작업 공간보다 오래 남는다

세션 뒤에 표시하는 보고서는 제목, 소스, 수정 문구를 소유해야 하며 해제된 버퍼를 빌려서는 안 된다. 내부 소스 범위는 바이트 오프셋, 사용자 열은 Unicode 스칼라 수를 쓴다. 터미널 표시는 화면 너비를 별도로 고려한다.

남아 있는 경계

Guidedog은 관리하는 작업 공간을 측정한다. Typst와 네이티브 의존성은 자체 할당을 관리하므로 큰 PDF 책은 호스트 예산보다 훨씬 많은 메모리를 쓸 수 있다. 원격 CPython 검증의 Typst 포함 최대 RSS는 약 14.5 GiB였다. 이는 해당 작업의 측정값이며 보편적 상한은 아니다. 예산에 도달했을 때의 선택은 오류 메시지에 있다.

이 설명서에 관하여

이 설명서는 구현된 도구와 의도적으로 정한 한계를 설명합니다. 제안, 결정, 검토 이력은 별도의 GDS 프로젝트에 기록합니다. 조판을 개선한다고 제안의 상태가 바뀌지는 않습니다.

실제 사용 흐름에 맞춰 구성했습니다. 튜토리얼로 배우고, 주제별 안내로 이해하고, 작업 안내로 실행하며, 참고 문서에서 세부 사항을 확인합니다. 내부 구조에서는 불변 조건과 증명을 설명합니다.

작은 예제를 들고, 가정을 명시하며, 요점을 직접 설명합니다. 독자가 직접 확인할 수 있는 설명을 지향합니다. 절제된 색상, 뚜렷한 제목, 읽기 편한 세리프 글꼴, 접근성을 고려한 탐색 기능을 사용합니다. 특정 저자의 문장이나 지면을 모방하지는 않습니다.

이 프로젝트는 `guidedog quickstart`로 만들었으며 Guidedog로 빌드합니다. 본문은 RST, 도표는 Graphviz, 수식은 LaTeX 수학 표기법으로 작성합니다. Typst는 PDF 조판 템플릿에만 사용합니다.

빌드 검증 기록

2026-10-01 원격 검증에서 CPython, Django, Flask의 HTML과 PDF를 각각 새로 빌드하고 증분 빌드했습니다. 명령 12개가 모두 성공했으며 진단과 링크 검사 결과는 기존 기준과 일치했습니다. Linux 테스트 1,039개와 대상별 sanitizer 테스트 94개가 통과했습니다. 이는 검사한 버전의 동작을 뒷받침할 뿐, Sphinx와 완전히 호환된다는 뜻은 아닙니다. 남은 한계는 [Sphinx 호환성](#)를 참조하세요.

문서 유지 관리 명령은 [이 문서 관리하기](#)에 있습니다.

Guidedog

English · 简体中文 · 日本語 · 한국어 · हिन्दी · Español · Deutsch

한 번 쓰고, 명확히 설명하고, 웹과 종이로 전하세요.

Guidedog은 reStructuredText와 Markdown으로 HTML 문서 사이트와 PDF 책을 만듭니다. Odin으로 구현되었으며 변환 엔진의 이름은 **Guidedoc**입니다. PDF는 Typst로 조판합니다. Python 환경이나 LaTeX 설치가 필요하지 않습니다.

작성 흐름은 Sphinx에서 배웠지만 Guidedog은 독립적인 시스템입니다. 설정은 선언형 TOML이며 API는 정적으로 추출합니다. Python 설정이나 확장은 실행하지 않습니다. Sphinx와의 완전한 호환은 목표가 아닙니다.

프로젝트 시작

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

프로젝트에는 `conf.toml`, `index.rst`, `_templates/layout.html`, `_templates/book.typ`, `_static/` 이 있습니다. 템플릿은 직접 편집할 수 있는 일반 파일입니다. HTML은 탐색, 검색, 코드 강조, 도표, 수식을 제공합니다. PDF는 장, 상호 참조, 그림, 색인, 언어별 조판을 제공합니다.

기본 테마는 밝게이며 어둡게 또는 시스템 설정을 선택할 수 있습니다. 매뉴얼 첫 페이지와 저장소 개요는 브라우저 언어에 맞는 판을 선택합니다. 저장한 선택이 우선하며 특정 언어판이나 장으로 직접 들어가는 링크는 그대로 따릅니다.

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

설치

각 릴리스에는 Linux(x86-64, ARM64), macOS(Apple 실리콘, Intel), Windows(x86-64)용 압축 파일이 있습니다. 압축 파일마다 Typst, Graphviz, tree-sitter가 들어 있어 사이트와 책을 빌드하는 데 다른 것을 설치할 필요가 없습니다.

니다. Linux에는 glibc 2.35 이상과 libcurl이, macOS에는 버전 12 이상이 필요합니다. 사용하는 시스템에 맞는 압축 파일과 SHA256SUMS 를 내려받아 확인한 뒤 설치하세요.

```
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/
```

macOS에서는 공식 [Guidedog Homebrew tap](#)에서 설치하세요. Homebrew가 Apple Silicon 또는 Intel용 패키지를 자동으로 선택합니다. 이 tap은 Linux도 지원합니다.

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

Windows에서는 압축 파일을 풀고 그 폴더를 PATH 에 추가하세요. Graphviz DLL과 config8 은 guidedog.exe 옆에 그대로 두어야 합니다. 프로그램에는 코드 서명이 없습니다. macOS에서 브라우저로 내려받았다면 `xattr -d com.apple.quarantine guidedog` 으로 격리 속성을 제거하세요. 플랫폼 별 제한 사항은 릴리스 노트에 있습니다.

소스에서 빌드

Odin dev-2026-10(tools/install-toolchain.sh odin DIR 로 내려받고 확인할 수 있습니다), C 컴파일러, 그리고 Linux에서는 libcurl, expat, zlib 개발 패키지를 설치하세요. 네이티브 라이브러리를 먼저 빌드한 다음 Guidedog을 빌드합니다.

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

이렇게 만든 build/guidedog 은 PDF에 설치된 Typst 0.15.1을 사용합니다. 릴리스처럼 Typst 컴파일러를 내장하려면 Rust와 Cargo를 설치하고 다음을 실행하세요.

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

Windows에서는 Visual Studio의 C++ 도구와 LLVM을 갖춘 Git Bash를 사용하세요. Graphviz는 Windows용 정적 빌드가 없으므로 build.sh 대신 native/graphviz/windows.sh 를 실행합니다. 이 스크립트는 공식 Graphviz 16.1.0 DLL을 설치하고, tools/release/build.sh 가 이를 build/guidedog.exe 옆에 복사합니다. 테스트 스위트는 tools/release/suites.sh 로 실행하며, 릴리스 절차는 [RELEASING.md](#)에 설명되어 있습니다.

Guidedog으로 번역

```
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 파일의 `msgstr` 을 번역하고 검토하세요. Guidedog은 POT 추출, 카탈로그 병합, 진행 상황 표시, 각 언어판 빌드를 처리하지만 번역문을 만들어 주지는 않습니다. 명령, 경로, 코드, API 이름, 참조 대상은 바꾸지 않습니다. 매뉴얼과 저장소 개요만 번역하며 API와 GDS는 영어로 유지합니다. 전체 사이트와 PDF를 조합하는 방법은 빌드 안내를 참고하세요.

GitHub에서는 위 링크로 README 언어를 선택합니다. 자동 언어 선택은 생성된 문서 웹사이트에서 동작합니다.

문서 읽기와 임베딩

- **매뉴얼:** 튜토리얼, 주제 설명, 작업 안내, 참조, 내부 구조.
- **API 참조:** Odin 소스에서 생성하며 매뉴얼 PDF의 부록에도 포함합니다.
- **라이브러리 안내:** 호출자가 소유하는 작업 공간, 메모리, 변환 예제.
- **템플릿:** Jinja로 HTML을, Typst로 책을 사용자화합니다.
- **GDS:** 번호가 지정된 설계 기록과 구현 결정.
- **베타 검토:** 수정, 검증, 지원 범위, 남은 출시 조건.

Guidedoc은 호출자가 제공한 저장 공간을 사용하고 구조화된 진단을 반환합니다. 호스트는 파일, 예산, 캐시, 게시, 네이티브 컴파일러를 관리합니다. `--untrusted` 는 기능을 제한할 뿐 프로세스 샌드박스나 모든 네이티브 자원의 상한은 아닙니다. 베타는 macOS와 Linux의 신뢰할 수 있는 문서 프로젝트를 대상으로 합니다.

기여자와 라이선스

Vikrant Rathore가 만들고 **Ronak Rathore**와 **Kanak Rathore**가 개발을 도왔습니다. 이슈와 풀 리퀘스트를 통한 기여를 환영합니다. 문서에는 이름이나 계정 이름만 표기하고 개인 이메일 주소는 공개하지 않습니다. Git 작성자 설정은 별도로 관리합니다.

Sphinx와 Georg Brandl, Docutils와 David Goodger, Typst, Tree-sitter와 Max Brunsfeld, Graphviz, Ginger Bill과 Odin 커뮤니티에 감사드립니다.

Guidedog은 GNU Affero General Public License v3.0 이상(AGPL-3.0-or-later)으로 배포합니다. 다른 라이선스 조건이 필요하면 저자에게 문의하세요. 연락 방법과, 포함된 글꼴처럼 각자의 라이선스를 따르는 서드파티 구성 요소 목록은 NOTICE에 있습니다.

추가 허가에 따라 Guidedog을 Graphviz와 링크하여 배포할 수 있습니다. 자세한 내용은 NOTICE를 참고하세요.

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 “Guidedog: 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
```

```
Inline
```

```
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 $\backslash(\backslash)$ or $\backslash[\backslash]$, 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' stack 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 platforms' 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"
```

사용 가능 여부: 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 :: enum 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 `<IMG>`): 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++"}
```

사용 가능 여부: 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"}
```

사용 가능 여부: 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"}
```

사용 가능 여부: Windows.

```
@(extra_linker_flags = #config(GRAPHDOG_LINK_FLAGS, ""))
foreign import graphdog.graphviz {"system:gvc",
"system:cgraph"}
```

사용 가능 여부: 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’ `ttype` 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 Guidedog’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 ! = 0`) 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 `0`. 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; `0`: `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
```

0: 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: [ju8) -> 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)
```

사용 가능 여부: Windows, Darwin, Linux, FreeBSD, OpenBSD, NetBSD.

exchange: see switch.odin. macOS swaps with rename_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)
```

사용 가능 여부: 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
```

사용 가능 여부: 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
```

사용 가능 여부: 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
```

사용 가능 여부: 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
```

사용 가능 여부: 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"
```

사용 가능 여부: Windows.

```
foreign import host.system "system:System"
```

사용 가능 여부: 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

```
il18n.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.

```
il18n.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.

```
il18n.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 :: enum 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 :: struct`

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 (bools 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

```
odindoc.Arch :: runtime.Odin_Arch_Type
```

```
odindoc.Archs :: runtime.Odin_Arch_Types
```

```
odindoc.Config :: struct
```

```
default_options: []Option
```

```
member_order: string
```

“source” (the default), “alphabetical”, “groupwise”.

```
odindoc.Decl :: struct
```

`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`”).

```
name: string
```

```
kind: Kind
```

```
signature: string
```

```
attributes: []string
```

as written, in order: “`@(require_results)`”.

```
docs: string
```

```
private: Private
```

```
members: []Member
```

```
foreign_lib: string
```

the library of a foreign block it is declared in.

```
targets: Targets
```

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_permlinks_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:** boola 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.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 0-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"}
```

사용 가능 여부: Windows.

```
foreign import typst.bridge {"../..../native/typst_bridge/
target/release/
libguidedog_typst_bridge.a", "system:Security.framework",
"system:CoreFoundation.framework",
"system:System", "system:iconv"}
```

사용 가능 여부: 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"}
```

사용 가능 여부: Linux, FreeBSD, OpenBSD, NetBSD, WASI, JS, Orca, Freestanding.

색인

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), 195
- Arena.bytes (field in host), 195
- Arena.last (field in host), 195
- 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), 195
- Artifact_Request.allocator (field in host), 196
- Artifact_Request.budget (field in host), 196
- 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), 195
- 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), 196
- Artifact_Request.to (field in host), 196
- Artifact_Request.typst_options (field in host), 196
- Artifact_Request.typst_program (field in host), 196
- Artifact_Result (struct in host), 196
- 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), 167
- 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), 167
- Attribute_Kind.Edge (enumerator in graphdog), 168
- Attribute_Kind.Graph (enumerator in graphdog), 167
- Attribute_Kind.Node (enumerator in graphdog), 167
- 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), 208
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), 197
Budget.freed (field in host), 198
Budget.limit (field in host), 197
Budget.mutex (field in host), 198
Budget.peak (field in host), 197
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), 197

Budget.waiting (field in host), 197
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), 198
Budget_Policy.locate (field in host), 198
Budget_Policy.locate_user (field in host),
198
Budget_Policy.on_refusal (field in host),
198
Budget_Policy.spill (field in host), 198
Budget_Policy.user (field in host), 198
budget_problem (procedure in host), 209
budget_report (procedure in host), 209
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), 151
 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), 160
 catalog_refused (procedure in gettext), 160
 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 pyscan), 281
 Class.bases (field in pyscan), 282
 Class.decorators (field in pyscan), 282
 Class.doc (field in pyscan), 282
 Class.keywords (field in pyscan), 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), 171
 - (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), 210
- 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), 171
- 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), 211
 - Disk (struct in host), 199
 - Disk.chunks (field in host), 200
 - Disk.dir (field in host), 199
 - Disk.failure (field in host), 200
 - Disk.limit (field in host), 199
 - Disk.mutex (field in host), 200
 - Disk.peak (field in host), 199
 - Disk.refused (field in host), 200
 - Disk.used (field in host), 199
 - 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), 200
 - 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), 161
 - domain_plural_with_context (procedure in gettext), 161
 - 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), 168
- Engine.Patchwork (enumerator in graphdog), 168
- Engine.Sfdp (enumerator in graphdog), 168
- Engine.Twopi (enumerator in graphdog), 168
- engine_name (procedure in graphdog), 171
- 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), 152
 - (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), 169
- 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), 169
- Error_Kind.Missing_Format (enumerator in graphdog), 169
- 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), 200
 Files.canonical (field in host), 201
 Files.ledger (field in host), 201
 Files.max_bytes (field in host), 200
 Files.project (field in host), 201
 Files.roots (field in host), 200
 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), 190
 find_lexer_for_filename (procedure in
 highlight), 190
 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), 190
 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), 201
 Format.Commonmark (enumerator in
 host), 201
 Format.Dot (enumerator in graphdog), 170
 Format.Png (enumerator in graphdog), 170
 Format.Rst (enumerator in host), 201
 Format.Svg (enumerator in graphdog), 170
 Format.Typst (enumerator in host), 201
 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), 162
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),
166
FUZZY_THRESHOLD (constant in
gettext), 166
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), 201
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), 153
Header_Field.name (field in gettext), 154
Header_Field.value (field in gettext), 154
highlight (package), 174
highlight/guidedoc (package), 194
highlight_html (procedure in highlight),
191

highlight_html_lines (procedure in
highlight), 191
highlight_typst (procedure in highlight),
191
highlighter (procedure in highlight/
guidedoc), 194
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), 311
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), 191
 kind_name (procedure in core), 132
 (procedure in highlight), 191
 kind_parent (procedure in highlight), 191

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), 202
- Ledger.budget (field in host), 202
- Ledger.held (field in host), 202
- Ledger.refused (field in host), 202
- length (procedure in jinja), 246
- Lexer (enum in highlight), 175
- Lexer.Bash (enumerator in highlight), 176
- Lexer.C (enumerator in highlight), 175
- Lexer.Console (enumerator in highlight), 176
- Lexer.Cpp (enumerator in highlight), 175
- Lexer.Csharp (enumerator in highlight), 176
- Lexer.Css (enumerator in highlight), 176
- 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), 176
- 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), 175
- Lexer.Pytb (enumerator in highlight), 175
- 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), 191
- lexer_info (procedure in highlight), 192
- Lexer_Info (struct in highlight), 177
- Lexer_Info.aliases (field in highlight), 177
- Lexer_Info.filenames (field in highlight), 177
- Lexer_Info.mimetypes (field in highlight), 177
- Lexer_Info.name (field in highlight), 177
- Lexer_Info.spec (field in highlight), 178
- Lexer_Info.step (field in highlight), 177
- 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), 163
 Merge_Options (struct in gettext), 154
 Merge_Options.fuzzy_matching (field in gettext), 154
 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), 155
 Message.id_plural (field in gettext), 155
 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
 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), 172
 parse_expression (procedure in pyscan),
 294
 parse_format (procedure in graphdog), 172
 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), 164
- Plural_Node (struct in gettext), 156
- Plural_Node.a (field in gettext), 156
- Plural_Node.b (field in gettext), 156
- Plural_Node.c (field in gettext), 156
- 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), 157
- 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 pycan), 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), 178
 - (struct in host), 203
- Problem.code (field in gds), 147
 - (field in host), 203
- 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), 203
- Problem.kind (field in highlight), 178

Problem.lexer (field in highlight), 178
 Problem.line (field in highlight), 179
 Problem.message (field in gds), 147
 (field in host), 203
 Problem.name (field in highlight), 178
 Problem.near (field in highlight), 179
 Problem.path (field in gds), 147
 (field in host), 203
 Problem.suggestion (field in highlight), 178
 Problem.title (field in gds), 147
 (field in host), 203
 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
 (enum in highlight), 179
 Problem_Kind.Unknown_Style
 (enum 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), 192
 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), 158
- Reference_Style.None (enumerator in
 - gettext), 159
- Reference_Style.One_Per_Line
 - (enumerator in gettext), 159
- Reference_Style.Wrapped (enumerator in
 - gettext), 158
- 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),
 - 204
- Refusal_Choice.Raise (enumerator in host),
 - 204
- Refusal_Choice.Stop (enumerator in host),
 - 204
- Refusal_Decision (struct in host), 204
- Refusal_Decision.choice (field in host), 204
- Refusal_Decision.limit (field in host), 204
- 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), 172
 - (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), 205
- 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), 205
- 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), 205
- 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), 192
 - (procedure in readers/rst/inline), 330
- resolve_link (procedure in core), 136
- resolve_style (procedure in highlight), 192
- 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.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.name (field in readers/commonmark), 300
- Role_Handler.run (field in readers/commonmark), 300
- 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
- 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), 179
Scanner.code (field in highlight), 179
Scanner.count (field in highlight), 180
Scanner.head (field in highlight), 180
Scanner.inner (field in highlight), 180
Scanner.limit (field in highlight), 179
Scanner.nested (field in highlight), 180
Scanner.pos (field in highlight), 179
Scanner.queue (field in highlight), 180
Scanner.region (field in highlight), 180
Scanner.region_start (field in highlight),
180
Scanner.state (field in highlight), 179
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), 206
Session.map_size (field in host), 206
Session.merged (field in autodoc), 86
Session.output (field in host), 206
Session.own (field in host), 206
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), 206
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.sectsubtile (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), 207
- Sizes.links (field in host), 207
- 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), 207
- Sizes.second_bank (field in host), 208
- Sizes.sources (field in host), 208
- Sizes.text (field in host), 207
- 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.autoescape (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), 180
- Style.Autumn (enumerator in highlight), 180
- Style.Borland (enumerator in highlight), 180
- Style.Bw (enumerator in highlight), 180
- Style.Coffee (enumerator in highlight), 180
- Style.Colorful (enumerator in highlight), 180
- 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), 181
- Style.Manni (enumerator in highlight), 181
- Style.Material (enumerator in highlight), 181
- Style.Monokai (enumerator in highlight), 181
- Style.Murphy (enumerator in highlight), 181
- Style.Native (enumerator in highlight), 181
- 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), 182
 - Style.Staroffice (enumerator in highlight), 182
 - Style.Stata_Dark (enumerator in highlight), 182
 - Style.Stata_Light (enumerator in highlight), 182
 - Style.Tango (enumerator in highlight), 182
 - Style.Trac (enumerator in highlight), 182
 - 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), 183
 - 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), 183
 - Style_Info.special_background (field in highlight), 183
 - Style_Info.special_line_number (field in highlight), 183
 - style_name (procedure in highlight), 193
 - Style_Rule (struct in highlight), 183
 - Style_Rule.definition (field in highlight), 184
 - Style_Rule.kind (field in highlight), 183
 - 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 pyscan), 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), 150
- 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), 187
- Token_Kind.Comment_Multiline (enumerator in highlight), 187
- Token_Kind.Comment_PreProc (enumerator in highlight), 188
- Token_Kind.Comment_Preproc (enumerator in highlight), 187
- Token_Kind.Comment_Preproc_File (enumerator in highlight), 187
- Token_Kind.Comment_Single (enumerator in highlight), 187
- Token_Kind.Comment_Special (enumerator in highlight), 187
- 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), 188
- Token_Kind.Keyword (enumerator in highlight), 184
- Token_Kind.Keyword_Constant (enumerator in highlight), 184
- Token_Kind.Keyword_Declaration (enumerator in highlight), 184
- Token_Kind.Keyword_Namespace (enumerator in highlight), 184
- Token_Kind.Keyword_Pseudo (enumerator in highlight), 184

- 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), 188
- 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), 189
- Token_Kind.Name_Builtin_Pseudo
(enumerator in highlight), 185
- Token_Kind.Name_Builtin_Repeat_Type
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Scale
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Scheme_Builtin
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Scheme_Function
(enumerator in highlight), 189
- Token_Kind.Name_Builtin_Translator
(enumerator in highlight), 189
- 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), 185
- Token_Kind.Name_Lvalue (enumerator in
highlight), 188
- Token_Kind.Name_Namespace
(enumerator in highlight), 185
- Token_Kind.Name_Other (enumerator in
highlight), 185
- Token_Kind.Name_Property (enumerator
in highlight), 185
- Token_Kind.Name_Tag (enumerator in
highlight), 185
- Token_Kind.Name_Variable (enumerator
in highlight), 185
- 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), 184
- Token_Kind.Pitch (enumerator in
highlight), 190
- Token_Kind.Punctuation (enumerator in
highlight), 187
- Token_Kind.Punctuation_Indicator
(enumerator in highlight), 188
- 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), 186
 - Token_Kind.String_Heredoc (enumerator in highlight), 186
 - Token_Kind.String_Interpol (enumerator in highlight), 186
 - Token_Kind.String_Other (enumerator in highlight), 186
 - Token_Kind.String_Regex (enumerator in highlight), 186
 - Token_Kind.String_Single (enumerator in highlight), 186
 - 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), 189
 - 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), 165
 - translation (procedure in gettext), 165
 - 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), 194
- ## 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), 193
 write_figure_filename (procedure in i18n), 231
 write_html (procedure in highlight), 193
 write_html_lines (procedure in highlight), 193
 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), 159
 Write_Options.obsolete (field in gettext), 160
 Write_Options.previous (field in gettext), 160
 Write_Options.references (field in gettext), 159
 Write_Options.width (field in gettext), 159
 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