

Guidedog मैनुअल

मैनुअल और संपूर्ण Odin API संदर्भ

01 / स्रोत
अर्थ के साथ लिखें।

02 / संरचना
संबंध स्पष्ट करें।

03 / प्रकाशन
स्क्रीन और कागज़ पर पढ़ें।

अनुक्रम

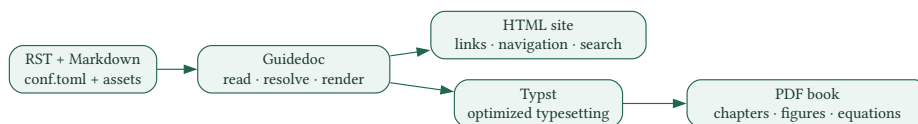
ट्यूटोरियल	3
आपकी पहली परियोजना	3
एक छोटी किताब बनाएँ	4
विषय मार्गदर्शिकाएँ	7
दस्तावेज़ लिखना	7
डोमेन	12
परियोजना कैसे काम करती है	16
गणित और प्रमाण	17
समझाने वाले आरेख	18
पढ़ने के लिए टाइपोग्राफी	19
टेम्पलेट	20
चित्र, वीडियो, ऑडियो और डाउनलोड	26
अंतरराष्ट्रीयकरण	27
अन्य परियोजनाओं से लिंक	30
ऑब्जेक्ट टाइप घोषित करना	31
autodoc से Python का दस्तावेज़ीकरण	32
Odin के दस्तावेज़ बनाना	34
व्यावहारिक काम	39
Guidedog बिल्ड और इंस्टॉल करना	39
दस्तावेज़ बिल्ड और प्रकाशित करना	40
Sphinx परियोजना का माइग्रेशन	41
कोडबेस के दस्तावेज़ बनाना	41
इन दस्तावेज़ों का रखरखाव	42
संदर्भ	43
कमांड लाइन संदर्भ	43
कॉन्फ़िगरेशन संदर्भ	47
रीडर, रेंडरर और बिल्डर	54
त्रुटि संदेश	55
Sphinx के साथ संगतता	58
आंतरिक कार्यप्रणाली	63
Guidedoc इंजन	63
एल्गोरिद्म से पहले इनवेरिएंट	64
एक पूरी आउटपुट पीढ़ी प्रकाशित करें	65
मेमोरी और स्वामित्व	66
इस मैनुअल के बारे में	69
बिल्ड के सत्यापन का रिकॉर्ड	69
Guidedog	71
परियोजना शुरू करें	71
स्थापित करें	71

स्रोत से बिल्ड करें	72
Guidedog से अनुवाद	72
पढ़ें और एम्बेड करें	73
श्रेय और लाइसेंस	73
API Reference	75
autodoc	76
cli	80
core	82
defaults	136
fetch	137
gds	139
gettext	145
graphdog	161
highlight	168
highlight/guidedoc	188
host	189
i18n	216
inventory	225
jinja	228
odindoc	242
project	250
pyscan	274
readers/commonmark	289
readers/rst	301
readers/rst/inline	319
readers/sphinx	324
render/html	331
render/text	332
render/typst	332
treесitter	333
typst	339
इंडेक्स	345

एक बार लिखें। अच्छी तरह समझाएँ। वेब और कागज़ पर प्रकाशित करें।

Guidedog, reStructuredText और Markdown स्रोतों से HTML वेबसाइट और PDF पुस्तकें बनाता है। इसके लिए Python वातावरण या अलग से LaTeX स्थापित करने की ज़रूरत नहीं है। इसके रूपांतरण इंजन का नाम **Guidedoc** है। Typst PDF की पृष्ठ-सज्जा करता है और Guidedog वेबसाइट सीधे बनाता है।

छोटी परियोजना को तीन चीज़ें चाहिए: स्रोत दस्तावेज़, विषय-सूची और `conf.toml`। आपकी पहली परियोजना से शुरू करें। किसी खास काम के लिए व्यावहारिक काम और ठीक अनुबंध के लिए संदर्भ देखें।



चित्र 1 एक स्रोत परियोजना, दो प्रकाशन।

ट्यूटोरियल

इन पृष्ठों को क्रम से पढ़ें। हर कदम ऐसा परिणाम देता है जिसे आप जाँच सकते हैं।

आपकी पहली परियोजना

लक्ष्य छोटा है: एक पृष्ठ ब्राउज़र में और वही पृष्ठ PDF में। पहले Guidedog तैयार करें। PDF के लिए अंतर्निहित कंपाइलर या स्थापित Typst 0.15.1 या नया संस्करण इस्तेमाल कर सकते हैं। Guidedog बिल्ड और इंस्टॉल करना देखें।

macOS पर `brew install insanai/guidedog/guidedog` से रिलीज़ स्थापित करें। यह आधिकारिक tap जोड़ता है और PDF कंपाइलर भी स्थापित करता है।

परियोजना बनाएँ

ऐसी निर्देशिका से चलाएँ जहाँ अभी notes न हो:

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

Guidedog कॉन्फ़िगरेशन, पहला RST दस्तावेज़ और संपादन योग्य टेम्पलेट लिखता है। चित्र, ऑडियो और वीडियो के फ़ोल्डर भी बनाता है। मौजूदा परियोजना पर नहीं लिखता।

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

RST फ़ाइलों में सामग्री और टेम्पलेट में प्रस्तुति रहती है। `quickstart` मूल `_templates/layout.html` और `_templates/book.typ` को परियोजना में ही रखता है, बाहरी थीम या कैश में नहीं। इन्हें कभी भी देख और बदल सकते हैं।

पहला पृष्ठ लिखें

`notes/index.rst` इससे बदलें:

```
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.

```

बिल्ड और निरीक्षण

```

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

```

notes/_build/html/index.html और notes/_build/pdf का PDF खोलें। -w चेतावनी पर बिल्ड विफल करता है; इसे दस्तावेज़ जाँच में रखें। HTML पृष्ठ में नेविगेशन और पाठ हैं। PDF में किताब के पृष्ठ-संबंधी तत्व जुड़ते हैं। गणित हर आउटपुट के लिए सजता है। सेटअप गणित और प्रमाण में है।

बदलाव करें

```

guidedog serve notes

```

कमांड का दिखाया पता खोलें। स्रोत बदलकर सहेजें और ब्राउज़र रिफ्रेश करें। HTML भेजने से पहले Guidedog स्रोत के बदलाव का समय जाँचता है और नया होने पर फिर बिल्ड करता है। यह ब्राउज़र को रीलोड नहीं भेजता। सर्वर Ctrl+C से रोकें।

कुछ बदले बिना फिर बिल्ड करें। अपरिवर्तित दस्तावेज़ फिर इस्तेमाल होते हैं। आउटपुट नया है यह साबित करने के लिए हर पृष्ठ फिर पढ़ना ज़रूरी नहीं। एक छोटी किताब बनाएँ पर आगे बढ़ें।

एक छोटी किताब बनाएँ

साइट दस्तावेज़ों का ग्राफ़ है। किताब को पढ़ने का क्रम चाहिए। toctree दोनों देता है।

अध्याय जोड़ें

notes/measurement.rst बनाएँ:

```

Measurement
=====

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

.. _measurement-units:

Units
-----

Keep one unit within a comparison.

```

notes/index.rst की सामग्री इससे बदलें:


```
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``.

दस्तावेज़ के नाम से `.rst` हटता है। लेबल दस्तावेज़ के भीतर किसी विचार को नाम देता है। अध्याय के लिए दस्तावेज़ लिंक और किसी खास तर्क के लिए लेबल इस्तेमाल करें।

आरेख जोड़ें

यह `measurement.rst` में डालें:

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

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

Guidedog अपनी लिंक की Graphviz लाइब्रेरी इस्तेमाल करता है। HTML में SVG और PDF में चित्र मिलता है। ब्राउज़र में आरेख इंटरप्रेटर नहीं चाहिए। कैप्शन निष्कर्ष समझाता है। वैकल्पिक पाठ चित्र समझाता है।

किताब बिल्ड करें

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

अध्याय साइडबार और किताब की विषय-सूची में है। ब्राउज़र में खंड लिंक खोलें। PDF में भी उसका लक्ष्य जाँचें।

अगला उपयोगी कदम दस्तावेज़ लिखना है। हर अध्याय इतना छोटा रखें कि एक विषय समझाए।

विषय मार्गदर्शिकाएँ

ये अध्याय मॉडल समझाते हैं। निर्णय लेने के लिए ज़रूरी संदर्भ भी देते हैं।

दस्तावेज़ लिखना

Guidedog reStructuredText (.rst) और MyST Markdown (.md) संभालता है। दोनों का एक साझा अर्थपूर्ण दस्तावेज़ ट्री बनता है, जिससे HTML साइट और PDF पुस्तक बनती हैं।

स्रोत भाषा चुनना

डोमेन, जटिल तालिका और विस्तार योग्य निर्देश चाहिए तो reStructuredText चुनें। Markdown सिंटेक्स के साथ Sphinx जैसे निर्देश और रोल चाहिए तो MyST Markdown चुनें।

परियोजना में Markdown, MyST के रूप में पढ़ा जाता है। एक फ़ाइल के `guidedog convert` रूपांतरण में CommonMark डिफ़ॉल्ट है, जब तक MyST स्पष्ट रूप से न चुनें।

शीर्षक और संरचना

दस्तावेज़ का शीर्षक पृष्ठ की पहचान देता है; खंड पाठ को व्यवस्थित करते हैं। RST में शीर्षक के नीचे चिह्नों की रेखा लगती है; ऊपर भी वैसी रेखा लगा सकते हैं। रेखा शीर्षक से छोटी न हो।

शीर्षक के लिए सुझाए गए चिह्न:

```
Document Title
=====

Section Heading
-----

Subsection
~~~~~

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

Paragraph Heading
.....
```

पूरी परियोजना में शीर्षक के स्तर एक जैसे रखें। MyST Markdown में #, ##, ### इस्तेमाल करें।

पाठ का रूप और इनलाइन मार्कअप

इनलाइन मार्कअप अर्थ बताने के लिए इस्तेमाल करें, केवल सजावट के लिए नहीं:

तालिका 1 इनलाइन मार्कअप

शैली	reStructuredText	MyST Markdown
मोटे अक्षर	<code>**strong text**</code>	<code>**strong text**</code>
ज़ोर /तिरछे अक्षर	<code>*emphasized text*</code>	<code>*emphasized text*</code>
इनलाइन कोड	<code>`inline code`</code>	<code>`inline code`</code>
सबस्क्रिप्ट	<code>:sub:`text`</code>	<code>{sub}`text`</code>
सुपरस्क्रिप्ट	<code>:sup:`text`</code>	<code>{sup}`text`</code>

तारे या बैकटिक को साधारण अक्षर की तरह लिखने के लिए बैकस्लैश से एस्केप करें:

```
\*not italic\*
```

सूचियाँ

Guidedog चार तरह की सूचियाँ संभालता है:

बुलेट सूची

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

क्रमांकित सूची

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

परिभाषा सूची

परिभाषा सूची में शब्द और उसका अर्थ साथ रहते हैं। शब्द एक पंक्ति में लिखें और अगली पंक्ति से इंडेंट किया अर्थ दें:

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

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

फ़ील्ड सूची

फ़ील्ड सूची से व्यवस्थित मेटाडेटा और पैरामीटर का वर्णन दें:

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

कोड ब्लॉक

सिंटैक्स हाइलाइटिंग के लिए `code-block` या `sourcecode` इस्तेमाल करें। Guidedog 35 से अधिक प्रोग्रामिंग भाषाएँ संभालता है:

```
.. 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)
```

कोड ब्लॉक के विकल्प:

- `:linenos:`: कोड के साथ पंक्ति नंबर दिखाता है।
- `:caption: Title:` कोड ब्लॉक के ऊपर या नीचे कैप्शन जोड़ता है।
- `:emphasize-lines: 1, 3-5:` चुनी हुई पंक्तियों पर ज़ोर देता है।
- `:name: label:` कोड ब्लॉक को संदर्भ का लक्ष्य देता है।

बाहरी स्रोत फ़ाइल सीधे शामिल करने के लिए `literalinclude` इस्तेमाल करें:

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

सूचना ब्लॉक

सूचना ब्लॉक नोट, चेतावनी और अतिरिक्त सलाह उभारते हैं:

```
.. 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.
```

`Guidedog danger`, `error`, `hint` और `attention` भी संभालता है। अपना शीर्षक देने के लिए `admonition` इस्तेमाल करें:

```
.. admonition:: Design Rationale

    Explains why a particular architecture was chosen.
```

तालिकाएँ

`Guidedog` सरल, ग़्रिड और निर्देशों से बनी तालिकाएँ संभालता है।

सरल तालिकाएँ

सरल तालिका में क्षैतिज रेखाएँ कॉलम की सीमा बताती हैं:

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

ग्रिड तालिकाएँ

ग्रिड तालिका में कई पंक्तियों वाले सेल और कई कॉलम या पंक्तियाँ घेरने वाले सेल बन सकते हैं:

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

सूची से बनी तालिकाएँ

`list-table` नेस्ट की गई बुलेट सूचियों से तालिका बनाता है। इससे चौड़ी तालिका का स्रोत पढ़ना और संस्करण नियंत्रण में सँभालना आसान होता है:

```
.. 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 तालिकाएँ

`csv-table` कॉमा से अलग किए मानों से तालिका बनाता है:

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

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

चित्र और आकृतियाँ

चित्र, स्क्रीनशॉट और आरेख जोड़ने के लिए `image` और `figure` इस्तेमाल करें:

```
.. 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` चित्र के साथ कैप्शन और वैकल्पिक संदर्भ लक्ष्य जोड़ता है।

विषय-सूची ट्री से दस्तावेज़ की संरचना

`toctree` दस्तावेज़ का पदानुक्रम, नेविगेशन मेनू और HTML तथा PDF का पढ़ने का क्रम तय करता है:

```
.. toctree::
   :maxdepth: 2
```

```
:caption: User Guide
:numbered:
```

```
installation
quickstart
configuration
```

toctree के सामान्य विकल्प:

- `:maxdepth: N`: विषय-सूची में शामिल शीर्षकों की गहराई।
- `:caption: Title`: नेविगेशन प्रविष्टियों के ऊपर दिखने वाला वर्ग शीर्षक।
- `:numbered::` अध्याय और शीर्षकों को क्रमांक देता है।
- `:titlesonly::` केवल दस्तावेज़ के मुख्य शीर्षक दिखाता है, भीतर के खंड नहीं।
- `:hidden::` पाठ में सूची दिखाए बिना पढ़ने का क्रम दर्ज करता है।
- `:glob:: tutorials/*` जैसे वाइल्डकार्ड से दस्तावेज़ चुनने देता है।

पार-संदर्भ और लिंक

स्पष्ट लक्ष्य लेबल से संदर्भ फ़ाइल के नाम से स्वतंत्र और स्थिर रहता है।

लक्ष्य लेबल और `:ref:`

शीर्षक, तालिका या चित्र के ठीक पहले लक्ष्य लेबल रखें:

```
.. _storage-model:

Storage model
-----

Caller storage is bounded and measured upfront.
```

परियोजना के किसी भी दस्तावेज़ से इस लक्ष्य का संदर्भ दे सकते हैं:

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

`:doc:` से दस्तावेज़ संदर्भ

एक्सटेंशन छोड़े हुए पथ से दूसरे दस्तावेज़ का लिंक दें:

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

`:term:` से शब्द का संदर्भ

glossary से शब्दों की परिभाषा दें:

```
.. 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:` रोल से शब्द का लिंक दें:

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

`:download:` से फ़ाइल डाउनलोड

फ़ाइल को डाउनलोड फ़ोल्डर में कॉपी कर लिंक बनाता है:

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

बाहरी लिंक

बाहरी वेब पते का लिंक दें:

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

प्रतिस्थापन और शामिल करना

दोबारा इस्तेमाल होने वाला वाक्यांश, चिह्न या चित्र तय करें:

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

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

कई फ़ाइलों में reStructuredText सामग्री साझा करने के लिए:

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

MyST Markdown लिखना

MyST Markdown में फेंस ब्लॉक से वही निर्देश और रोल व्यक्त किए जाते हैं:

```
# 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`.
```

डोमेन

डोमेन निर्देशों और रोल का समूह है। इससे किसी भाषा या विषय के प्रोग्राम ऑब्जेक्ट का वर्णन और पार-संदर्भ दिया जाता है।

Guidedog में ये Sphinx डोमेन मूल रूप से उपलब्ध हैं:

- **मानक डोमेन** (`std`): कमांड लाइन विकल्प, वातावरण चर, शब्द और UI लेबल जैसे सामान्य ऑब्जेक्ट।
- **Python डोमेन** (`py`): मॉड्यूल, क्लास, फ़ंक्शन, मेथड, एट्रिब्यूट, प्रॉपर्टी और एक्सेप्शन।
- **C डोमेन** (`c`): फ़ंक्शन, टाइप, स्ट्रक्ट, यूनियन, एनम, एनम मान, चर और मैक्रो।

- **C++ डोमेन** (`cpp`): क्लास, कॉन्सेप्ट, टेम्पलेट, नेमस्पेस, मेथड और एक्सप्रेसन।
- **JavaScript डोमेन** (`js`): फ़ंक्शन, मेथड, क्लास और प्रॉपर्टी।
- **Odin डोमेन** (`odin`): Odin पैकेज, प्रोसीजर, स्ट्रक्ट, यूनियन और एनम। [Odin के दस्तावेज़ बनाना देखें](#)।

मुख्य डोमेन तय करना

`conf.toml` के `primary_domain` से डिफ़ॉल्ट डोमेन तय होता है; डिफ़ॉल्ट `"py"` है। इसके बाद उपसर्ग छोड़ सकते हैं: `.. py:function::` की जगह `.. function::` और `:py:func:`name`` की जगह `:func:`name`` लिखें।

दस्तावेज़ में स्थानीय रूप से डोमेन बदलने के लिए:

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

मानक डोमेन

मानक डोमेन कमांड लाइन टूल, वातावरण चर और दस्तावेज़ पार-संदर्भ का वर्णन करता है।

निर्देश

```
.. program:: mytool

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

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

रोल

- `:doc:`path``: सापेक्ष पथ से परियोजना के दस्तावेज़ का लिंक देता है।
- `:ref:`label``: स्पष्ट लक्ष्य लेबल `.. _label:` का लिंक देता है।
- `:term:`term``: glossary में परिभाषित शब्द का लिंक देता है।
- `:option:`--config``: `option` में बताए कमांड लाइन विकल्प का लिंक देता है।
- `:envvar:`VARIABLE``: `envvar` में बताए वातावरण चर का लिंक देता है।
- `:command:`name``: सिस्टम कमांड का नाम दर्शाता है।
- `:file:`path``: फ़ाइल या फ़ोल्डर का पथ दर्शाता है; `{variable}` प्लेसहोल्डर भी चलता है।
- `:kbd:`Ctrl+C``: कीबोर्ड की कुंजियाँ दर्शाता है।
- `:menuselection:`File --> Save As``: मेनू चयन का क्रम दिखाता है।
- `:guilabel:`Submit``: बटन या UI तत्व का नाम दिखाता है।
- `:pep:`8``: Python Enhancement Proposal का लिंक देता है।
- `:rfc:`7231``: IETF RFC दस्तावेज़ का लिंक देता है।

Python डोमेन

Python डोमेन Python इंटरफ़ेस और सिग्नेचर का वर्णन करता है।

निर्देश

```
.. 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.

```

निर्देश हैं: `py:module`, `py:currentmodule`, `py:function`, `py:data`, `py:class`, `py:method`, `py:staticmethod`, `py:classmethod`, `py:attribute`, `py:property`, `py:exception`, `py:decorator` और `py:type`।

रोल

- `:py:func:`name``: Python फ़ंक्शन का लिंक देता है।
- `:py:class:`name``: Python क्लास का लिंक देता है।
- `:py:meth:`name``: Python क्लास या इंस्टेंस मेथड का लिंक देता है।
- `:py:attr:`name``: Python एट्रिब्यूट का लिंक देता है।
- `:py:data:`name``: Python मॉड्यूल-स्तरीय डेटा का लिंक देता है।
- `:py:exc:`name``: Python एक्सेप्शन क्लास का लिंक देता है।
- `:py:mod:`name``: Python मॉड्यूल का लिंक देता है।

C डोमेन

C डोमेन C लाइब्रेरी और हेडर का वर्णन करता है।

निर्देश

```

.. 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.

```

रोल

- `:c:func:`name``: C फ़ंक्शन का लिंक देता है।
- `:c:member:`name``: C स्ट्रक्चर या यूनियन के सदस्य का लिंक देता है।
- `:c:data:`name`` / `:c:var:`name``: C चर का लिंक देता है।
- `:c:type:`name``: C typedef या टाइप का लिंक देता है।

- `:c:struct:`name``: C स्ट्रक्ट का लिंक देता है।
- `:c:union:`name``: C यूनियन का लिंक देता है।
- `:c:enum:`name``: C एनम का लिंक देता है।
- `:c:macro:`name``: C प्रीप्रोसेसर मैक्रो का लिंक देता है।

C++ डोमेन

C++ डोमेन आधुनिक C++ संरचनाएँ, कॉन्सेप्ट, नेमस्पेस और स्कोप संभालता है।

निर्देश

```
.. 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.
```

रोल

- `:cpp:class:`name``: C++ क्लास या स्ट्रक्ट का लिंक देता है।
- `:cpp:func:`name``: C++ फ़ंक्शन या मेथड का लिंक देता है।
- `:cpp:member:`name`` / `:cpp:var:`name``: सदस्य या चर का लिंक देता है।
- `:cpp:type:`name``: C++ टाइप उपनाम या typedef का लिंक देता है।
- `:cpp:concept:`name``: C++20 कॉन्सेप्ट का लिंक देता है।
- `:cpp:enum:`name``: C++ एनम का लिंक देता है।

JavaScript डोमेन

JavaScript डोमेन ब्राउज़र स्क्रिप्ट, लाइब्रेरी और Node.js मॉड्यूल का वर्णन करता है।

निर्देश

```
.. js:module:: theme

.. js:class:: ThemeController(options)

    Manages color themes and local storage persistence.

.. js:method:: toggleTheme()

    Toggles between light and dark modes.
```

रोल

- `:js:func:`name``: JavaScript फ़ंक्शन का लिंक देता है।
- `:js:meth:`name``: JavaScript मेथड का लिंक देता है।
- `:js:class:`name``: JavaScript क्लास का लिंक देता है।
- `:js:data:`name``: JavaScript चर का लिंक देता है।
- `:js:attr:`name``: JavaScript ऑब्जेक्ट प्रॉपर्टी का लिंक देता है।
- `:js:mod:`name``: JavaScript मॉड्यूल का लिंक देता है।

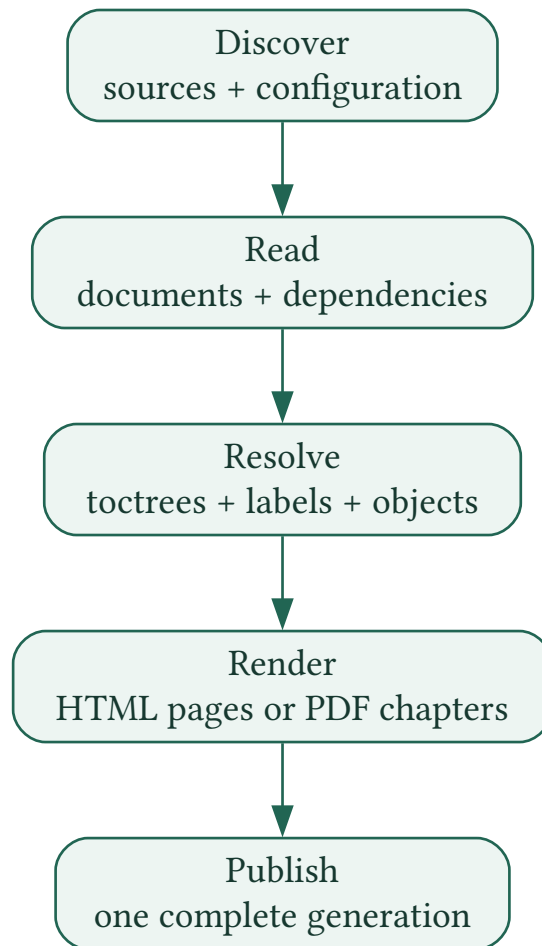
पार-संदर्भ की परंपराएँ

डोमेन के पार-संदर्भ में ये संशोधक इस्तेमाल कर सकते हैं:

- **उपसर्ग छिपाना:** लक्ष्य से पहले `~` लगाने पर नाम का आखिरी भाग ही दिखता है। जैसे `:py:func:`~pipeline.reader.DocumentReader.parse`` से `parse()` दिखता है।
- **वर्तमान स्कोप में खोज:** सापेक्ष संदर्भ पहले वर्तमान मॉड्यूल या नेमस्पेस में खोजता है, फिर वैश्विक खोज करता है।
- **सटीक मिलान:** पहले `.` लगाने पर खोज वर्तमान मॉड्यूल या घेरने वाले ऑब्जेक्ट से शुरू होती है।

परियोजना कैसे काम करती है

प्रोजेक्ट मूल `conf.toml` वाला फ़ोल्डर है। `root_doc` पढ़ने का पहला दस्तावेज़ है। Guidedog सोर्स खोजता, पढ़ता, संबंध हल करता और पूरा आउटपुट संस्करण प्रकाशित करता है।



चित्र 2 बिल्ड खोज, अर्थ, प्रस्तुति और प्रकाशन को अलग रखता है।

दस्तावेज़ और संसाधन

प्रत्यय रीडर चुनता है। RST और प्रोजेक्ट Markdown एक ग्राफ़ साझा करते हैं। चित्र व डाउनलोड संसाधन हैं; उपयोग किए संसाधन कॉपी होते हैं। थीम सामग्री `html_static_path` में है। `include` सोर्स या दिए `include_roots` के भीतर हो।

अलग परियोजनाएँ

मैनुअल और डिज़ाइन रिकॉर्ड अलग प्रोजेक्ट रखें। पाठक और पढ़ने का क्रम अलग हैं। हर एक का अपना कॉन्फ़िगरेशन, टेम्पलेट, कैश और आउटपुट है। Guidedog में `docs/manual` और `docs/gds` ऐसे हैं।

इन्क्रिमेंटल बिल्ड

Guidedog हर आउटपुट के इनपुट और खोज दर्ज करता है। बदला सोर्स फिर पढ़ता है। निर्भरता बदलने पर उपयोगकर्ता अमान्य होते हैं। टेम्पलेट सभी पेज बदल सकता है। अपरिवर्तित दस्तावेज़ सहेजे ऑब्जेक्ट से लोड हो सकता है।

कैश पिछली बिल्ड का प्रमाण है, मूल सत्य नहीं। `-E` अगली बिल्ड के लिए सहेजा वातावरण हटाता है। `-a` सभी आउटपुट लिखता है। ये अलग समस्याएँ हल करते हैं।

पुनः उपयोग = समान इनपुट $\wedge$ समान खोज परिणाम $\wedge$ आउटपुट मौजूद

टेम्पलेट का स्पष्ट टाइमस्टैम्प भी पेज इनपुट है। वर्तमान मिनट दिखाता पेज पाठ बदले बिना बदल सकता है। तय समय के लिए `SOURCE_DATE_EPOCH` दें।

प्रकाशन

बिल्ड समान स्तर के स्टेजिंग फ़ोल्डर में बदलाव लिखता है। पूरा करके फ़ाइल और रिकॉर्ड स्थायी करता, फिर संस्करण बदलता है। पाठक पूरा पुराना या नया आउटपुट देखते हैं। अधूरा कमिट अगली बिल्ड सँभालती है। डिज़ाइन और प्रमाण दायित्व एक पूरी आउटपुट पीढ़ी प्रकाशित करें में हैं।

गणित और प्रमाण

समीकरण तब उपयोगी है जब संबंध समझना आसान हो। हर प्रतीक पहले परिभाषित करें। ज़रूरी जगह इकाई दें। समीकरण व्याख्या का सहारा हो, उसका विकल्प नहीं।

इनलाइन और अलग पंक्ति के सूत्र

छोटे व्यंजक के लिए `math` रोल और अलग समीकरण के लिए `math` निर्देश दें। दोनों प्रारूपों का साझा LaTeX गणित उपसमुच्चय लिखें।

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

.. math::
  :label: area-square

  A = s^2

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

भुजा s वाले वर्ग का क्षेत्रफल s^2 होता है।

$$A = s^2 \quad (1)$$

संबंध (1) है। लेबल वाला समीकरण स्थिर लक्ष्य पाता है। HTML MathJax मार्कअप देता है। PDF समर्थित LaTeX को Typst गणित में बदलता है। असमर्थित PDF कमांड की सूचना मिलती है। मनमाने LaTeX पैकेज नहीं चलते।

एक छोटा प्रमाण

दर्ज बाइट U , बजट सीमा B और माँगा आवंटन n हो। सभी गैरऋणात्मक और $U \leq B$ मानें।

स्वीकृति का नियम। अनुरोध तभी स्वीकार करें जब

$$n \leq B - U. \quad (2)$$

कथन। स्वीकृत अनुरोध $U + n \leq B$ बनाए रखता है।

प्रमाण। (2) के दोनों ओर U जोड़ें तो $U + n \leq B$ है। कार्यान्वयन घटाव रूप जाँचता है। पहले $U + n$ नहीं निकालता, जो मशीन पूर्णांक में ओवरफ्लो कर सकता है। प्रमाण और गणना क्रम समान हैं।

सीमा। यह दर्ज होस्ट स्टोरेज का हिसाब है। अलग Typst कंपाइलर का आंतरिक आवंटन सीमित नहीं। प्रमाण अपनी बताई सीमा तक ही मान्य है।

HTML विकल्प

इन प्रोजेक्ट में MathJax 3.2.2 और SVG फ्रॉन्ट डेटा साथ हैं। केवल गणित वाले पेज इसे लोड करते हैं। बिल्ड के बाद पेज ऑफ़लाइन पढ़े जा सकते हैं। चित्र स्क्रिप्ट या वेब फ्रॉन्ट सेवा नहीं चाहिए।

आपके अपने प्रोजेक्ट में डिफ़ॉल्ट रूप से MathJax CDN से आता है। स्थानीय प्रति देने के लिए उसे `_static` में रखें और सेट करें:

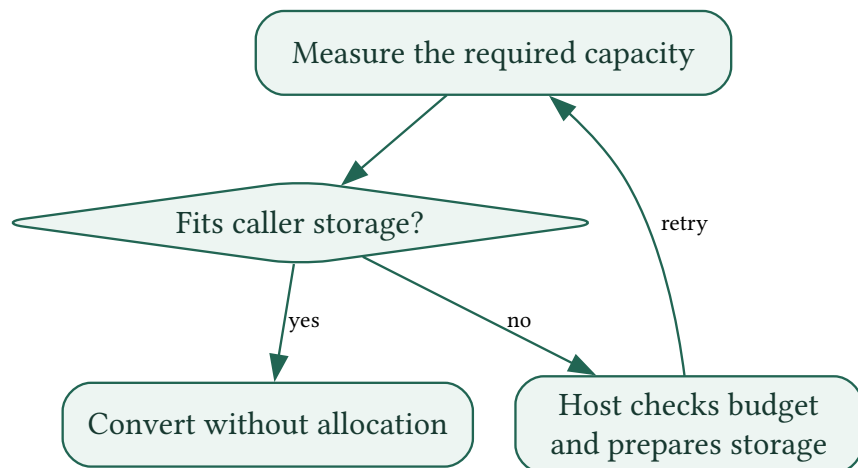
```
html_static_path = ["_static"]
mathjax_path = "_static/tex-svg-full.js"
```

वेब समीकरण सजाने के लिए JavaScript चाहिए। बिना इसके सोर्स दिखता है। PDF गणित ब्राउज़र JavaScript पर निर्भर नहीं। [MathJax घटक अनुबंध देखें](#)।

समझाने वाले आरेख

प्रश्न के अनुसार आरेख चुनें। फ्लोचार्ट चुनाव, ब्लॉक आरेख ज़िम्मेदारी, क्रम-आरेख क्रम और ग्राफ़ संबंध बताता है। गद्य अधिक सीधे उत्तर दे तो गद्य लिखें।

फ्लोचार्ट



चित्र 3 स्टोरेज उधार लेने या बढ़ाने से पहले क्षमता जाँची जाती है।

चित्र बढ़ाने के चरण में मालिक बताता है। वह कोर द्वारा आवंटन नहीं दिखाता। यह अंतर अनुबंध का हिस्सा है।

ग्राफ़ लिखना

```
.. 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 और digraph अंतर्निहित हैं। Guidedog Graphviz से रेंडर करता है। HTML में SVG और PDF में बना चित्र आता है। दोनों का ग्राफ़ सोर्स समान है।

परिणाम पठनीय बनाएँ

नोड लेबल छोटे रखें। लंबा नियम चित्र के नीचे समझाएँ। मुख्य पथ एक दिशा में रखें; लौटती कड़ी पर शर्त दें। रंग के साथ आकार और पाठ भी दें। प्रिंट में रंग न हो सकता है।

वैकल्पिक पाठ और कैप्शन दें। संकरे ब्राउज़र और PDF पेज जाँचें। फिट करने के लिए बहुत छोटे अक्षर चाहिए हों तो चित्र दो में बाँटें। आकार के लिए चित्र, वीडियो, ऑडियो और डाउनलोड देखें।

पढ़ने के लिए टाइपोग्राफी

पढ़ने से पहले पेज की संरचना देखें। स्पष्ट शीर्षक, शांत नेविगेशन और संतुलित पाठ चौड़ाई मुख्य काम करते हैं। रंग संबंध या क्रिया बताएँ; अकेला अर्थ न ढोएँ।

वेब पृष्ठ

कस्टम लेआउट में प्रोजेक्ट नेविगेशन, दस्तावेज़ और वर्तमान पेज रूपरेखा के तीन भाग हैं। छोटी स्क्रीन पर नेविगेशन ड्रॉअर है। Libertinus Serif और Sans साथ हैं। शीर्षक बोल्ड सेरिफ़ में, नेविगेशन, लेबल और नियंत्रण उसके sans साथी में हैं। संपीड़ित WOFF2 साइट से मिलते हैं; फ्रॉन्ट सेवा अनुरोध नहीं होता।

हल्का थीम डिफ़ॉल्ट है। गहरा और सिस्टम मोड भी हैं। चुनाव उसी ब्राउज़र origin पर याद रहता है। सिस्टम मोड ऑपरेटिंग सिस्टम अपनाता है। HTML प्रिंट हल्के रंग में है। JavaScript के बिना भी सामग्री और नेविगेशन काम करते हैं; थीम और बेहतर खोज अतिरिक्त संवर्द्धन हैं।

अंग्रेज़ी गद्य में ब्राउज़र पूरा अनुच्छेद देखकर **Knuth–Plass पंक्ति-विभाजन** कर सकता है। छोटा स्क्रिप्ट लोड फ्रॉन्ट मापकर बॉक्स, स्पेस और हाइफ़न दंड बनाता है; पूरे अनुच्छेद के विराम चुनता है। Frank Liang की पैटर्न विधि अंग्रेज़ी शब्द-विभाजन विकल्प देती है। मूल TeX पैटर्न और लाइसेंस साइट में हैं।

संवर्द्धन ज़ोर और मूल पाठ रखता है; चौड़ाई बदले तो फिर गणना करता है। एक अनुकूलित अनुच्छेद के भीतर कॉपी से मूल पाठ मिलता है, दृश्य पंक्ति-विभाजन या नए हाइफ़न नहीं। प्रिंट पर सामान्य अनुच्छेद लौटता है, ताकि पेज लेआउट उसे बाँटे।

लिंक, गणित, कोड, संवादात्मक सामग्री, संकरी चौड़ाई और दूसरी भाषाएँ ब्राउज़र लेआउट इस्तेमाल करती हैं। उचित अंतर वाला हल न मिले तो भी यही है। JavaScript बंद होने पर साइट पठनीय है। विकल्प hyphens: auto और text-wrap: pretty भाषा के अनुसार हैं; केवल ये CSS Knuth–Plass की गारंटी नहीं।

PDF पृष्ठ

पुस्तक में वेब वाला *Libertinus* है। अनुच्छेद दोनों ओर बराबर, पंक्ति-विभाजन अनुकूलित और हाइफ़न भाषा के अनुसार हैं। अंतर, शीर्षलेख, कैप्शन, कोड और तालिका रेखाएँ भी नियंत्रित हैं। फ़ॉन्ट, विभाजन लागत और चौड़ाई साथ काम करते हैं। ढीले अनुच्छेद को ज़बरन लाइन तोड़कर सुधारना शायद ही उचित है।

Typst PDF अनुच्छेद अनुकूलित करता है; वेब संवर्द्धन योग्य गद्य करता है। सोर्स, फ़ॉन्ट और दृश्य क्रम साझा हैं, लेकिन लाइन विराम अलग हो सकते हैं। Typst पाठ अनुबंध और ब्राउज़र हाइफ़न अनुबंध देखें।

डिज़ाइन बदलना

दृश्य मान `_static/editorial.css`, संरचना `_templates/layout.html` और प्रिंट `_templates/book.typ` में बदलें। पढ़ने का क्रम, फ़ोकस संकेत, लेबल और पर्याप्त कंट्रास्ट बनाए रखें।

बदलाव पर दोनों प्रारूप बनाएँ। कोड, तालिका, आरेख और गणित वाला पेज देखें। सुंदर कवर पूरी पुस्तक के सही होने का प्रमाण नहीं। पूरा टेम्पलेट अनुबंध टेम्पलेट और API [API Reference](#) तथा उसी PDF में है।

टेम्पलेट

टेम्पलेट प्रस्तुति तय करता है; स्रोत के पाठ की नकल नहीं करता। Guidedog HTML के लिए Jinja और PDF के लिए Typst इस्तेमाल करता है।

Guidedog के टेम्पलेट परियोजना की सामान्य फ़ाइलें हैं, जिन्हें सीधे देखा और संपादित किया जा सकता है। `guidedog quickstart` काम में आने वाले टेम्पलेट परियोजना में ही लिखता है:

- `_templates/layout.html` में Jinja से HTML साइट की संरचना तय होती है।
- `_templates/book.typ` में Typst से PDF पुस्तक की पृष्ठ-सज्जा, अक्षर-विन्यास और आवरण तय होते हैं।
- `_static/guidedog.css` और `_static/guidedog.js` डिफ़ॉल्ट स्टाइल और ब्राउज़र के व्यवहार देते हैं।

टेम्पलेट बाहरी पैकेज या बाइनरी कैश में छिपे नहीं रहते; वे परियोजना में होते हैं। उन्हें संपादित करें, `guidedog build` से दोबारा बनाएँ और `guidedog serve` से देखें।

HTML टेम्पलेट

Guidedog अंतर्निहित Jinja इंजन से मुख्य `layout.html` टेम्पलेट रेंडर करता है। पूरा लेआउट एक बड़ी फ़ाइल में रखना ज़रूरी नहीं है। इसे छोटे टेम्पलेट, मैक्रो लाइब्रेरी और कई स्तरों की इनहेरिटेंस में बाँट सकते हैं।

मॉड्यूलर टेम्पलेट संरचना

`templates_path` के फ़ोल्डरों में टेम्पलेट को कई फ़ाइलों और उपफ़ोल्डरों में रख सकते हैं। डिफ़ॉल्ट मान `["_templates"]` है:

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



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

Guidedog `templates_path` के सभी फ़ाइल और उपफ़ोल्डर खोजता है। शामिल या इम्पोर्ट करने का पथ इन्हीं फ़ोल्डरों के सापेक्ष होता है:

कोड सूची 1 `_templates/layout.html`

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

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

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

Jinja मैक्रो भी अलग फ़ाइलों में परिभाषित करके ज़रूरत की जगह इम्पोर्ट कर सकते हैं:

कोड सूची 2 `_templates/macros/components.html`

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

कोड सूची 3 टेम्पलेट में मैक्रो का इस्तेमाल

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

!layout.html से थीम इनहेरिट करना

डिफ़ॉल्ट थीम के कुछ हिस्से बदलने के लिए पूरा पृष्ठ दोबारा लिखना ज़रूरी नहीं है। Sphinx वाली विस्मयादिबोधक सिटैक्स से अंतर्निहित लेआउट इनहेरिट कर सकते हैं:

कोड सूची 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" %}` या `{% extends "basic/layout.html" %}` से लोडर मूल थीम का टेम्पलेट पढ़ता है, परियोजना की `_templates/layout.html` को बार-बार नहीं पढ़ता।

बदले गए ब्लॉक में `{{ super() }}` बुलाने से मूल ब्लॉक का पाठ मिलता है। उसकी नकल किए बिना पहले या बाद में सामग्री जोड़ सकते हैं।

लेआउट ब्लॉक

अंतर्निहित `layout.html` में Sphinx की परंपरा के अनुसार ब्लॉक परिभाषित हैं:

| ब्लॉक | उद्देश्य |
|-------------|--|
| doctype | दस्तावेज़ प्रकार की घोषणा। डिफ़ॉल्ट <code><!DOCTYPE html></code> है। |
| htmltitle | <code><head></code> के भीतर <code><title></code> तत्व। |
| linktags | नेविगेशन और मेटा लिंक: <code>favicon</code> , <code>index</code> , <code>search</code> , <code>prev</code> , <code>next</code> । |
| css | स्टाइलशीट लिंक और इनलाइन CSS के मूल चर। |
| scripts | खोज इंडेक्स और इंटरैक्टिव सुविधाओं वाले JavaScript टैग। |
| extrahead | फ्रॉन्ट, विश्लेषण या अपने मेटा टैग जोड़ने के लिए <code><head></code> के अंत की जगह। |
| header | मुख्य नेविगेशन से ठीक पहले का हेडर स्थान। |
| relbar1 | नाम, संस्करण, खोज और थीम बदलने वाली ऊपरी नेविगेशन पट्टी। |
| rootrellink | संबंधित लिंक से पहले नेविगेशन में जोड़ने की जगह। |
| relbaritems | नेविगेशन पट्टी में अपने अतिरिक्त आइटम। |
| sidebar1 | बाई नेविगेशन पट्टी का कंटेनर। |
| sidebartoc | <code>sidebar1</code> के भीतर विषय-सूची का नेविगेशन ट्री। |
| breadcrumbs | मुख्य पाठ के ऊपर क्रमबद्ध नेविगेशन पथ। |
| document | मुख्य पाठ को घेरने वाला कंटेनर। |
| body | वर्तमान दस्तावेज़ का रेंडर किया HTML पाठ, <code>{{ body }}</code> । |
| relbar2 | पिछले और अगले अध्याय के लिंक वाला निचला नेविगेशन। |
| footer | कॉपीराइट, अद्यतन समय और स्रोत लिंक वाला पादलेख। |
| sidebar2 | वर्तमान पृष्ठ की रूपरेखा दिखाने वाली दाई पट्टी। |

टेम्पलेट खोजना

Guidedog `conf.toml` के `templates_path` वाले फ़ोल्डर क्रम से खोजता है, फिर अंतर्निहित टेम्पलेट देखता है। नाम इन फ़ोल्डरों के सापेक्ष होते हैं; `..` से बाहर नहीं जा सकते।

`templates_path` की हर फ़ाइल और उपफ़ोल्डर बिल्ड की निर्भरता है। टेम्पलेट जोड़ने, बदलने या हटाने पर `guidedog build` और `guidedog serve` बदलाव पहचानकर साइट दोबारा रेंडर करते हैं।

पृष्ठ के वेरिएबल

Sphinx के मौजूदा वेरिएबल नाम Guidedog भी रखता है, जिससे थीम के कुछ हिस्से लाए जा सकते हैं।

| वेरिएबल | मान |
|--|--|
| body | HTML में दस्तावेज़। |
| title | दस्तावेज़ का शीर्षक। |
| pagename, docname | दस्तावेज़ का नाम, जैसे <code>usage/install</code> । दस्तावेज़ न होने वाले पेज पर <code>docname</code> खाली और <code>pagename</code> पेज नाम है: Sphinx की तरह <code>genindex</code> , <code>py-modindex</code> , <code>search</code> या टेम्पलेट पेज का नाम। |
| toc | विषय-सूची वृक्ष से बना HTML नेविगेशन। |
| outline | दस्तावेज़ के खंड HTML में; खंड न हों तो खाली। |
| prev, next | पढ़ने के क्रम में पड़ोसी पृष्ठ, <code>url</code> (या <code>link</code>) और <code>title</code> सहित; आरंभ और अंत में पड़ोसी नहीं होता। |
| project, version, release, copyright, language | इसी नाम के विकल्प। |
| html_title, docstitle | <code>html_title</code> ; डिफ़ॉल्ट “ <code><project> <release> documentation</code> ”। |

| वेरिएबल | मान |
|--|--|
| <code>html_short_title</code> , <code>shorttitle</code> | <code>html_short_title</code> . |
| <code>root_doc</code> , <code>master_doc</code> | रूट दस्तावेज़ का नाम। |
| <code>path_to_root</code> | पृष्ठ से साइट के रूट का पथ, जैसे <code>../</code> । |
| <code>root_url</code> , <code>search_url</code> , <code>genindex_url</code> | रूट पृष्ठ, खोज पृष्ठ और सामान्य इंडेक्स के लिंक। |
| <code>css_files</code> , <code>js_files</code> | <code>url</code> वाली फ़ाइलों की सूची: पहले <code>Guidedog</code> की, फिर <code>html_css_files</code> और <code>html_js_files</code> । |
| <code>logo_url</code> , <code>favicon_url</code> | <code>_static</code> के भीतर <code>html_logo</code> और <code>html_favicon</code> ; न दिए हों तो खाली। |
| <code>accent</code> | थीम का रंग <code>html_theme_options.accent</code> । |
| <code>sourcelink_url</code> | स्रोत दिखाने पर <code>_sources</code> के भीतर दस्तावेज़ का स्रोत; अन्यथा खाली। |
| <code>last_updated</code> | <code>html_last_updated_fmt</code> हो तो उस रूप में बिल्ड की तारीख; अन्यथा खाली। |
| <code>show_copyright</code> , <code>show_sphinx</code> , <code>show_guidedog</code> , <code>has_source</code> , <code>show_source</code> | <code>html_show_*</code> और <code>html_copy_source</code> विकल्प। |
| <code>builder</code> , <code>file_suffix</code> | बिल्डर का नाम, जैसे <code>html</code> , और पृष्ठ का प्रत्यय। |
| <code>html_context</code> की हर कुंजी | मान और उसका प्रकार Sphinx के Python मानों जैसा है: <code>false</code> <code>{% if %}</code> में असत्य, संख्या संख्या, ऐरे सूची, और टेबल <code>conf.toml</code> के कुंजी क्रम वाला शब्दकोश है। <code>guidedog migrate conf.py</code> के <code>html_context</code> की लिटरल प्रविष्टियाँ लाता है। गणना से प्राप्त मान अपरिभाषित रहते हैं, जिन्हें टेम्पलेट असत्य पढ़ता है। |

`path_to` Sphinx का टेम्पलेट फ़ंक्शन है। `path_to("usage/install")` दस्तावेज़ पेज और `path_to("_static/logo.svg", 1)` साइट के नीचे फ़ाइल का URL देता है, वर्तमान पेज के सापेक्ष। `singlehtml` में दस्तावेज़ एक पेज का खंड है: पहला कॉल वहाँ `#document-usage-install` और खोज व सूचकांक पर `index.html#document-usage-install` देता है। `genindex` या टेम्पलेट पेज जैसे गैर-दस्तावेज़ नाम हर बिल्डर में साइट के मूल पेज हैं।

टेम्पलेट से बने पृष्ठ

`html_additional_pages` Sphinx की तरह केवल टेम्पलेट से पेज बनाता है। कुंजी पेज नाम है और मान `templates_path` का टेम्पलेट:

कोड सूची 5 `conf.toml`

```
root_doc = "contents"

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

पेज टेम्पलेट आम तौर पर `layout.html` extend करके उसके ब्लॉक भरता है, ताकि रूप बाकी पेजों जैसा हो:

कोड सूची 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 %}
```

टेम्पलेट गैर-दस्तावेज़ पेज के चर पाता है: `pagename` पेज नाम (`index`), `title` और `body` खाली हैं। `pathto`, `toc`, `toctree()` और `html_context` वैसे ही काम करते हैं। `html`, `dirhtml` और `singlehtml` हर बिल्ड में मूल पर `<name>.html` (`html_file_suffix`) लिखते हैं। टेम्पलेट या `templates_path` की किसी फ़ाइल का बदलाव सभी पेज फिर बनाता है।

पृष्ठ कहाँ जाता है और Sphinx से कहाँ अलग है:

- समान नाम का पेज दस्तावेज़ की जगह लेता है, जैसे Sphinx में अंत में लिखा जाता है। `index = "landing.html"` से `index.html` लैंडिंग पेज होता है; विषय-सूची फिर भी `index.rst` से आती है। `singlehtml` में मूल एकल पेज रहता है, टेम्पलेट पेज नहीं लिखा जाता और चेतावनी आती है।
- `dirhtml` पेज `genindex.html` और `search.html` के पास `download.html` में लिखता है; Sphinx `download/index.html` लिखता है। `pathto("download")` उसी फ़ाइल का पथ देता है।
- नाम साइट के मूल का फ़ाइल नाम हो। लिंक मूल से बनते हैं, इसलिए `"sub/page"` चेतावनी (`build.additional_page`) के साथ अस्वीकार है। `genindex` जैसा मौजूद नाम भी अस्वीकार है; Sphinx वहाँ एक पेज पर दूसरा लिख देता है।
- `guidedog migrate conf.py` से `html_additional_pages` लाता है।

त्रुटियाँ

टेम्पलेट की गलती बिल्ड रोकती है और फ़ाइल, पंक्ति संख्या, पंक्ति तथा सुझाव दिखाती है:

```
TEMPLATE ERROR                                     template.error

_templates/base.html:1

No filter named 'default'.

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

अपरिभाषित वेरिएबल Jinja की तरह खाली दिखता है।

अनियंत्रित टेम्पलेट भी क्रैश नहीं, त्रुटि से रुकते हैं: 100 से अधिक `syntax` स्तर (कोष्ठक, टैग, ऑपरेटर, फ़िल्टर या `elif` कड़ी), 200 से अधिक मैक्रो, `include` या `recursive` लूप स्तर, अथवा 512 KiB से अधिक `recursion` स्टैक। खुद को रखने वाली सूची Python की तरह `[...]` दिखती है।

PDF टेम्पलेट

PDF पुस्तक Typst से सजती है। `_templates/book.typ` सामान्य Typst फ़ाइल है जो `book` फ़ंक्शन देती है। Guidedog पुस्तक सोर्स इस रूप में लिखता है:

```
#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
```

इसलिए किताब का रूप तय करने वाली हर चीज़ उसी फ़ाइल में है: फ्रॉन्ट, पृष्ठ और हाशिए, शीर्षक, शीर्षक पृष्ठ, पृष्ठ शीर्ष और विषय-सूची।

पैरामीटर

पैरामीटर	मान
title	pdf_documents का title; अन्यथा project।
author	किताब का author; अन्यथा author विकल्प।
version	release.
date	today : दिया गया today; अन्यथा today_fmt में बिल्ड की तारीख।
lang	language.
paper	"a4"; pdf_paper_size letter हो तो "us-letter"।
numbering	toctree में numbered हो तो true।
logo	pdf_logo का पथ; न दिया हो तो अनुपस्थित।
copyright	अंतिम प्रकाशन टिप्पणी के लिए copyright; इसे स्वीकारने वाले टेम्पलेट को ही दिया जाता है।
body	अध्यायों की सामग्री।

टेम्पलेट डिफ़ॉल्ट वाले नए पैरामीटर जोड़ सकता है। quickstart में `accent`, फ़ॉन्ट `serif`, `sans`, `mono`, पाठ size और `page-ref` हैं। अंतिम को `n => [p. #n]` जैसे फ़ंक्शन से सेट करने पर दूसरे पेज के संदर्भ के साथ पेज नंबर आता है।

मॉड्यूलर Typst टेम्पलेट

Typst में भी एक ही फ़ाइल की सीमा नहीं है। शैली, मैक्रो और आवरण को `_templates/` की अलग `.typ` फ़ाइलों में बाँटें और `#import` तथा `#include` से जोड़ें:

कोड सूची 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
}
```

टेम्पलेट क्या इस्तेमाल कर सकता है

Typst पैकेज

`#import "@preview/cetz:0.4.2"` जैसे पैकेज चलते हैं और पहली बार डाउनलोड होते हैं। केवल डिस्क के पैकेज से बनाने के लिए `pdf_packages = "offline"` सेट करें।

फ़ॉन्ट

`typst` की तरह सिस्टम फ़ॉन्ट उपलब्ध हैं। `pdf_font_paths` से फ़ोल्डर जोड़ें, या `pdf_fonts = "embedded"` से केवल Typst के अंतर्निहित फ़ॉन्ट चुनें, ताकि पुस्तक बाइट-दर-बाइट दोहराई जा सके।

प्रीएम्बल

`pdf_preamble` उस Typst फ़ाइल का नाम है जो `#show: book` के बाद आती है। अपना टेम्पलेट बनाए बिना कुछ `set` और `show` नियम देने के लिए ठीक है।

दस्तावेज़ में Typst

दस्तावेज़ `reStructuredText` और `Markdown` हैं; Typst फ़ाइलें दस्तावेज़ नहीं। पुस्तक में Typst मार्कअप देने के लिए `raw` ब्लॉक लिखें, जिसे वेब पेज छोड़ देते हैं:

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

`.. only:: pdf` भी सामान्य सामग्री केवल किताब में रखता है।

Typst की त्रुटि पर बिल्ड फ़ाइल और पंक्ति बताता है, चाहे टेम्पलेट में हो या बनी पुस्तक में। हर PDF का सोर्स `_build/pdf/sources/<pdf-filename>.typ` है; `reference-en.pdf` का `sources/reference-en.pdf.typ`। एक पुस्तक वाला बिल्ड `_build/pdf/book-<language>.typ` की जाँच प्रति भी रखता है। विफलता कुछ प्रकाशित नहीं करती; सोर्स त्रुटि में बताए `_build/.doctrees/failed` पथ पर रहता है। कई पुस्तकों के विफल सोर्स अलग हैं।

नया टेम्पलेट लिखना

`quickstart` के टेम्पलेट से शुरू करें; उनमें हर हिस्से का उपयोग है:

1. डिफ़ॉल्ट स्टाइलशीट वाला `layout.html` मार्कअप रखें या उसके साथ `_static/guidedog.css` भी बदलें।
2. दोहराए हिस्से अलग फ़ाइल में ले जाकर `include` करें, या `base.html` बनाकर `layout.html` में `extends` करें।
3. किताब के लिए `book` के नियम बदलें या उसी नाम और पैरामीटर का नया फ़ंक्शन लिखें।

संपादन में `guidedog serve` से प्रीव्यू देखें। स्रोत सहेजकर ब्राउज़र रिफ़्रेश करें; सर्वर बदले इनपुट फिर बिल्ड करके पृष्ठ भेजता है।

चित्र, वीडियो, ऑडियो और डाउनलोड

संसाधन उसे समझाने वाले दस्तावेज़ के पास रखें। सापेक्ष पथ दें या स्रोत रूट के लिए `/` से शुरू करें। `Guidedog` दस्तावेज़ों के संसाधन कॉपी करता है। वेब पर वीडियो चलता है; PDF रिकॉर्डिंग का नाम बताता है।

चित्र

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

.. figure:: /screenshots/search.png

    Searching the site.
```

पृष्ठ चित्रों को `_images` में कॉपी करते हैं; किताब समाहित करती है। PNG, JPEG, GIF, SVG और WebP दोनों में काम करते हैं। लापता चित्र चेतावनी देता है। पृष्ठ उसका वैकल्पिक पाठ रखता है; किताब चित्र छोड़ती है।

वीडियो और ऑडियो

वीडियो (.mp4, .webm, .ogv, .mov, .m4v) या ऑडियो (.mp3, .ogg, .oga, .m4a, .opus, .wav, .flac, .aac) बताने वाला image या figure उसे चलाता है, जैसा Docutils वीडियो के लिए करता है:

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

   The two-minute tour.

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

पृष्ठ ब्राउज़र प्लेयर दिखाते हैं और न चला पाने वाले ब्राउज़र के लिए फ़ाइल लिंक देते हैं। किताब मीडिया नहीं चला सकती, इसलिए प्लेयर की जगह बॉक्स में रिकॉर्डिंग बताती है। पाठ आउटपुट [video: ...] या [audio: ...] लिखता है।

डाउनलोड

download रोल किसी भी फ़ाइल से लिंक करता और उसे _downloads में कॉपी करता है:

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

थीम की फ़ाइलें

_static थीम की स्टाइलशीट, स्क्रिप्ट, फ़ॉन्ट और लोगो रखता है। ये साइट के _static में वैसे ही कॉपी होते हैं। html_static_path निर्देशिका देता है और html_css_files, html_js_files, html_logo, html_favicon उसकी फ़ाइलें बताते हैं। सामग्री का मीडिया दस्तावेज़ों के साथ रखें ताकि साथ ले जाने पर लिंक चलते रहें।

अंतरराष्ट्रीयकरण

संरचना बनाए रखकर व्याख्या का अनुवाद करें। Guidedog संदेश कैटलॉग निकालता, अनुवाद मिलाता और चुनी भाषा में बिल्ड करता है। स्रोत पाठ संदर्भ रहता है। अनुवाद न हो तो स्रोत दिखता है, नया पाठ नहीं गढ़ा जाता।

काम का क्रम

1. gettext बिल्डर से संदेश टेम्पलेट लिखें:

```
guidedog build gettext
```

हर टेक्स्ट डोमेन का एक टेम्पलेट (.pot) _build/gettext में बनता है। दस्तावेज़ का हर अनुच्छेद, शीर्षक, सूची मद, तालिका सेल, कैप्शन, सूचना, शब्द और फ़ील्ड एक संदेश बनता है। साथ में मूल फ़ाइल और पंक्ति भी दर्ज होती है।

2. sphinx-intl update की तरह हर भाषा के कैटलॉग बनाएँ या अपडेट करें:

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

कैटलॉग locales/de/LC_MESSAGES/*.po में बनते हैं। अगली बार कमांड msgmerge की तरह नए टेम्पलेट मिलाता है। अनुवाद बने रहते हैं। मूल संदेश बदलने पर पुराने अनुवाद को समीक्षा के लिए fuzzy चिह्न मिलता है। हटे हुए संदेश अंत में अप्रचलित (#~) रूप में रहते हैं, ताकि लौटने पर अनुवाद मिल सके।

3. हर `msgstr` भरें। अनुवाद सही होने पर संदेश की `#`, `fuzzy` पंक्ति हटाएँ। `fuzzy` अनुवाद इस्तेमाल नहीं होते।

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

4. `sphinx-intl stat` की तरह अनुवाद का हाल देखें:

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

5. उस भाषा में बिल्ड करें:

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

या `conf.toml` में `language = "de"` रखें।

हर पुस्तक के नाम में भाषा जुड़ती है: `manual.pdf` से `manual-de.pdf` बनता है। एक बिल्ड पूरा आउटपुट प्रकाशित करता है; कई संस्करण रखने के लिए अलग लक्ष्य फ़ोल्डर चुनें। जैसे जर्मन के लिए `guidedog build -b pdf docs build/books/de -D language=de` और अंग्रेज़ी के लिए `guidedog build -b pdf docs build/books/en -D language=en`।

अन्य इनपुट की तरह केवल बदले कैटलॉग वाले दस्तावेज़ फिर पढ़ते हैं। कैटलॉग जोड़ने या हटाने पर सभी फिर पढ़ते हैं।

`guidedog intl` में `sphinx-intl` के विकल्प हैं: टेम्पलेट के लिए `-p DIR` (डिफ़ॉल्ट `_build/gettext`), भाषा के लिए बार-बार दिया जा सकने वाला `-l LANG` (डिफ़ॉल्ट प्रोजेक्ट की भाषा), कैटलॉग के लिए `-d DIR` (डिफ़ॉल्ट `locale_dirs` की पहली प्रविष्टि), पंक्ति चौड़ाई के लिए `-w N` (डिफ़ॉल्ट 76), और `--no-obsolete`।

अनुवाद लिखना

संदेश पैराग्राफ़, शीर्षक या अन्य तत्व का स्रोत पाठ है, मार्कअप सहित। `.rst` में `reStructuredText` और `.md` में `Markdown` है। अनुवाद उसी मार्कअप में पढ़ता है, इसलिए मूल की तरह ज़ोर, लिंक और संदर्भ दे सकता है।

संदर्भ समान रहने चाहिए। हर संदर्भ का वही रूप रखें; केवल आसपास का पाठ और उसका शीर्षक बदलें:

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

अनुवाद से संदर्भ हटे तो बिल्ड Sphinx की तरह चेतावनी (`inconsistent references in translated message`) देता है और अनुवाद इस्तेमाल करता है। मूल में न मौजूद संदर्भ जोड़ने पर वह हल नहीं हो सकता; तब चेतावनी देकर मूल पाठ रखा जाता है। रीडर अनुवाद का मार्कअप अस्वीकार करे तो भी मूल रहता है और चेतावनी में अनुवाद दिखता है।

खंड एंकर भाषा से नहीं बदलते। अनूदित शीर्षक मूल एंकर रखता है, इसलिए भीतर और बाहर के लिंक चलते रहते हैं।

सेटिंग्स

मानक विकल्प Sphinx के नाम और डिफ़ॉल्ट रखते हैं। अनूदित न होने वाले संदर्भ परिशिष्ट के लिए Guidedog केवल निष्कर्षण पर लागू बहिष्करण भी देता है।

विकल्प	अर्थ
<code>language</code>	बिल्ड की भाषा, जैसे <code>"de"</code> या <code>"pt_BR"</code> ।

विकल्प	अर्थ
<code>locale_dirs</code>	सोर्स फ़ोल्डर के सापेक्ष कैटलॉग की जगह। हर जगह <code>LANGUAGE/LC_MESSAGES/DOMAIN.po</code> (या संकलित <code>.mo</code>) होता है। डिफ़ॉल्ट <code>["locales"]</code> है। कई जगह हों तो संदेश का अनुवाद देने वाली पहली जगह चुनी जाती है।
<code>gettext_compact</code>	तय करता है कि दस्तावेज़ के संदेश किस कैटलॉग में जाएँ। <code>true</code> (डिफ़ॉल्ट): शीर्ष स्तर के हर दस्तावेज़ का अपना कैटलॉग; एक फ़ोल्डर के दस्तावेज़ साझा करते हैं (<code>guide/usage.rst</code> का <code>guide.po</code>)। <code>false</code> : हर दस्तावेज़ का अलग कैटलॉग। कोई नाम दें तो सभी दस्तावेज़ उसी नाम का कैटलॉग साझा करते हैं।
<code>gettext_location</code>	संदेश का स्थान (<code>#: ../../index.rst:12</code>) लिखें। डिफ़ॉल्ट <code>true</code> ।
<code>gettext_uuid</code>	हर संदेश के हर स्थान का पहचानकर्ता लिखें। <code>Guidedog</code> दस्तावेज़, संदेश और स्थान से इसे बनाता है, इसलिए हर बिल्ड में समान है।
<code>gettext_auto_build</code>	हर <code>.po</code> को साथ के <code>.mo</code> में कम्पाइल करें, दूसरे टूल के लिए। डिफ़ॉल्ट <code>true</code> है। <code>Guidedog</code> खुद <code>.po</code> पढ़ता है।
<code>gettext_additional_targets</code>	और अनुवाद सामग्री: <code>index</code> (इंडेक्स प्रविष्टियाँ), <code>literal-block</code> , <code>doctest-block</code> , <code>raw</code> और <code>image</code> (वैकल्पिक पाठ)।
<code>gettext_allow_fuzzy_translations</code>	<code>gettext</code> अनुवाद भी इस्तेमाल करें। डिफ़ॉल्ट <code>false</code> ।
<code>gettext_exclude_patterns</code>	केवल POT निष्कर्षण से हटने वाले दस्तावेज़ पथ पैटर्न, जैसे <code>["api/**"]</code> । डिफ़ॉल्ट <code>[]</code> । दस्तावेज़ HTML और PDF में बने रहते हैं।
<code>gettext_last_translator</code> , <code>gettext_language_team</code>	टेम्पलेट हेडर के <code>Last-Translator</code> और <code>Language-Team</code> ।
<code>figure_language_filename</code>	बिल्ड की भाषा में चित्र की फ़ाइल। डिफ़ॉल्ट <code>"{root}.{language}{ext}"</code> है। <code>language = "de"</code> हो तो <code>img/logo.png</code> की जगह <code>img/logo.de.png</code> लिया जाता है, यदि वह मौजूद हो। फ़ील्ड हैं <code>{root}</code> , <code>{path}</code> , <code>{basename}</code> , <code>{ext}</code> , <code>{docpath}</code> और <code>{language}</code> ।
<code>translation_progress_class</code>	अनुदित तत्वों को <code>translated</code> और बाकी को <code>untranslated</code> क्लास दें (<code>true</code>)। केवल एक भी चुन सकते हैं (" <code>translated</code> " या " <code>untranslated</code> "), ताकि स्टाइलशीट बचा हुआ काम दिखा सके।

Guidedog के इंटरफ़ेस के शब्द

`Guidedog` कुछ पाठ खुद बनाता है: सूचना के शीर्षक, "Added in version 2.1", "Fig. 1", "Contents", पेज के "Previous", "Next", "Navigation", "Search", खोज और सूचकांक के पेज, और पुस्तक का "Version"। ये `guidedog` टेक्स्ट डोमेन से इस क्रम में खोजे जाते हैं:

1. परियोजना की `locales/LANGUAGE/LC_MESSAGES/guidedog.po`।
2. परियोजना की `locales/LANGUAGE/LC_MESSAGES/sphinx.po`, जैसे `Sphinx` पढ़ता है।
3. `Guidedog` का बिल्ट-इन कैटलॉग: जर्मन, स्पेनिश, फ्रेंच, जापानी, ब्राज़ीलियाई पुर्तगाली, रूसी और सरलीकृत चीनी के लिए।

अंग्रेज़ी शब्द संदेश पहचानकर्ता हैं। `Sphinx` में वही शब्द हो तो उसका पहचानकर्ता भी वही है, इसलिए प्रोजेक्ट का `sphinx.po` चलता रहता है। अपना `guidedog.po` किसी शब्द को बदल सकता है या नई भाषा जोड़ सकता है।

टेम्पलेट `Sphinx` की तरह `_()`, `gettext()` और `ngettext()` से अनुवाद करते हैं:

```
<small>{{ _('Previous') }}</small>
```

```
{{ _('Last updated on %s.')|format(last_updated) }}
```

conf.toml में न दिए हों तो numfig_format, html_title और html_short_title भाषा के अनुसार बदलते हैं।

तारीख

|today|, पुस्तक की तारीख और पेज का last_updated Sphinx के नियम से बनते हैं। today_fmt और html_last_updated_fmt में strftime प्रारूप होता है; महीने और दिन के नाम language से आते हैं। %b %d, %Y अंग्रेज़ी में "Sep 29, 2026" और जर्मन में "Sept. 29, 2026" है। खाली प्रारूप _('%b %d, %Y') है, जिसे कैटलॉग अनुवाद कर सकता है। ऊपर की भाषाओं के नाम Unicode CLDR से आते हैं। बाकी में अंग्रेज़ी इस्तेमाल होती है, जैसे Babel को भाषा न पता होने पर Sphinx करता है।

तारीख स्थानीय समय क्षेत्र की होती है। SOURCE_DATE_EPOCH (1970 से बीते सेकंड, UTC में) सेट करने पर हर मशीन पर समान तारीख के साथ पुनरुत्पाद्य बिल्ड बनता है।

अन्य परियोजनाओं से लिंक

sphinx.ext.intersphinx से यहाँ न सुलझे संदर्भ दूसरे प्रोजेक्ट की इन्वेंटरी में खोजे जाते हैं। यह हर Sphinx या Guidedog साइट की objects.inv है। परियोजनाएँ conf.toml में दें:

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

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

हर प्रविष्टि दूसरे प्रोजेक्ट के पृष्ठों का पता और इन्वेंटरी की जगह देती है। यह URL, स्रोत फ़ोल्डर से सापेक्ष फ़ाइल या उस पते की objects.inv के लिए "" है। सूची क्रम से आजमाई जाती है। यह Sphinx का intersphinx_mapping है, Python के None की जगह "" के साथ। guidedog migrate इसे बदलता है।

फिर संदर्भों में नया कुछ नहीं चाहिए:

```
: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.
```

लिंक दूसरे प्रोजेक्ट के लेबल और दस्तावेज़ के शीर्षक दिखाते हैं और “(in Python v3.14)” जैसा टूलटिप रखते हैं। Sphinx की तरह :doc: संदर्भ को परियोजना का नाम (:doc:`python:tutorial/index`) चाहिए, क्योंकि intersphinx_disabled_reftypes का डिफ़ॉल्ट ["std:doc"] है।

बिल्ड शुरू होने पर इन्वेंटरी लाकर _build/.doctrees/_intersphinx_cache में intersphinx_cache_limit दिनों तक रखी जाती है: डिफ़ॉल्ट 5, ऋणात्मक मान पर बनी रहती है। स्थानीय फ़ाइल हर बिल्ड में पढ़ती है। Sphinx की तरह रीडायरेक्ट के बाद नई निर्देशिका लिंक का आधार बनती है और कैश उसे याद रखता है, ताकि कैश से वही लिंक बने। जैसे https://werkzeug.palletsprojects.com/objects.inv का रीडायरेक्ट /en/stable/ पर है। intersphinx_timeout हर डाउनलोड का समय सेकंड में सीमित करता है। न पहुँच सकने वाला प्रोजेक्ट चेतावनी देता है और उसके संदर्भ अनसुलझे रहते हैं।

Guidedog सभी पृष्ठ, लेबल, शब्द और ऑब्जेक्ट सहित अपनी objects.inv लिखता है, जिससे अन्य प्रोजेक्ट भी इसी तरह यहाँ लिंक कर सकते हैं।

ऑब्जेक्ट टाइप घोषित करना

परियोजना को ऐसे नाम चाहिए हो सकते हैं जिन्हें मानक डोमेन नहीं जानते। TOML में उनके ऑब्जेक्ट टाइप घोषित करें। Guidedog Python एक्सटेंशन लोड किए बिना लक्ष्य, इंडेक्स प्रविष्टि और रोल बना सकता है। घोषणा सिंटैक्स बताती है, चलने वाला व्यवहार नहीं।

लक्ष्य: `add_crossref_type`

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

तब `.. setting:: DEBUG` `setting-DEBUG` ID वाला लक्ष्य और इंडेक्स प्रविष्टि बनाता है। `:setting: `DEBUG`` उसे लिंक करता है, Sphinx की तरह। `index` Sphinx का `indextemplate` है: पहले कोलन से पहले प्रकार (`single`, `pair`, `triple`, `see`, `seealso`), बाद में प्रविष्टि, और `%s` नाम है।

विवरण: `add_object_type`

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

`.. django-admin:: migrate [app_label]` कमांड को ऑब्जेक्ट की तरह सिग्नेचर, सामग्री, ID (`django-admin-migrate`) और इंडेक्स प्रविष्टि के साथ बताता है। Sphinx में `parse_node` फ़ंक्शन सिग्नेचर पढ़ सकता है; उसका सामान्य काम यहाँ फ़ील्ड से तय होता है:

name

"whole" (डिफ़ॉल्ट): पूरा सिग्नेचर ऑब्जेक्ट का नाम है। "first-word": पहले खाली स्थान तक का भाग नाम है। जैसे `pdb` का `b(reak) [lineno]` नाम `b(reak)` रखता है।

display

सिग्नेचर कैसे दिखे; `%s` उसका स्थान लेता है। डिफ़ॉल्ट में लिखा हुआ रूप।

program

`true`: नाम आगे बताए विकल्पों का प्रोग्राम बनता है, जैसा `.. program::` करता है।

निर्देश के अन्य नाम

एक्सटेंशन Sphinx के निर्देश को दूसरे नाम से पंजीकृत कर सकता है:

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

डोमेन के बिना लक्ष्य मानक या डिफ़ॉल्ट डोमेन का निर्देश है। `py:function` जैसा लक्ष्य अपना डोमेन बताता है।

माइग्रेशन

`guidedog migrate conf.py` में सूचीबद्ध परियोजना के एक्सटेंशन मॉड्यूल ढूँढता है। उनके `add_crossref_type`, `add_object_type` और `add_directive` कॉल पढ़कर, बिना चलाए, ये

तालिकाएँ लिखता है। जो घोषित नहीं कर सकता वह बताता है: `parse_node` फ़ंक्शन (`name`, `display`, `program` से मिलाएँ), और Python के निर्देश, रोल तथा इवेंट हैंडलर।

autodoc से Python का दस्तावेज़ीकरण

Guidedog सोर्स कोड पढ़कर Python का दस्तावेज़ीकरण करता है। वह पैकेज को `import` नहीं करता। `docstring` और `signature` दस्तावेज़ के ग्राफ़ में शामिल होते हैं। केवल रनटाइम पर बनने वाले ऑब्जेक्ट का विवरण अलग से लिखना पड़ता है। यह सीमा जानबूझकर रखी गई है।

सेटअप

एक्सटेंशन बताएँ और Guidedog को सोर्स फ़ोल्डर के सापेक्ष पैकेजों का स्थान दें, जैसे `sys.path` Python को बताता है:

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

`guidedog migrate` आपके लिए `autodoc_source_paths` लिखता है। इसके लिए वह `conf.py` में पैकेज तक पहुँचने वाली पंक्तियाँ पढ़ता है, जैसे `sys.path.insert(0, os.path.abspath('../'))`। कहीं और इंस्टॉल किए गए पैकेज के लिए उसका फ़ोल्डर जोड़ना होगा, मसलन किसी dependency का सोर्स checkout। Guidedog कभी Python के अपने `site-packages` में नहीं खोजता।

इसके बाद directives Sphinx की तरह काम करते हैं:

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

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

Sphinx 9.1 के सभी विकल्प उसी अर्थ में स्वीकार किए जाते हैं: `: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:` और `autodoc_default_options` को रद्द करने वाले `no-` रूप।

सेटिंग्स

इन `conf.toml` सेटिंग्स के नाम, मान और डिफ़ॉल्ट Sphinx जैसे ही हैं।

`autodoc_source_paths`

यह Guidedog का अपना विकल्प है। Python मॉड्यूल खोजने वाली निर्देशिकाएँ खोज के क्रम में दें। पथ स्रोत निर्देशिका के सापेक्ष हैं। केवल इन निर्देशिकाओं की फ़ाइलें पढ़ी जाती हैं। बाहर जाने वाले प्रतीकात्मक लिंक स्वीकार नहीं होते।

`autoclass_content`

क्लास का वर्णन किस `docstring` से हो: `"class"` (डिफ़ॉल्ट), `"init"` या `"both"`।

`autodoc_class_signature`

`"mixed"` (डिफ़ॉल्ट) या `"separated"`। दूसरा विकल्प `__init__` को एक मेथड के रूप में दिखाता है।

autodoc_default_options

हर निर्देश को मिलने वाले विकल्पों की तालिका, जैसे `members = true` या `member-order = "bysource" | false` देने पर वह विकल्प छोड़ दिया जाता है।

autodoc_docstring_signature

`true` (डिफ़ॉल्ट): docstring की पहली पंक्ति `name(args) -> result` जैसी हो तो उसे सिग्नेचर माना जाता है।

autodoc_inherit_docstrings

`true` (डिफ़ॉल्ट): अपना docstring न होने पर सदस्य बेस क्लास का docstring लेता है।

autodoc_member_order

"alphabetical" (डिफ़ॉल्ट), "groupwise" या "bysource" |

autodoc_preserve_defaults

`false` (डिफ़ॉल्ट): डिफ़ॉल्ट मान `repr()` के रूप में दिखते हैं। `true` देने पर स्रोत में लिखा रूप दिखता है।

autodoc_typehints

"signature" (डिफ़ॉल्ट), "description" (:type: और :rtype: फ़ील्ड में), "both" या "none" |

autodoc_typehints_description_target

"all" (डिफ़ॉल्ट), "documented" या "documented_params" |

autodoc_typehints_format

"short" (डिफ़ॉल्ट) या "fully-qualified" |

autodoc_use_type_comments

`true` (डिफ़ॉल्ट): `# type: टिप्पणियाँ टाइप एनोटेशन मानी जाती हैं।`

strip_signature_backslash

Sphinx की तरह सिग्नेचर में बैकस्लैश दोहरे किए जाते हैं।

autodoc_mock_imports, autodoc_type_aliases, autodoc_warningiserror

ये विकल्प स्वीकार होते हैं। कोई मॉड्यूल इम्पोर्ट नहीं किया जाता, इसलिए mock की ज़रूरत नहीं होती। टाइप एलियस लागू नहीं होते।

Guidedog Python के व्यवहार का अनुमान कैसे लगाता है

नाम Python के बाइंडिंग नियमों से खोजे जाते हैं। पैकेज के `__init__.py` में `from .app import Flask` लिखने से `flask.Flask` उस क्लास को दर्शाता है जो `flask/app.py` में परिभाषित है। autodoc इसे `:canonical: flask.app.Flask` के साथ दिखाता है। बेस क्लास भी इसी तरह खोजी जाती हैं। मेथड रिज़ॉल्यूशन क्रम Python के नियमों से निकलता है और विरासत में मिले सदस्य बेस क्लास के स्रोत से आते हैं। केवल `if TYPE_CHECKING:` के भीतर इम्पोर्ट किए गए नाम रनटाइम पर मौजूद नहीं होते। इसलिए उन नामों वाले एनोटेशन, Sphinx की तरह, लिखे हुए रूप में रहते हैं।

docstring वही स्ट्रिंग होती है जिसे Python संग्रहीत करता। एस्केप प्रोसेस होते हैं और इंडेंटेशन Python 3.13 कम्पाइलर के नियमों से हटाता है। एट्रिब्यूट का वर्णन Sphinx के विश्लेषक वाली जगहों से मिलता है: असाइनमेंट के बाद या ठीक ऊपर की `#:` टिप्पणी और उसके ठीक बाद का स्ट्रिंग लिटरल। सिग्नेचर `def` से मिलता है। क्लास के लिए Python के खोज क्रम में मेटाक्लास का `__call__`, `__new__` या `__init__` लिया जाता है। `dataclass` और `NamedTuple` के लिए फ़ील्ड काम आते हैं। `@overload` के रूप कार्यान्वयन के सिग्नेचर की जगह लेते हैं।

Python चलाए बिना क्या नहीं जाना जा सकता

जब स्रोत से Python का व्यवहार पता नहीं चल सकता, Guidedog संबंधित ऑब्जेक्ट का नाम लेकर बताता है। दस्तावेज़ बनाना संभव न हो तो चेतावनी देता है। Sphinx से कम जानकारी के साथ दस्तावेज़ बने तो एक सूचना देता है, जो `-v` से दिखाई देती है।

- कम्पाइल किए गए मॉड्यूल (`.so`, `.pyd`) का स्रोत उपलब्ध नहीं होता, इसलिए उसके ऑब्जेक्ट नहीं मिलते।
- कोड चलाकर बने मान (`app = Flask(__name__)`, `now = datetime.now()`) के लिए `:value:` नहीं मिलता। गणना से मिलने वाला डिफ़ॉल्ट स्रोत के लिखे रूप में दिखता है।
- स्रोत पथों के बाहर के डेकोरेटर (`@click.command()`) से बनी फ़ंक्शन का वर्णन मूल फ़ंक्शन के आधार पर होता है।
- स्रोत पथों के बाहर के पैकेज की क्लास का नाम उसके इम्पोर्ट पथ से दिया जाता है। यह पथ उसे परिभाषित करने वाले मॉड्यूल से अलग हो सकता है। उसके सदस्य, कन्स्ट्रक्टर का सिग्नेचर और आगे विरासत में मिलने वाले docstring अज्ञात रहते हैं।
- बिल्ट-इन टाइप के docstring उपलब्ध नहीं होते। इसलिए अपना docstring न रखने वाला मेथड, जिसे `dict` से मिलना होता, छोड़ दिया जाता है।
- autodoc के इवेंट (`autodoc-process-docstring`, `autodoc-skip-member` आदि) संभालने वाले Python एक्सटेंशन नहीं चलते।
- सामान्य कोड से बहुत अधिक नेस्टिंग वाले मॉड्यूल का विश्लेषण नहीं होता और `autodoc.too_deep` चेतावनी मिलती है। Python खुद 200 स्तर के नेस्टेड ब्रैकेट अस्वीकार करता है। Guidedog कुछ सौ ऑपरैंड वाली + श्रृंखला भी अस्वीकार करता है।
- बढ़ते जाने वाले काम की सीमाएँ तय हैं। इम्पोर्ट और एलियस से नाम खोजते समय एक पथ पर 48 कदम और कुल 100,000 कदम तक चलते हैं। टाइप एलियस या स्थिरांक 10,000 कदम तक खोले जाते हैं; उसके बाद लिखे हुए रूप में दिखते हैं। क्लास का MRO 100 क्लास तक खोजा जाता है। एक निर्देश अधिकतम 10,000 निर्देश बनाता है; सीमा पार करने पर `autodoc.limit` चेतावनी मिलती है। आखिरी सीमा तक केवल वह क्लास पहुँचती है जो अपने नाम से खुद को समाहित करती है।

अनुपलब्ध पैकेजों की स्रोत निर्देशिकाएँ `autodoc_source_paths` में जोड़ने से इनमें से अधिकतर सीमाएँ दूर हो जाती हैं।

दोबारा बिल्ड करना

autodoc निर्देश की पढ़ी हुई हर Python फ़ाइल पृष्ठ की निर्भरता बनती है। उसे बदलने पर अगली बिल्ड में पृष्ठ फिर पढ़ा जाता है। जिस पृष्ठ का ऑब्जेक्ट नहीं मिला, वह हर बिल्ड में फिर पढ़ा जाता है। स्रोत में ऑब्जेक्ट आते ही वह मिल जाता है।

सुरक्षा

दस्तावेज़ बनाते समय परियोजना का कोड कभी नहीं चलता। Guidedog Python स्रोत को पाठ की तरह पढ़ता है और `include` जैसी पथ सीमा लागू करता है: केवल `autodoc_source_paths` के भीतर, बाहर जाने वाले लिंक से नहीं।

Odin के दस्तावेज़ बनाना

Odin डोमेन पैकेज, प्रोसीजर, प्रकार और सदस्य को नाम वाले लक्ष्य देता है। Guidedog Odin सोर्स पार्स करके API पेज भी बनाता है। वह कंपाइलर या पैकेज नहीं चलाता। बने सिग्नेचर के साथ भी उपयोगी अनुबंध चाहिए।

ऑब्जेक्ट का वर्णन

Python मॉड्यूल की तरह पैकेज आगे के विवरण का संदर्भ है:

```
.. 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
```

सिग्नेचर Odin घोषणाओं की तरह एक पंक्ति में लिखे और वैसे ही दिखाए जाते हैं। नीचे के उदाहरण ऐसे दिखते हैं:

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

```
Circle = 1
```

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

एक वर्ग को मापता है।

पैरामीटर: `side` – इसकी भुजा।

रिटर्न:

- `a` – इसका क्षेत्रफल।
- `ok` – इसका प्रकार ज्ञात है या नहीं।

निर्देश और उनके स्वीकार किए सिग्नेचर:

odin:package और odin:currentpackage

`core:fmt` या प्रोजेक्ट के मूल के नीचे `readers/sphinx` जैसा पथ दें। ऑब्जेक्ट का नाम पैकेज इंपोर्ट पथ से है: `core:fmt.println`। `odin:currentpackage` बिना लक्ष्य बनाए संदर्भ बदलता है; `None` उसे हटाता है। विकल्प: `:synopsis:`, `:platform:`, `:deprecated:`, `:no-index:` और `:imports:`। अंतिम में सिग्नेचर के इंपोर्ट उपनाम `alias=package` जोड़ें (`gd=core rst=readers/rst`) में हैं; `gd.Node_Id` तब `core.Node_Id` से लिंक होता है। `:private:` निजी प्रकार और प्रोसीजर समूह के निजी सदस्य (`State Rules catalog_text`) देता है। वर्णन न होने से सिग्नेचर इन्हें पाठ दिखाता है। बने पेज इन्हें स्वतः सूचीबद्ध करते हैं।

odin:procedure (या odin:proc)

`name :: proc(params) -> results` में नाम से पहले गुण (`@(require_results)`), `proc` से पहले `#force_inline`, कॉल नियम (`proc "c" (...)`), डिफ़ॉल्ट तर्क (`x := 1`, `x: int = 1`), `$T` बहुरूपी तर्क, `..` परिवर्तनीय तर्क, `using`, `#c_vararg`, नाम वाले परिणाम, टैग (`#optional_ok`) और `where` खंड स्वीकार हैं। `name(params) -> results` इसका छोटा रूप है।

odin:procgroup

name :: proc{a, b}: हर सदस्य अपने प्रोसीजर से लिंक होता है।

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

Name :: struct(\$T: typeid) #packed, Name :: union #no_nil {A, B},
 Name :: enum u8, Name :: bit_set[Flag; u8], Name :: bit_field u32 |
 odin:type में distinct प्रकार और उपनाम (Handle :: distinct uintptr,
 Callback :: proc(x: int) -> bool) हैं। प्रकार की सामग्री में सदस्य आते हैं।

odin:field और odin:enumerator

प्रकार के भीतर name: Type के साथ टैग (name: string `json:"n"`) या बिट आकार
 (low: u8 | 3), और Name या Name = 3 हैं। इनके नाम Type.member होते हैं।

odin:const, odin:var, odin:foreign

NAME :: 64 या NAME : int : 64; name: Type, name := value या
 name: Type = value; विदेशी इंपोर्ट के लिए libc "system:c"।

हर ऑब्जेक्ट Sphinx के :no-index:, :no-index-entry:, :no-contents-entry: और :no-typesetting: लेता है। अपने विकल्प हैं :package: (दूसरे पैकेज में वर्णन), :private: और :deprecated: message (सिग्नेचर में न हों तो @(private) और @(deprecated="message")), :availability: Windows, Linux (पहली पंक्ति में उपलब्धता), :foreign: libc (foreign ब्लॉक का प्रोसीजर)। कई सिग्नेचर पंक्तियाँ अलग लक्ष्यों पर एक ऑब्जेक्ट दिखाती हैं; ID और सूचकांक केवल पहली को मिलते हैं।

प्रकार की जगह के नाम पहले सिग्नेचर के पैकेज और प्रकार में, फिर लिखे रूप में खोजकर लिंक होते हैं। अंतर्निहित प्रकार (int, string, rawptr, typeid, any आदि), कुंजीशब्द, बहुरूपी नाम (\$T और आगे T, बाहरी प्रकार सहित), ऐरे लंबाई, डिफ़ॉल्ट, स्थिरांक, टैग और where खंड केवल लिखे रूप में दिखते हैं, लिंक नहीं।

रोल और ID

:odin:pkg:, :odin:proc:, :odin:type: (struct, union, enum, bit set, bit field आदि), :odin:const:, :odin:var:, :odin:field:, :odin:enumerator: और :odin:obj: (कोई भी) लिंक देते हैं। Python की तरह ~ अंतिम हिस्सा दिखाता है (:odin:proc: `~core:fmt.println` में println); प्रारंभिक . किसी ऑब्जेक्ट नाम का अंत खोजता है। खोज संदर्भ के प्रकार और पैकेज, फिर मूल नाम, फिर पथ के : या / के बाद होती है। fmt.println को core:fmt.println और sphinx.Config को readers/sphinx.Config मिलता है, जैसे Odin अंतिम पथ नाम लेता है। :imports: उपनाम पहले देखा जाता है। default-domain:: odin या primary_domain = "odin" में odin: छोड़ सकते हैं।

ID Sphinx का make_id है, odin- और पूरे नाम पर: odin-core-fmt.println, odin-readers-sphinx.Config.docname। Python ID की तरह केस और बिंदु रहते हैं; पथ का : और / हाइफ़न बनते हैं। ID पढ़ने योग्य, स्थिर और समान अंतिम नाम के पैकेज में अलग हैं। पैकेज ID odin-package- तथा पथ है। सूचकांक "println (procedure in core:fmt)" जैसा है। objects.inv सभी ऑब्जेक्ट odin डोमेन और प्रकार (odin:procedure, odin:struct, odin:field आदि) में रखता है; Sphinx मॉड्यूल की तरह पैकेज पहले हैं।

सामग्री के सूचना फ़िल्ड दूसरे डोमेन की तरह समूहित होते हैं: :param name: (:type name: सहित), नाम वाले परिणाम का :result name:, :returns: और :rtype:।

स्रोत से बनाना

सोर्स फ़ोल्डर के सापेक्ष पैकेज की जगह बताएँ:


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

`odin_autoapi_dirs` के नीचे पैकेज सापेक्ष पथ से नाम पाता है (`src/shapes/round` का `shapes/round`)। संग्रह में Odin इंपोर्ट नाम (`core:strings`) है। निर्माण निर्देश `autodoc` की तरह चलते हैं:

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

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

`odin:autopackage` पैकेज और टिप्पणी लिखता है; `:members:` पर सभी या चुनी घोषणाएँ भी। `odin:autoproc`, `odin:autoprocgroup`, `odin:autotype`, `odin:autoconst`, `odin:autovar` वर्तमान पैकेज नाम या `package.name` की एक घोषणा देते हैं। विकल्प: `:members:`, `:undoc-members:`, `:private-members:` (सामान्यतः हटे `@(private)`), `:exclude-members:`, `:member-order:` (डिफ़ॉल्ट `source` में फ़ाइल नाम फिर स्थान; `alphabetical`; `groupwise` के शीर्षक "Types", "Procedures", "Procedure groups", "Constants", "Variables", "Foreign imports") और `:no-index:`।

दस्तावेज़ टिप्पणियाँ घोषणा के ठीक ऊपर `//` पंक्तियाँ या `/* */` ब्लॉक, तथा फ़िल्ड या `enum` सदस्य के अंत की टिप्पणी हैं। ये सादा पाठ हैं, जो इस तरह `reStructuredText` बनाता है:

- अनुच्छेद अनुच्छेद रहते हैं; `- item` पंक्तियाँ सूची रहती हैं।
- खाली पंक्ति या कोलन वाली पंक्ति (`Example:`) के बाद टैब या स्पेस से इंडेंट ब्लॉक `code-block:: odin` है; `Output:` के बाद `text` है। अनुच्छेद पंक्ति के ठीक बाद के पाठ से अधिक इंडेंट पंक्ति उसी को आगे बढ़ाती है।
- `Inputs:` के बाद `- name: text` से `:param name:` फ़िल्ड और `Returns:` से `:result name:` बनते हैं; नाम न हो तो `:returns:`। यह Odin कोर लाइब्रेरी का रूप है।
- `NOTE:` या `WARNING:` से शुरू पंक्ति नोट या चेतावनी बनती है।
- ``code`` लिटरल पाठ है। `reStructuredText` के मार्कअप वाले दूसरे अक्षर (`*`, `|`, शब्द के अंत का `_` आदि) एस्केप होते हैं, इसलिए टिप्पणी जैसी लिखी है वैसी दिखती है और चेतावनी नहीं आती।

API पृष्ठ

`odin_autoapi_dirs` देने पर हर पैकेज का पेज पढ़ने के दौरान बनता है, `sphinx-autoapi` की तरह। `api/shapes` में `odin_autoapi_options` के साथ `odin:autopackage:: shapes` और `api/index` में पैकेज व सारांश हैं। पेज प्रोजेक्ट दस्तावेज़ हैं: `toctree`, खोज, सामान्य सूचकांक, `singlehtml` और `PDF` में आते हैं, पर सोर्स फ़ोल्डर में नहीं लिखे जाते। सेटिंग:

odin_autoapi_dirs

पेज पाने वाले पैकेज के फ़ोल्डर, सोर्स के सापेक्ष। केवल इनके और `odin_collections` के नीचे की फ़ाइलें पढ़ी जाती हैं। बाहर जाता लिंक अस्वीकार है।

odin_collections

`core:strings` जैसे निर्देश तर्कों के लिए संग्रह नाम और फ़ोल्डर की तालिका। इन पैकेजों के पेज नहीं बनते।

odin_autoapi_root

पेज का फ़ोल्डर; डिफ़ॉल्ट "api"। स्रोत में समान नाम का दस्तावेज़ हो तो वही रहता है और चेतावनी मिलती है।

odin_autoapi_options

डिफ़ॉल्ट ["members", "undoc-members"]; private-members जोड़ा जा सकता है।

odin_autoapi_member_order

"source" (डिफ़ॉल्ट), "alphabetical" या "groupwise"।

odin_autoapi_add_tocentry

true (डिफ़ॉल्ट): api/index मूल दस्तावेज़ के पहले toctree में जुड़ता है।

odin_autoapi_generate_api_docs

true (डिफ़ॉल्ट); false में फ़ोल्डर केवल निर्देशों के लिए पढ़े जाते हैं।

जिस पैकेज का दस्तावेज़ नहीं, उसके प्रकार (core:io.Writer) लिंक नहीं होते। -n में nitpick_ignore_regex = ["odin:type", "(core|base):.*"] से चेतावनी दबाएँ। Guidedog का docs/api भी ऐसे बनता है।

प्लैटफ़ॉर्म

पैकेज के Odin लक्ष्यों का दस्तावेज़ उसी के फ़ाइल नियमों से बनता है। *_windows.odin, *_linux_arm64.odin, *_amd64.odin केवल संबंधित लक्ष्य हैं। #+build linux, darwin, #+build !windows और पुराने //+build दायरा घटाते हैं। फ़ाइल स्तर का when ODIN_OS == .Windows केवल ODIN_OS व ODIN_ARCH तुलना करे तो शाखा उसके लक्ष्य और else बाकी हैं। *_test.odin और #+ignore हटते हैं। सब लक्ष्यों की घोषणा एक बार; कुछ की "Availability: Windows" सहित। अलग सिग्नेचर अलग लक्ष्य सहित दिखते हैं; ID पहले का है। सीमित लक्ष्य वाला पैकेज :platform: बताता है।

दोबारा बिल्ड करना

पेज द्वारा पढ़ी हर Odin फ़ाइल उसकी निर्भरता है। सोर्स बदलने पर अगली बिल्ड केवल उस पैकेज के पेज फिर पढ़ती है। पैकेज जुड़ने, हटने या सारांश बदलने पर सूचकांक फिर पढ़ा जाता है। guidedog serve HTML देने से पहले odin_autoapi_dirs और odin_collections का बदलाव समय देखता है। संपादन के बाद ब्राउज़र रिफ़्रेश करें।

सीमाएँ

- सोर्स केवल पार्स होता है, संकलन या प्रकार जाँच नहीं। प्रकार और स्थिरांक जैसे लिखे हैं वैसे दिखते हैं; 100 अक्षर से बड़ा मान हटता है। अनुमान नहीं होता। A :: B का प्रकार या स्थिरांक नाम से तय है: बड़े अक्षर से शुरू लेकिन पूरी तरह बड़े अक्षर नहीं, या अंतर्निहित प्रकार, तो प्रकार है।
- when में ODIN_OS और ODIN_ARCH की तुलना के अलावा शर्तें नहीं जाँची जातीं; दोनों शाखाएँ लिखी जाती हैं।
- निजी प्रकार का नाम पाठ दिखता है क्योंकि :private-members: के बिना उसकी प्रविष्टि नहीं। बना पैकेज उन्हें :private: में रखता है, इसलिए -n रिपोर्ट नहीं करता।
- core:odin/parser दोनों #optional_ok और where वाला प्रोसीजर अस्वीकार करता है। फ़ाइल की सूचना दी जाती है और पढ़ा जा सका हिस्सा दस्तावेज़ बनता है।
- गहरे कोड में Odin पार्सर बहुत स्टैक लेता है; कुछ सौ कोष्ठक श्रेड स्टैक भर सकते हैं। वास्तविक कोड से कहीं गहरी फ़ाइल (लगभग 50 कोष्ठक या प्रकार स्तर, या एक कड़ी में करीब हज़ार else) odin.autodoc.nesting चेतावनी से हटती है। शेष पैकेज लिखा जाता है। when के &&, ||, ! 64 स्तर तक जाँचे जाते हैं; अधिक पर बिना मूल्यांकन माना जाता है।

व्यावहारिक काम

हर पृष्ठ एक ठोस काम से शुरू होता है। कमांड और उसका परिणाम बताता है।

Guidedog बिल्ड और इंस्टॉल करना

macOS पर Homebrew से स्थापित करें

Apple silicon या Intel वाले Mac पर आधिकारिक `tap insanai/homebrew-guidedog` का उपयोग करें:

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

Homebrew आपके प्रोसेसर के लिए सही रिलीज़ पैकेज चुनता है और उसका SHA-256 जाँचता है। पैकेज में Typst, Graphviz और tree-sitter शामिल हैं। HTML और PDF बनाने के लिए Odin कंपाइलर, Python वातावरण या अलग से Typst स्थापित करने की ज़रूरत नहीं है। इस रिलीज़ के लिए macOS 12 या उससे नया संस्करण चाहिए।

अपना पहला प्रोजेक्ट आपकी पहली परियोजना के अनुसार बनाएँ। स्थापित रिलीज़ को अपडेट करने के लिए:

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

स्थापित पैकेज की जाँच के लिए `brew test insanai/guidedog/guidedog` चलाएँ। यह एक प्रोजेक्ट बनाकर उसका HTML और PDF तैयार करता है।

स्रोत से बिल्ड करें

`tools/odin-release.txt` में तय किया गया Odin टूलचेन इस्तेमाल करें। रिलीज़ में Odin dev-2026-10 और Typst 0.15.1 का उपयोग होता है। साझा लाइब्रेरी वाले बिल्ड के लिए Graphviz की डेवलपमेंट लाइब्रेरी चाहिए।

डेवलपमेंट बिल्ड

रिपॉज़िटरी के रूट से:

```
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 बिल्ड सोर्स-आधारित autodoc के लिए निश्चित संस्करण का Python व्याकरण देता है। Graphviz की विकास फ़ाइलें सिस्टम पैकेज मैनेजर से स्थापित करें। Typst ब्रिज के बिना बिल्ड PDF के लिए स्थापित typst चलाता है। उसे स्पष्ट चुनने के लिए `--typst=/absolute/path/to/typst` दें।

एम्बेडेड कम्पाइलर

नेटिव निर्भरताएँ एक बार बिल्ड करें, फिर Odin प्रोग्राम से लिंक करें:

```
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_TYPOST_BRIDGE=true -define:GRAPHDOG_STATIC=true
```

Typst ब्रिज Rust में है और Guidedog की आवंटन-मुक्त सीमा के बाहर है। एम्बेडेड बिल्ड अलग Typst प्रोग्राम चलाने की ज़रूरत हटाता है, लेकिन कंपाइलर का अपना मेमोरी खर्च नहीं।

Windows पर आउटपुट का नाम `build/guidedog.exe` रखें। बिना पथ दिए `guidedog` चलाने के लिए प्रोग्राम को `PATH` में रखें।

दस्तावेज़ बिल्ड और प्रकाशित करना

परियोजना की सामान्य आउटपुट निर्देशिका में बिल्ड करें:

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

साइट `docs/manual/_build/html` में है। PDF और बना Typst सोर्स `docs/manual/_build/pdf` में हैं। वेब सर्वर HTML फ़ोल्डर को स्थिर फ़ाइलों की तरह दे सकता है। उत्पादन में Odin सर्वर आवश्यक नहीं।

आउटपुट निर्देशिका चुनना

Sphinx जैसा रूप स्रोत और गंतव्य स्पष्ट देता है:

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

आउटपुट सोर्स के बाहर रखें। प्रकाशन से सोर्स ट्री बदलने वाली संरचना Guidedog अस्वीकार करता है। कस्टम लक्ष्य `build/.manual-site.guidedog/.doctrees` जैसे स्थिर, समान स्तर के केश का उपयोग करता है। इसलिए प्रकाशन फ़ोल्डर में केवल प्रकाशित सामग्री रहती है।

प्रीव्यू और जाँच

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

dummy बिल्डर प्रोजेक्ट पढ़कर संबंध हल करता है, प्रकाशन नहीं बनाता। अर्थ की त्वरित जाँच के लिए इस्तेमाल करें। अंतिम लेआउट इससे नहीं जाँचता।

दोहराने योग्य समय

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

इससे बिल्ड का समय तय होता है। पुनरुत्पाद्यता सोर्स, टेम्पलेट, फ्रॉन्ट, कंपाइलर संस्करण और संसाधनों पर भी निर्भर है। केवल समय तय करना काफी नहीं।

बिल्ड सफल होने पर पूरा फ़ोल्डर प्रकाशित करें। सफल निकास आवश्यक है; टेम्पलेट बदलने पर दृश्य जाँच भी चाहिए।

Sphinx परियोजना का माइग्रेशन

Guidedog Sphinx की प्रोजेक्ट संरचना और कई सेटिंग नाम अपनाता है। वह `conf.py` या Python एक्सटेंशन नहीं चलाता। माइग्रेशन की शुरुआत डेटा को चलने वाले कॉन्फिगरेशन से अलग करने से होती है।

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

`migrate` लिटरल सेटिंग पढ़कर `conf.toml` लिखता है। गणना से आए मानों में जहाँ मानव निर्णय चाहिए, वह बताता है। उन्हें जानने के लिए प्रोजेक्ट इंपोर्ट या Python नहीं चलाता। समान मानने से पहले बना कॉन्फिगरेशन जाँचें।

तीन चरणों में काम करें

1. दस्तावेज़ ग्राफ़ सही करें। रूट, प्रत्यय, बहिष्करण और `include` जाँचें।
2. संदर्भ सही करें। डोमेन, इन्वेंटरी और कस्टम ऑब्जेक्ट टाइप जाँचें।
3. प्रस्तुति सही करें। टेम्पलेट, आरेख, गणित और दोनों आउटपुट जाँचें।

अज्ञात निर्देश प्रोजेक्ट के निजी एक्सटेंशन का हो सकता है। HTML बिल्ड की सफलता उसके अर्थ के पुनरुत्पादन का प्रमाण नहीं। निदान पढ़ें। फिर निर्देश बदलें, समर्थित उपनाम दें, या सीमा दर्ज करके बनाए रखें।

समर्थित सुविधाएँ और CPython, Django, Flask के प्रमाण Sphinx के साथ संगतता में हैं। घोषणात्मक कस्टम ऑब्जेक्ट के लिए ऑब्जेक्ट टाइप घोषित करना देखें।

कोडबेस के दस्तावेज़ बनाना

सार्वजनिक नाम के साथ अनुबंध चाहिए। इनपुट, परिणाम, स्वामित्व और विफलता की शर्तें समझाएँ। केवल सिग्नेचर से मैनुअल नहीं बनता।

Odin

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

Guidedog Odin के पार्सर से सोर्स पढ़ता है। वह संबंधित पैकेज को संकलित या चलाता नहीं। बने हुए पेज दस्तावेज़ ग्राफ़ और ऑब्जेक्ट सूची में जुड़ते हैं। इस रिपॉज़िटरी का अलग API प्रोजेक्ट `docs/api` है। निर्देश, सिग्नेचर और विकल्प के लिए Odin के दस्तावेज़ बनाना देखें।

Python

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

Guidedog Python सोर्स को पार्स करता है, इंपोर्ट नहीं। इससे इंपोर्ट के दुष्प्रभाव बचते हैं। रनटाइम पर बने ऑब्जेक्ट छूट सकते हैं। सार्वजनिक इंटरफ़ेस का हिस्सा हों तो उन्हें स्पष्ट रूप से लिखें। खोज के सटीक नियम `autodoc` से Python का दस्तावेज़ीकरण में हैं।

परियोजनाओं के बीच लिंक

बना हुआ `objects.inv` साइट के साथ प्रकाशित करें। दूसरे प्रोजेक्ट का intersphinx मैपिंग उससे नाम हल कर सकता है। पुनरुत्पाद्य बिल्ड के लिए स्पष्ट मैपिंग दें। अन्य परियोजनाओं से लिंक देखें।

इन दस्तावेज़ों का रखरखाव

Guidedog के तीन प्रोजेक्ट हैं: `docs/manual`, `docs/api` और `docs/gds`। मैनुअल और GDS `guidedog quickstart` से बने। बने हुए टेम्पलेट सीधे संपादित करें। डिज़ाइन रिकॉर्ड मैनुअल की विषय-सूची में न रखें।

मैनुअल बिल्ड करना

रिपॉज़िटरी के रूट से:

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

मैनुअल की विषय-सूची में पूरा Odin API संदर्भ है। PDF कमांड व्याख्या और घोषणाओं को लिंक करती एक पुस्तक बनाता है। मैनुअल और अलग API प्रोजेक्ट दोनों `lib/` से संदर्भ बनाते हैं। API की नकल की हुई पांडुलिपि अलग से सँभालनी नहीं पड़ती।

केवल API परियोजना बिल्ड करने के लिए:

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

चर्चा दस्तावेज़ बिल्ड और जाँचना

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

GDS CLI मेटाडेटा, संख्या, स्थिति और रिकवरी सँभालता है। RST प्रकाशन प्रोजेक्ट इंजन बुलाकर पूरी HTML साइट और हर रिकॉर्ड का अलग PDF बनाता है। चयन केवल PDF सीमित करता है; HTML में पूरा कैटलॉग रहता है। चुने हुए PDF का प्रीव्यू `_build/selected/<artifact>/pdf` में होता है और `_build/pdf` का पूरा कैटलॉग सुरक्षित रहता है।

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

सामान्य प्रोजेक्ट कमांड भी चलते हैं। GDS का अपना `conf.toml` और `_build` फ़ोल्डर है। लेखन प्रक्रिया `docs/gds/workflow.rst` में पढ़ें।

प्रकाशन से पहले

लिंक, निदान, चौड़ी तालिकाएँ और कोड ब्लॉक जाँचें। टाइपोग्राफी बदलने पर PDF पेज रेंडर करके देखें। HTML को डेस्कटॉप और फ़ोन की चौड़ाई, हल्के और गहरे थीम में जाँचें। गणित और आरेख बिना नेटवर्क भी चलने चाहिए। इंजन बदलने पर संबंधित लाइब्रेरी और एकीकरण टेस्ट चलाएँ।

संदर्भ

ये पृष्ठ अनुबंध बताते हैं और खोजकर देखने के लिए हैं। पहला उदाहरण चाहिए तो ट्यूटोरियल से शुरू करें।

कमांड लाइन संदर्भ

guidedog की कमांड लाइन से दस्तावेज़ परियोजनाओं का प्रबंधन, बिल्ड, पूर्वावलोकन, अनुवाद और रूपांतरण किया जा सकता है।

सामान्य सिंटैक्स

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

निकास कोड

तालिका 6 निकास स्थितियाँ

निकास कोड	स्थिति
0	सफलता: सभी काम पूरे हुए और कोई त्रुटि अनसुलझी नहीं रही।
1	बिल्ड या रूपांतरण में त्रुटि हुई, या -w ने चेतावनी को त्रुटि माना।
2	कमांड लाइन का सिंटैक्स, फ़्लैग या विकल्प गलत है।
3	तय संसाधन सीमा पूरी हुई या मेमोरी उपलब्ध नहीं हुई।
4	फ़ाइल संबंधी काम या बाहरी कंपाइलर विफल हुआ।
5	आंतरिक त्रुटि; निदान में रिपोर्ट करने का तरीका दिया जाता है।
70	क्रैश हुआ; पहले प्रकाशित आउटपुट उपलब्ध रहता है।

कमांड का सार

तालिका 7 CLI कमांड

कमांड	उद्देश्य
<code>quickstart [DIR]</code>	नई दस्तावेज़ परियोजना बनाता है, जिसमें टेम्पलेट और कॉन्फिगरेशन सीधे संपादित किए जा सकते हैं।
<code>build [BUILDER] [DIR]</code>	दस्तावेज़ परियोजना को चुने गए आउटपुट प्रारूप में बनाता है।
<code>serve [DIR]</code>	स्थानीय HTTP पूर्वावलोकन सर्वर शुरू करता है और फ़ाइल बदलने पर अपने आप दोबारा बिल्ड करता है।
<code>clean [DIR]</code>	बिल्ड से बनी फ़ाइलें और कैश हटाता है।
<code>convert INPUT</code>	एक फ़ाइल या मानक इनपुट को सीधे रूपांतरित करता है।
<code>migrate [conf.py]</code>	Sphinx की <code>conf.py</code> को घोषणात्मक <code>conf.toml</code> में बदलता है।
<code>intl <update stat></code>	PO अनुवाद कैटलॉग अद्यतन करता है या अनुवाद की प्रगति बताता है।
<code>formats</code>	अंतर्निहित रीडर, रेंडर और प्रारूप-अनुपालन की जानकारी दिखाता है।
<code>gds <COMMAND></code>	Guidedog Discussions के डिज़ाइन अभिलेख और उनका जीवनचक्र संभालता है।

guidedog quickstart

परियोजना फ़ोल्डर में `conf.toml`, `index.rst`, `_templates/` में टेम्पलेट और `_static/` में स्टाइल बनाता है:

```
# 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
```

विकल्प:

- `-q`: बिना सवाल पूछे कमांड लाइन पर दिए गए मान इस्तेमाल करता है।
- `-p, --project NAME`: परियोजना का प्रदर्शित नाम तय करता है।
- `-a, --author NAME`: लेखक या संगठन का नाम तय करता है।
- `-v VERSION`: छोटा संस्करण नंबर तय करता है, जैसे `1.0`।
- `-r RELEASE`: पूरा रिलीज़ पहचानकर्ता तय करता है, जैसे `1.0.0-rc1`।
- `-l LANGUAGE`: परियोजना की डिफ़ॉल्ट भाषा तय करता है। डिफ़ॉल्ट मान `en` है।
- `--sep`: स्रोत को मुख्य फ़ोल्डर में रखने के बजाय अलग `source/` और `build/` फ़ोल्डर बनाता है।
- `--suffix EXT`: स्रोत दस्तावेज़ का डिफ़ॉल्ट एक्सटेंशन तय करता है। डिफ़ॉल्ट `.rst` है।

guidedog build

स्रोत पढ़ने, विषय-सूची और पार-संदर्भ सुलझाने, तथा अंतिम आउटपुट प्रकाशित करने का काम संभालता है:

```
# 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
```

समर्थित बिल्डर:

- `html`: खोज और नेविगेशन वाला पूरा HTML साइट बनाता है।
- `dirhtml`: `dir/index.html` जैसे फ़ोल्डर-आधारित HTML URL बनाता है।
- `singlehtml`: सभी दस्तावेज़ एक HTML पृष्ठ में जोड़ता है।
- `pdf`: Typst से PDF पुस्तक बनाता है। इसके उपनाम `latex` और `latexpdf` हैं।
- `text`: हर दस्तावेज़ के लिए सादा टेक्स्ट फ़ाइल बनाता है।
- `gettext`: अनुवाद योग्य पाठ को GNU gettext POT टेम्पलेट में निकालता है।
- `dummy`: अंतिम फ़ाइल बनाए बिना दस्तावेज़ ट्री पार्स करता है और संदर्भ सुलझाता है; सिंटैक्स जाँच के लिए उपयोगी है।

बिल्ड के विकल्प:

- `-a`: संशोधन समय की परवाह किए बिना सभी आउटपुट फ़ाइलें लिखता है।
- `-E`: कैश की स्थिति अनदेखी कर वातावरण नए सिरे से बनाता है।
- `-W`: हर चेतावनी को बिल्ड की विफलता मानता है।
- `--keep-going`: `-W` के साथ भी बाकी दस्तावेज़ों का काम पूरा कर, निकलने से पहले सभी त्रुटियाँ बताता है।
- `-n`: सभी अनसुलझे पार-संदर्भों पर चेतावनी देता है।
- `-j N / -j auto`: दस्तावेज़ पार्स करने के लिए थ्रेड की संख्या तय करता है। `auto` इसे CPU कोर की संख्या के बराबर रखता है।
- `-D name=value`: इस रन के लिए `conf.toml` की सेटिंग बदलता है, जैसे `-D language=de`।
- `-D table.key=value`: `html_context` जैसी टेबल सेटिंग की एक कुंजी सेट करता है और बाक़ी कुंजियाँ वैसी ही रखता है, जैसे `sphinx-build` करता है।
- `-t TAG: only` निर्देश के लिए शर्त वाला टैग तय करता है।
- `-c DIR`: `conf.toml` वाला फ़ोल्डर बताता है।
- `-C`: कोई `conf.toml` फ़ाइल पढ़े बिना बिल्ड करता है।
- `-v`: विस्तृत मोड में अतिरिक्त जानकारी और समय के आँकड़े दिखाता है।
- `-q`: केवल चेतावनी और त्रुटि दिखाता है।
- `--color=always|never|auto`: टर्मिनल में ANSI रंगों का इस्तेमाल नियंत्रित करता है।
- `--diagnostics=json`: मानक त्रुटि आउटपुट पर मशीन-पठनीय JSON निदान लिखता है।
- `--budget=MIB`: परियोजना होस्ट की मेमोरी सीमा तय करता है। डिफ़ॉल्ट 1024 MiB है।
- `--memory=ram|disk`: मेमोरी बजट खत्म होने पर अपनाई जाने वाली नीति तय करता है।
- `--untrusted`: रॉ एम्बेडेड मार्कअप छोड़ता है और नेटवर्क से लाना बंद करता है। यह क्षमता की नीति है, प्रक्रिया का सैंडबॉक्स नहीं।

guidedog serve

स्थानीय HTTP सर्वर और फ़ाइल निगरानी शुरू करता है। स्रोत, टेम्पलेट या संसाधन बदलने पर दोबारा बिल्ड करता है और ब्राउज़र ताज़ा करता है:

```
# 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
```

विकल्प:

- `-p, --port PORT`: स्थानीय पोर्ट। डिफ़ॉल्ट 8000 है।
- `--host HOST`: सर्वर का IP पता। डिफ़ॉल्ट 127.0.0.1 है।

guidedog clean

बिल्ड फ़ोल्डर और वातावरण का कैश हटाता है:

```
guidedog clean docs
```

guidedog convert

परियोजना फ़ोल्डर या कॉन्फिगरेशन के बिना फ़ाइल या मानक इनपुट को सीधे बदलता है:

```
# 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 -
```

विकल्प:

- `-o, --output FILE`: आउटपुट फ़ाइल का पथ; `-` का मतलब मानक आउटपुट है।
- `--to FORMAT`: आउटपुट प्रारूप `html`, `pdf` या `text` है; फ़ाइल नाम से भी पहचाना जा सकता है।
- `--from FORMAT`: स्रोत रीडर `rst`, `commonmark`, `myst` या `typst` है; इनपुट के एक्सटेंशन से भी पहचाना जा सकता है।
- `--force`: पूछे बिना मौजूदा आउटपुट फ़ाइल बदल देता है।
- `--strict`: किसी भी चेतावनी पर विफल होता है।
- `--raw=allow|omit`: एम्बेड किए गए रॉ कोड की नीति; डिफ़ॉल्ट `omit` है।
- `--emit-typst FILE`: बीच में बना Typst स्रोत अलग फ़ाइल में लिखता है।

guidedog migrate

Sphinx की Python कॉन्फिगरेशन फ़ाइल `conf.py` को घोषणात्मक `conf.toml` में बदलता है:

```
# 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
```

विकल्प:

- `-o FILE`: बताए गए पथ पर लिखता है; `-` से मानक आउटपुट पर लिखता है।
- `--force`: मौजूदा `conf.toml` को बदल देता है।

guidedog intl

बहुभाषी दस्तावेज़ों के अनुवाद कैटलॉग संभालता है:

```
# 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` के विकल्प:

- `-l, --language LANG`: लक्षित भाषा का कोड; कई भाषाओं के लिए दोहरा सकते हैं।
- `-p, --pot-dir DIR`: निकाले गए POT टेम्पलेट का पथ बताता है।

guidedog formats

अंतर्निहित रीडर, रेंडरर, सिंटैक्स हाइलाइटर और मानक परीक्षणों के नतीजे दिखाता है:

```
guidedog formats
```

guidedog gds

Guidedog Discussions के आर्किटेक्चर प्रस्ताव (RFC) संभालता है:

```
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
```

कॉन्फिगरेशन संदर्भ

हर Guidedog परियोजना की सेटिंग मुख्य फ़ोल्डर की `conf.toml` में रहती है। Guidedog इसे घोषणात्मक TOML डेटा की तरह पढ़ता है; मनमाना कोड नहीं चलाता।

कई सेटिंग कुंजियाँ Sphinx की `conf.py` के चर से मेल खाती हैं, इसलिए `guidedog migrate conf.py` से माइग्रेशन किया जा सकता है।

न्यूनतम कॉन्फिगरेशन

परियोजना के लिए मूल जानकारी ही काफी है। छोड़ी गई सेटिंग के लिए डिफ़ॉल्ट मान लागू होते हैं:

```
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"
```

परियोजना का मेटाडेटा

तालिका 8 मेटाडेटा के विकल्प

कुंजी	प्रकार	डिफ़ॉल्ट	विवरण
<code>project</code>	स्ट्रिंग	<code>"Python"</code>	पाठक को दिखाया जाने वाला परियोजना का नाम।
<code>author</code>	स्ट्रिंग	<code>" "</code>	लेखक या संगठन का नाम।
<code>copyright</code>	स्ट्रिंग	<code>" "</code>	पृष्ठ के नीचे दिखाया जाने वाला कॉपीराइट विवरण।
<code>version</code>	स्ट्रिंग	<code>" "</code>	छोटा संस्करण नंबर, जैसे <code>"1.0"</code> ।
<code>release</code>	स्ट्रिंग	<code>" "</code>	पूरा संस्करण नंबर, जिसमें alpha या beta पहचान भी हो सकती है, जैसे <code>"1.0.0b1"</code> ।
<code>language</code>	स्ट्रिंग	<code>"en"</code>	टाइपसेटिंग, शब्द-विभाजन और अनुवाद के लिए ISO भाषा कोड, जैसे <code>"de"</code> या <code>"zh_CN"</code> ।
<code>today</code>	स्ट्रिंग	<code>" "</code>	अपनी तारीख का पाठ। इसे छोड़ने पर वर्तमान तारीख <code>today_fmt</code> से स्वरूपित होती है।
<code>today_fmt</code>	स्ट्रिंग	<code>" "</code>	दस्तावेज़ की तारीख के लिए फ़ॉर्मेट स्ट्रिंग।

सामान्य विकल्प

तालिका 9 खोज और पार्सिंग के विकल्प

कुंजी	प्रकार	डिफ़ॉल्ट	विवरण
root_doc	स्ट्रिंग	"index"	विषय-सूची का मूल दस्तावेज़।
source_suffix	ऐरे	[".rst"]	दस्तावेज़ स्रोत के रूप में पहचाने जाने वाले एक्सटेंशन।
exclude_patterns	ऐरे	[]	स्रोत खोज में छोड़ने वाले फ़ोल्डर और फ़ाइलों के glob पैटर्न।
include_roots	ऐरे	[]	स्रोत शामिल करने के लिए अनुमत बाहरी मूल फ़ोल्डर।
templates_path	ऐरे	[]	HTML और PDF टेम्पलेट के फ़ोल्डर; अंतर्निहित टेम्पलेट से पहले खोजे जाते हैं।
extensions	ऐरे	[]	सक्रिय करने वाले समर्थित अंतर्निहित Sphinx एक्सटेंशन, जैसे "sphinx.ext.autodoc"।
primary_domain	स्ट्रिंग	"py"	बिना उपसर्ग के निर्देश और रोल का डिफ़ॉल्ट डोमेन, जैसे "py", "c" या "odin"।
highlight_language	स्ट्रिंग	"default"	कोड ब्लॉक में भाषा न लिखने पर इस्तेमाल होने वाली भाषा।
pygments_style	स्ट्रिंग	""	हल्की थीम के लिए Pygments हाइलाइटिंग स्टाइल।
pygments_dark_style	स्ट्रिंग	""	गहरी थीम के लिए सिंटेक्स हाइलाइटिंग स्टाइल।
smartquotes	बूलियन	true	सीधे उद्धरण चिह्न और डैश को टाइपोग्राफ़िक चिह्नों में बदलता है।
rst_prolog	स्ट्रिंग	""	हर दस्तावेज़ के शुरू में जोड़ा जाने वाला reStructuredText अंश।
rst_epilog	स्ट्रिंग	""	हर दस्तावेज़ के अंत में जोड़ा जाने वाला reStructuredText अंश।

क्रमांक और गणित

तालिका 10 क्रमांक के विकल्प

कुंजी	प्रकार	डिफ़ॉल्ट	विवरण
<code>numfig</code>	बूलियन	false	सक्रिय करने पर चित्र, तालिका और कोड को अपने आप क्रमांक देता है।
<code>numfig_secnum_depth</code>	पूर्णक	1	चित्र के क्रमांक में शीर्षक की गहराई; जैसे 1 से Fig. 2.1 बनता है।
<code>numfig_format</code>	टेबल	नीचे देखें	"figure", "table" और "code-block" के क्रमांक उपसर्ग का प्रारूप।
<code>math_number_all</code>	बूलियन	false	हर अलग पंक्ति वाले समीकरण को अपने आप क्रमांक देता है।
<code>math_eqref_format</code>	स्ट्रिंग	"({number})"	:eq: से दिए गए समीकरण संदर्भ का फ़ॉर्मेट।
<code>mathjax_path</code>	स्ट्रिंग	URL	MathJax JavaScript फ़ाइल का CDN पता या स्थानीय पथ।

`numfig_format` के डिफ़ॉल्ट मान:

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

निदान और सख़्त जाँच

तालिका 11 निदान के विकल्प

कुंजी	प्रकार	डिफ़ॉल्ट	विवरण
<code>nitpicky</code>	बूलियन	false	अनसुलझे पार-संदर्भ और टूटे लक्ष्य पर चेतावनी देता है।
<code>nitpick_ignore</code>	ऐरे	[]	सख़्त संदर्भ जाँच से छूट पाने वाले [type, target] जोड़ों की सूची।
<code>suppress_warnings</code>	ऐरे	[]	न दिखाए जाने वाले चेतावनी वर्गों के कोड।
<code>keep_warnings</code>	बूलियन	false	प्रकाशित दस्तावेज़ में चेतावनियाँ शामिल करता है।

HTML आउटपुट के विकल्प

तालिका 12 HTML के विकल्प

कुंजी	प्रकार	डिफ़ॉल्ट	विवरण
html_theme	स्ट्रिंग	"guidedog"	HTML बनाने के लिए इस्तेमाल होने वाली थीम।
html_theme_options	टेबल	{}	HTML थीम को दिए जाने वाले कुंजी-मान विकल्प।
html_title	स्ट्रिंग	अपने आप तय होता है	ब्राउज़र टैब में दिखने वाला शीर्षक।
html_short_title	स्ट्रिंग	अपने आप तय होता है	नेविगेशन पथ में दिखने वाला छोटा शीर्षक।
html_logo	स्ट्रिंग	""	स्रोत फ़ोल्डर के सापेक्ष परियोजना के लोगो का पथ।
html_favicon	स्ट्रिंग	""	फ़ेविकॉन फ़ाइल का पथ।
html_static_path	ऐरे	[]	आउटपुट के <code>_static/</code> में कॉपी किए जाने वाले फ़ोल्डर।
html_extra_path	ऐरे	[]	बिना बदलाव आउटपुट के मूल फ़ोल्डर में कॉपी किए जाने वाले फ़ोल्डर।
html_css_files	ऐरे	[]	HTML पृष्ठों में लोड की जाने वाली अपनी CSS फ़ाइलों के नाम।
html_js_files	ऐरे	[]	HTML पृष्ठों में लोड की जाने वाली अपनी JavaScript फ़ाइलों के नाम।
html_permaLinks	बूलियन	true	अनुच्छेद और खंडों के लिए स्थायी लिंक जोड़ता है।
html_permaLinks_icon	स्ट्रिंग	"¶"	स्थायी लिंक के लिए इस्तेमाल होने वाला चिह्न या पाठ।
html_baseurl	स्ट्रिंग	""	साइटमैप और मेटाडेटा के लिए मानक आधार URL।
html_context	टेबल	{}	Jinja टेम्पलेट को दिए जाने वाले अपने चर का शब्दकोश।
html_additional_pages	टेबल	{}	बनाए जाने वाले अतिरिक्त पृष्ठ: <pre>{ "page_name" = "template.html" }</pre>

PDF और Typst के विकल्प

तालिका 13 PDF के विकल्प

कुंजी	प्रकार	डिफ़ॉल्ट	विवरण
pdf_paper_size	स्ट्रिंग	"a4"	कागज़ का आकार: "a4" या "us-letter"।
pdf_logo	स्ट्रिंग	" "	पुस्तक के आवरण पर लगाने वाले लोगो का पथ।
pdf_toplevel_sectioning	स्ट्रिंग	"chapter"	पुस्तक के मुख्य विभाजन का स्तर: "chapter" या "part"।
pdf_show_urls	स्ट्रिंग	"no"	मुद्रित पाठ में URL दिखाने की नीति: "no", "inline" या "footnote"।
pdf_preamble	स्ट्रिंग	" "	बने दस्तावेज़ की शुरुआत में जोड़ा जाने वाला रॉ Typst स्रोत।
pdf_font_paths	ऐरे	[]	अपने OTF/TTF फ़ॉन्ट खोजने के अतिरिक्त फ़ोल्डर।
pdf_packages	स्ट्रिंग	"download"	पैकेज खोजने की नीति: "download" या "offline"।
typst	स्ट्रिंग	"typst"	सिस्टम के Typst CLI का नाम या पूरा पथ।

PDF पुस्तक तय करना

[[pdf_documents]] टेबल में परियोजना से बनने वाली एक या अधिक पुस्तकें तय की जाती हैं:

```
[[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 के विकल्प

तालिका 14 MyST Markdown के विकल्प

कुंजी	प्रकार	डिफ़ॉल्ट	विवरण
<code>myst_enable_extensions</code>	ऐरे	<code>["dollarmath"]</code>	सिंटैक्स एक्सटेंशन: <code>"colon_fence"</code> , <code>"deflist"</code> , <code>"dollarmath"</code> , <code>"fieldlist"</code> , <code>"tasklist"</code> , <code>"substitution"</code> ।
<code>myst_heading_anchors</code>	पूर्णांक	<code>0</code>	शीर्षक से एंकर बनाने की गहराई; <code>0</code> से यह बंद होता है।
<code>myst_substitutions</code>	टेबल	<code>{}</code>	Markdown में उपलब्ध चर के प्रतिस्थापन: <code>{key = "value"}</code> ।
<code>myst_url_schemes</code>	ऐरे	<code>["http", ...]</code>	बाहरी लिंक माने जाने वाले URI स्कीम।
<code>myst_commonmark_only</code>	बूलियन	<code>false</code>	MyST एक्सटेंशन के बिना केवल CommonMark सिंटैक्स पार्स करता है।

Intersphinx के विकल्प

```

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", ""]]

```

पूरी प्रक्रिया के लिए अन्य परियोजनाओं से लिंक देखें।

Python autodoc के विकल्प

तालिका 15 Autodoc के विकल्प

कुंजी	प्रकार	डिफ़ॉल्ट	विवरण
<code>autodoc_source_paths</code>	ऐरे	<code>[]</code>	स्थिर विश्लेषण में Python मॉड्यूल खोजने के फ़ोल्डर।
<code>autoclass_content</code>	स्ट्रिंग	<code>"class"</code>	क्लास के docstring का स्रोत: <code>"class"</code> , <code>"init"</code> या <code>"both"</code> ।
<code>autodoc_member_order</code>	स्ट्रिंग	<code>"alphabetical"</code>	सदस्यों का क्रम: <code>"alphabetical"</code> , <code>"bysource"</code> या <code>"groupwise"</code> ।
<code>autodoc_typehints</code>	स्ट्रिंग	<code>"signature"</code>	टाइप संकेत दिखाने की जगह: <code>"signature"</code> , <code>"description"</code> या <code>"none"</code> ।
<code>autodoc_default_options</code>	टेबल	<code>{}</code>	सभी <code>auto*</code> निर्देशों पर लागू डिफ़ॉल्ट विकल्प।

खोज के पूरे नियमों के लिए autodoc से Python का दस्तावेज़ीकरण देखें।

Odin API दस्तावेज़ के विकल्प

तालिका 16 Odin डोमेन के विकल्प

कुंजी	प्रकार	डिफ़ॉल्ट	विवरण
<code>odin_autoapi_dirs</code>	ऐरे	<code>[]</code>	Odin पैकेज खोजने के स्रोत फ़ोल्डर।
<code>odin_autoapi_root</code>	स्ट्रिंग	<code>"api"</code>	बने Odin API दस्तावेज़ों का आउटपुट उपफ़ोल्डर।
<code>odin_autoapi_options</code>	ऐरे	<code>["members", "undoc-members"]</code>	शामिल किए जाने वाले सदस्यों का फ़िल्टर।
<code>odin_autoapi_member_order</code>	स्ट्रिंग	<code>"source"</code>	सदस्यों का क्रम: <code>"source"</code> या <code>"alphabetical"</code> ।

Odin डोमेन के पूरे उपयोग के लिए [Odin के दस्तावेज़ बनाना](#) देखें।

अंतर्राष्ट्रीयकरण के विकल्प

तालिका 17 i18n के विकल्प

कुंजी	प्रकार	डिफ़ॉल्ट	विवरण
<code>locale_dirs</code>	ऐरे	<code>["locales"]</code>	gettext अनुवाद कैटलॉग खोजने के फ़ोल्डर।
<code>gettext_compact</code>	बूलियन	<code>true</code>	एक फ़ोल्डर के दस्तावेज़ों के संदेश एक कैटलॉग में जोड़ता है।
<code>gettext_uuid</code>	बूलियन	<code>false</code>	POT संदेशों में स्थिर UUID लिखता है।
<code>gettext_auto_build</code>	बूलियन	<code>true</code>	बिल्ड के दौरान <code>.po</code> को बाइनरी <code>.mo</code> कैटलॉग में बदलता है।
<code>figure_language_filename</code>	स्ट्रिंग	<code>"{root}."{language}{ext}"</code>	भाषा के अनुसार चित्र चुनने के लिए फ़ाइल नाम का टेम्पलेट।

अनुवाद के निर्देशों के लिए [अंतरराष्ट्रीयकरण](#) देखें।

रीडर, रेंडरर और बिल्डर

रीडर सोर्स भाषा समझता है। रेंडरर एक दस्तावेज़ वृक्ष लिखता है। बिल्डर प्रोजेक्ट का संयोजन और प्रकाशन करता है। ये अलग ज़िम्मेदारियाँ हैं।

तालिका 18 स्रोत के रास्ते

इनपुट	रीडर	रास्ता
परियोजना में RST	RST और बिल्ट-इन Sphinx परत	परियोजना ग्राफ़ से HTML, Typst/PDF, पाठ या कैटलॉग बनते हैं।
परियोजना में Markdown	MyST-संगत Markdown परत	वही परियोजना ग्राफ़ और बिल्डर।
<code>convert file.md</code>	डिफ़ॉल्ट रूप में CommonMark	एक दस्तावेज़ से HTML या Typst/PDF।
<code>convert file.typ</code>	नेटिव Typst अडैप्टर	Guidedoc के सिमैटिक वृक्ष से बाहर Typst कम्पाइल करना।

बिल्डर

html

हर दस्तावेज़ की HTML फ़ाइल, इंडेक्स, खोज और इन्वेंटरी सहित।

dirhtml

निर्देशिका जैसे दस्तावेज़ URL।

singlehtml

दस्तावेज़ खंड और स्थानीय संदर्भ वाली एक मुख्य HTML पृष्ठ।

pdf

परियोजना में तय Typst किताबें। LaTeX बिल्डर के एलियस यही रास्ता चुनते हैं।

text

दस्तावेज़ों की सादे पाठ की फ़ाइलें।

gettext

अनुवाद के संदेश टेम्पलेट।

dummy

आर्टिफ़ैक्ट प्रकाशित किए बिना परियोजना पढ़ता और संदर्भ हल करता है।

कवरेज एक प्रमाण है

`guidedog formats` प्रोग्राम के वास्तविक अडैप्टर बताता है। संगतता पेज निश्चित टेस्ट और वास्तविक प्रोजेक्ट जाँच दर्ज करता है। CommonMark पास होना पूरा MyST समर्थन नहीं; RST पढ़ना हर Python एक्सटेंशन का व्यवहार नहीं। [Sphinx के साथ संगतता देखें](#)।

त्रुटि संदेश

रिपोर्ट समस्या का नाम, स्थान और उपयोगी अगला कदम बताए। Guidedog यह जानकारी संरचित डेटा में रखता है। टर्मिनल रंग और स्रोत चिह्न जोड़ता है। JSON बिना टर्मिनल सजावट के डेटा रखता है।

रिपोर्ट पढ़ना

```

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"`.

रिपोर्ट के चार भाग हमेशा इसी क्रम में होते हैं:

1. **शीर्षक पट्टी।** शीर्षक समस्या का प्रकार बड़े अक्षरों में बताता है। चेतावनी और सूचना WARNING: और NOTE: से शुरू होती हैं। दाईं ओर `rst.substitution.undefined` जैसा रिपोर्ट कोड है: खोजने के लिए स्थिर नाम।
2. **स्थान।** फ़ाइल और पंक्ति, मूल पंक्ति और समस्या वाले पाठ के नीचे चिह्न दिखता है। किसी एक पंक्ति से न जुड़ी समस्या, जैसे लापता फ़ोल्डर या फ़्लैग, में केवल पथ दिखता है या स्थान नहीं दिखता।
3. **क्या हुआ।** पहले कारण की व्याख्या होती है। ज़रूरत हो तो आउटपुट पर असर बताया जाता है: यहाँ कुछ भी नहीं लिखा गया।
4. **क्या करें।** रंगीन टर्मिनल में हरे रंग का संकेत ठोस दिशा देता है। कई उपाय हों तो सबसे उपयोगी लगने वाला पहले आता है।

रंग केवल शब्दों की बात पर ज़ोर देते हैं। `--color=never` रंग बंद करता है। `--color=always` गैर-टर्मिनल आउटपुट में भी रंग रखता है।

हर संकेत आगे की राह बताता है

Guidedog अपने संदेशों पर भी यह नियम लागू करता है। संकेत पहले क्रिया ("Add", "Pass", "Set", "Rename", "Run") और उसका लक्ष्य बताता है, जैसे `--budget=MIB, conf.toml` का `root_doc` या निर्देश का विकल्प। `tests/hints_test.odin` हर संभव संदेश पढ़ता है। खाली संकेत, संदेश की पुनरावृत्ति या बिना ज़रूरी विवरण के केवल "report this" कहने पर टेस्ट विफल होता है। दो अन्य रूप स्वीकार हैं: सुधार सुझाना ("Did you mean numfig?") और कार्रवाई-रहित नोट में "Nothing to do:" के साथ कारण देना।

समस्या Guidedog की हो तो संकेत साफ़ बताता है। रिपोर्ट के साथ चलाया कमांड, समस्या वाला इनपुट और `guidedog --version` का आउटपुट माँगता है।

प्रोग्राम के लिए रिपोर्ट

`--diagnostics=json` हर रिपोर्ट को मानक त्रुटि पर एक पंक्ति में एक JSON ऑब्जेक्ट के रूप में लिखता है, ताकि संपादक और निरंतर एकीकरण पढ़ सकें:

```
{
  "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": ""
}
```

यहाँ पाठ मोड़ा गया है; वास्तव में हर ऑब्जेक्ट एक पंक्ति में है। `schema` प्रारूप का संस्करण है, जो फ़ील्ड बदलने पर ही बदलता है। बाकी पढ़ने से पहले इसे जाँचें। कॉलम 1 से Unicode स्केलर गिनते हैं, बाइट या टर्मिनल खाने नहीं। `source` पूरी मूल पंक्ति है। टर्मिनल त्रुटि के आसपास सीमित हिस्सा दिखाता है और नियंत्रण अक्षर एस्केप करने तथा चौड़े अक्षर मापने के बाद निशान मिलाता है। `category` `suppress_warnings` वाली चेतावनी श्रेणी है, या खाली है।

मेमोरी कम पड़ने पर

मेमोरी कम पड़ना भी रिपोर्ट है, क्रैश नहीं। दो प्रकार हैं:

host.budget (मेमोरी बजट पूरा)

बिल्ड अपनी मेमोरी सीमा तक पहुँचा (`--budget=MIB`, डिफ़ॉल्ट 1 GiB)। रिपोर्ट चरण और टेम्पलेट

में मेमोरी माँगने वाली पंक्ति बताती है। टर्मिनल पर एक बार चुनाव मिलता है: पर्याप्त बजट बढ़ाएँ, कार्यशील मेमोरी डिस्क पर रखकर चलें, या रुकें। दूसरे वातावरण में बिल्ड रुकता है। `--budget` या `--memory=disk` से पहले चुनाव कर सकते हैं। संकेत बजट का अंक भी देता है, जैसे:

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

संख्या वर्तमान उपयोग में अस्वीकृत चरण की माँग का चार गुना जोड़ती है (चरण आउटपुट और लिखने की प्रति भी रखता है), या पुराने बजट का दोगुना लेती है—जो बड़ा हो। इसे 64 MiB तक ऊपर गोल किया जाता है। चरण खाली मेमोरी में समा सके तो सीमा उपलब्ध मेमोरी से अधिक नहीं। न समाए तो संकेत बताकर पहले `--memory=disk` सुझाता है:

```
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 (मेमोरी उपलब्ध नहीं)

सिस्टम ने बजट के भीतर की मेमोरी माँग ठुकराई। कमांड उस मेमोरी को छुए बिना रुकता है: बिल्ड कुछ प्रकाशित नहीं करता, GDS बदलाव पूरा होता है या बिल्कुल नहीं, और `convert` पूरा पेज लिखता है या कुछ नहीं। मेमोरी खाली करें या श्रेड घटाएँ (`-j 1`), फिर चलाएँ। प्रकाशन का निर्णय हो जाने पर आउटपुट बदलने को मेमोरी नहीं चाहिए; अस्वीकृति से अधूरा प्रकाशन नहीं होता। पुराने प्रकाशन को पूरा करने की योजना हेतु मेमोरी न हो तो उसे अगली बिल्ड के लिए जस का तस छोड़ा जाता है।

दोनों स्टेटस 3 पर समाप्त होते हैं।

एग्ज़िट स्टेटस

हर कमांड इनमें से एक स्टेटस पर समाप्त होता है, जिससे स्क्रिप्ट विफलता के प्रकार पहचान सकती हैं:

स्थिति अर्थ

- | | |
|-----|---|
| 0 | सफलता। आउटपुट लिखा गया और बिल्ड में प्रकाशित हुआ। |
| 1 | इनपुट में समस्या: दस्तावेज़ त्रुटि, नीति से अस्वीकृति (<code>--raw</code> के बिना <code>raw</code> सामग्री), न पढ़ा जा सकने वाला दस्तावेज़, या <code>-w</code> अथवा <code>--strict</code> से त्रुटि मानी गई चेतावनी। |
| 2 | कमांड का गलत उपयोग: अज्ञात फ़्लैग या कमांड, लापता या अमान्य मान, अथवा <code>conf.toml</code> या <code>-D</code> में अमान्य विकल्प। |
| 3 | सीमा पूरी हुई: मेमोरी बजट (<code>--budget</code>), नेस्टिंग गहराई (<code>--max-depth</code>), नोड संख्या (<code>--max-nodes</code>), स्टैक (<code>--stack-kib</code>), बहुत बड़ा स्रोत या सिस्टम से न मिल सकी मेमोरी। |
| 4 | फ़ाइल या बाहरी प्रोग्राम विफल हुआ: न पढ़ा या लिखा जा सकने वाला पथ, पढ़ते समय बदली फ़ाइल, या <code>Typst</code> का शुरू या पूरा न होना। |
| 5 | Guidedog में आंतरिक विफलता हुई और रिपोर्ट इसे बताने को कहती है। |
| 70 | Guidedog क्रैश होने पर GUIDEDOG CRASHED (internal error), चला कमांड, दोष और <code>guidedog --version</code> का आउटपुट देता है। रिपोर्ट की जगह बिल्ड का इश्यू ट्रैकर, या बिल्ड देने वाला व्यक्ति है। प्रकाशन अंत में होने से क्रैश पर पुराना आउटपुट बना रहता है। |
| 130 | कमांड <code>Ctrl+C</code> से रोका गया; प्रकाशित फ़ाइलें बदली नहीं गईं। |

किसी भी कारण से बिल्ड विफल हो तो पिछला आउटपुट और बिल्ड रिकॉर्ड वैसे ही रहते हैं। समस्या सुधारें और फिर बिल्ड करें।

Sphinx के साथ संगतता

परियोजना का माइग्रेशन आसान बनाने के लिए Guidedog उपयोगी जगहों पर Sphinx का अनुसरण करता है। संगतता हर सुविधा के लिए अलग जाँची जाती है। Python कॉन्फिगरेशन या मनमाने Python एक्सटेंशन नहीं चलते। **सफल बिल्ड, बिल्ड होने का प्रमाण है; पूरी समानता का नहीं।**

परियोजना और प्रकाशन

तालिका 20 परियोजना के काम

काम	लागू व्यवहार
परियोजना संरचना और बिल्ड विकल्प	Sphinx जैसे स्रोत और आउटपुट निर्देशिका विकल्प: <code>-a -E -W -n -D -t -j -b -M -c -C I</code>
कॉन्फिगरेशन	परिचित Sphinx नामों के साथ TOML डेटा। माइग्रेशन लिटरल बदलता है और गणना वाले मानों की सूचना देता है।
HTML बिल्डर	html, dirhtml और singlehtml। एकल पृष्ठ के एंकर में दस्तावेज़ का नाम शामिल होता है।
अन्य बिल्डर	text, gettext और dummy। pdf में Typst इस्तेमाल होता है; latex और latexpdf भी इसी रास्ते को चुनते हैं।
टेम्पलेट	Guidedog का Jinja कार्यान्वयन, टाइप वाला html_context, लेआउट इनहेरिटेंस और अतिरिक्त पृष्ठ।
इन्क्रिमेंटल बिल्ड	स्रोत, निर्भरताएँ, टेम्पलेट, कॉन्फिगरेशन और संदर्भ खोज बदलने पर संबंधित आउटपुट अमान्य होता है।
प्रकाशन	एक पूरी आउटपुट पीढ़ी प्रकाशित होती है। बिल्ड विफल होने पर पिछली प्रकाशित पीढ़ी बनी रहती है।

-D बूलियन के लिए true और false या 1 और 0 स्वीकार करता है। index न हो और contents हो तो चेतावनी के साथ उसे रूट बना सकते हैं। अतिरिक्त टेम्पलेट पृष्ठ साइट के रूट में फ़ाइलों के रूप में आते हैं, dirhtml में भी। ठीक नियम टेम्पलेट में हैं।

बिना बदले पृष्ठ के निदान भी दोबारा दिखते हैं। इसलिए नई और अपरिवर्तित बिल्ड में -w का अर्थ समान रहता है। दस्तावेज़ हटाने पर उसका पृष्ठ हटता है। प्रकाशन रिकवर होने वाला जर्नल इस्तेमाल करता है। अगली बिल्ड बीच में रुके कमिट को निपटाती है। इनवेरिफ़ेंट और प्लैटफ़ॉर्म सीमा एक पूरी आउटपुट पीढ़ी प्रकाशित करें में हैं।

epub, man, texinfo, linkcheck, doctest, coverage और changes बिल्डर उपलब्ध नहीं हैं। इनके नाम देने पर सुधार का संदेश मिलता है। रिपॉज़िटरी का tools/linkcheck अलग सत्यापन टूल है।

स्रोत भाषा

तालिका 21 निश्चित संस्करण की अनुरूपता जाँच

रीडर	प्रमाण और सीमाएँ
reStructuredText	कॉर्पस के 98 स्रोतों में तुलना किए गए Docutils वृक्ष और पहचानकर्ता मेल खाते हैं। स्रोत स्थान और कुछ आंतरिक प्रबंधन गुण तुलना में नहीं आते।
CommonMark 0.31.2	विनिर्देश के सभी 652 उदाहरण पास होते हैं।
MyST 0.16.1 reference	218 रीडर टेस्ट केस में 215 और 10 Sphinx बिल्ड केस में 8 मेल खाते हैं। बाकी अंतर दर्ज हैं।

इन टेस्ट के संस्करण निश्चित हैं। ये हर बाद के संस्करण से संगतता नहीं बताते। चल रहे प्रोग्राम के रीडर और रेंडर की जाँच के प्रमाण के लिए `guidedog formats` चलाएँ। रीडर निर्देशिकाओं में भी अनुरूपता संबंधी टिप्पणियाँ हैं।

`rst_prolog` आरंभिक ग्रंथसूची फ़ील्ड के बाद आता है। `rst_epilog` स्रोत के बाद आता है। दोनों Markdown में नहीं जुड़ते। `only` और `ifconfig` शर्त सही होने पर अपने खंड रखते हैं। शर्त वाला खंड आसपास का स्तर बदले तो उसका स्थान Sphinx से अलग हो सकता है। MyST के `eval-rst` में खंड शीर्षक त्रुटि हैं।

संदर्भ और डोमेन

Guidedog में `toctree`, लेबल, शब्दावली, खंड और चित्र क्रमांकन तथा `ref`, `doc`, `numref`, `term`, `download`, `any`, `keyword`, `option`, `envvar` और `token` संदर्भ लागू हैं। ऑब्जेक्ट विषय-सूची वृक्ष, साइडबार और पृष्ठ रूपरेखा में आ सकते हैं।

मानक, Python, C, C++, JavaScript, reStructuredText और गणित डोमेन लागू हैं। Odin डोमेन Guidedog का अपना एक्सटेंशन है। यह पैकेज, घोषणाएँ, सिग्नेचर और बने हुए API पृष्ठ दर्शाता है। केवल स्रोत से खोज और उसकी सीमाएँ Odin के दस्तावेज़ बनाना में हैं।

पैरामीटर, रिटर्न, अपवाद और वेरिएबल फ़ील्ड से संरचित विवरण बनता है। कैनोनिकल नाम संदर्भों और इन्वेंटरी में एलियस बनते हैं। Python के डिफ़ॉल्ट और एनोटेशन स्रोत का रूप रखते हैं। Sphinx इन्हें Python unparser से दोबारा लिख सकता है। C++ घोषणाएँ Sphinx-संगत पहचानकर्ता संस्करण और सिंबल इन्वेंटरी प्रकाशित करती हैं। नेस्टिंग की सीमाएँ लागू हैं। नेस्टेड कोष्ठक रैखिक समय में पार्स होते हैं।

`numref` एक संख्या स्थान बदलता है और बिना क्रमांक वाले लक्ष्य की सूचना देता है। खंड पढ़ने के क्रम में क्रमांकित होते हैं। दस्तावेज़ को एक बार क्रमांक मिलता है; दूसरी क्रमांकित `toctree` टकराव बताती है। फ़ॉर्मेट प्रतिस्थापन और पहचानकर्ता के विवरण संदर्भ टेस्ट में देखें।

सामान्य इंडेक्स, मॉड्यूल इंडेक्स और खोज बिल्ट-इन हैं। खोज शब्द के आरंभ से मिलान करती है; Sphinx का अंग्रेज़ी स्टेमिंग नहीं दोहराती।

घोषणात्मक ऑब्जेक्ट टाइप कई उपयोगी Python एक्सटेंशन पंजीकरण की जगह लेते हैं। `object_types`, `crossref_types` और `directive_aliases` उनका डेटा देते हैं। कस्टम Python `parse_node` को नाम, प्रदर्शन और प्रोग्राम के घोषणात्मक नियमों में बदलते हैं। माइग्रेशन वह कोड बताता है जिसे व्यक्त नहीं कर सकता। ऑब्जेक्ट टाइप घोषित करना देखें।

बिल्ट-इन एक्सटेंशन का व्यवहार

तालिका 22 एक्सटेंशन की सीमा

व्यवहार	स्थिति
<code>todo</code> , <code>ifconfig</code> , <code>extlinks</code> , <code>autosectionlabel</code>	बिल्ट-इन हैं। सीधे लिखे <code>extlinks</code> के लिए उपयुक्त रोल सुझाया जा सकता है।
<code>graphviz</code>	लिंक किया गया Graphviz स्थिर आरेख बनाता है। संसाधन सीमाएँ लागू होती हैं।
<code>intersphinx</code>	स्थानीय और डाउनलोड की गई इन्वेंटरी। Windows पर अभी केवल स्थानीय फ़ाइलें समर्थित हैं।
<code>githubpages</code>	स्थिर साइट के प्रकाशन की फ़ाइलें बिल्ट-इन हैं।
<code>mathjax</code> और <code>imgmath</code>	HTML के गणित में MathJax इस्तेमाल होता है। PDF समर्थित LaTeX उपसमुच्चय को Typst में बदलता है।
<code>myst_parser</code>	लागू परियोजना Markdown परत।
<code>doctest</code> निर्देश	सामग्री दिखाई जाती है। परियोजना बिल्डर इनके टेस्ट नहीं चलाता।
<code>autodoc</code>	पैकेज इम्पोर्ट या चलाए बिना Python स्रोत पढ़ा जाता है।

स्रोत आधारित autodoc रनटाइम पर बने हर ऑब्जेक्ट को नहीं खोज सकता। कम्पाइल किए मॉड्यूल, बाहरी डेकोरेटर, गणना वाले मान और एक्सटेंशन इवेंट हुक की स्पष्ट सीमाएँ हैं। Python 3.13 और निश्चित Sphinx documenter टेस्ट संदर्भ व्यवहार देते हैं। पूरे विकल्प और अपवाद autodoc से Python का दस्तावेज़ीकरण में हैं।

`autosummary`, `napoleon`, `viewcode` और मनमाने Python एक्सटेंशन इस रास्ते से बाहर हैं। कॉन्फ़िगरेशन में असमर्थित एक्सटेंशन की सूचना मिलती है। अज्ञात निर्देश या रोल स्रोत स्थान पर बताया जाता है। **छोड़ी गई सामग्री प्रकाशन से पहले जाँचना ज़रूरी है।** अन्य थीम नाम स्वीकार होते हैं, लेकिन Guidedog का थीम कार्यान्वयन इस्तेमाल होता है।

विश्वास और संसाधन सीमाएँ

सामान्य बिल्ड परियोजना का raw HTML और Typst, टेम्पलेट और नेटवर्क इन्वेंटरी स्वीकार करती है। स्थानीय पढ़ाई स्रोत निर्देशिका और स्पष्ट साझा रूट तक सीमित है। प्रतीकात्मक लिंक इस सीमा को नहीं बढ़ाता। टेम्पलेट, स्थिर फ़ाइलें, `include`, चित्र, फ़्रॉन्ट और API स्रोत यही नियम मानते हैं।

Graphviz के चित्र संसाधन दस्तावेज़ के सापेक्ष खोजे जाते हैं और सीमा में रहते हैं। समर्थित चित्र डेटा आरेख में समाहित होता है। `imagepath` और `fontpath` सीमा नहीं बढ़ाते। Typst पढ़ने योग्य निर्देशिकाओं वाला निजी रूट इस्तेमाल करता है। इससे किताब के टेम्पलेट और प्रीऐम्बल घोषित संसाधनों तक सीमित रहते हैं।

`--untrusted` पढ़ने की सीमा स्रोत फ़ोल्डर तक रखता है। raw सामग्री, असुरक्षित URL, बाहरी संसाधन, परियोजना के Typst टेम्पलेट और प्रीऐम्बल निदान के साथ छोड़े जाते हैं। इन्वेंटरी या Typst पैकेज नहीं लाए जाते। संसाधन पढ़ने वाले ग्राफ़ चेतावनी के साथ कोड में दिखते हैं। यह मोड नेटिव कम्पाइलर की मेमोरी या चलने का समय सीमित नहीं करता।

होस्ट का डिफ़ॉल्ट बजट 1 GiB है। आवंटन से पहले जगह आरक्षित होती है। पृष्ठ के कार्यक्षेत्र उपयोग के बाद मुक्त होते हैं; सहेजा परियोजना ग्राफ़ और कैटलॉग परियोजना बढ़ने पर मेमोरी लेते रहते हैं। पढ़ने वाले वर्कर प्रवेश का समन्वय करते हैं। अस्वीकृत अनुरोध में विफल काम और सुधार के विकल्प बताए जाते हैं।

`--memory=ram` प्रबंधित बजट पर रुकता है। `--memory=disk` प्रबंधित कार्य स्टोरेज को मेमोरी-मैप की गई अस्थायी फ़ाइलें इस्तेमाल करने देता है। सहेजा होस्ट स्टोरेज RAM बजट में गिना जाता

है। `--disk-budget` मैप स्टोरेज सीमित करता है; नीति डिस्क में आरक्षित जगह भी छोड़ती है। गैर-इंटरैक्टिव कमांड उत्तर की प्रतीक्षा नहीं करता। ज्ञात उपलब्ध क्षमता `GUIDEDOG_AVAILABLE_MIB` से दी जा सकती है।

`--max-depth`, `--max-nodes` और `--stack-kib` दस्तावेज़ के काम को सीमित करते हैं। सीमा बदलने पर पढ़ाई और आउटपुट कैश अमान्य होते हैं। गहराई और स्टैक क्षमता मेल खानी चाहिए। नेटिव Graphviz, tree-sitter और Typst आवंटन होस्ट बजट से बाहर हैं। कठोर सीमा के लिए ऑपरेटिंग सिस्टम की सीमाएँ इस्तेमाल करें। त्रुटि संदेश और मेमोरी और स्वामित्व देखें।

सत्यापन और वास्तविक परियोजनाएँ

`tools/sphinxdiff` निश्चित Sphinx टेस्ट रूटों में 42 की तुलना करता है। यह पृष्ठ, पहचानकर्ता, मुख्य पाठ के लिंक, इन्वेंटरी, क्रमांकन और निदान जाँचता है। सहेजे परिणाम और README जानबूझकर रखे अंतर समझाते हैं। उदाहरण हैं: सटीक स्रोत स्थान, स्थिर पढ़ने के क्रम में क्रमांकन और छोड़ी गई लक्ष्य-विशिष्ट raw सामग्री की स्पष्ट सूचना।

1 अक्टूबर 2026 के दूरस्थ सत्यापन में CPython, Django और Flask के HTML तथा PDF की नई और अपरिवर्तित बिल्ड चलीं। सभी बारह बिल्ड सफल रहीं और निदान तथा लिंक के परिणाम आधाररेखा से मेल खाए। साथ ही 1,039 Linux टेस्ट और 94 लक्षित AddressSanitizer टेस्ट पास हुए।

इन परियोजनाओं में असमर्थित Python एक्सटेंशन संरचनाएँ हैं। उनके निदान प्रमाण का हिस्सा बने रहते हैं। शून्य एग्जिट स्टेटस यह नहीं बताता कि सभी निजी निर्देश दोहराए गए हैं। प्रकाशन में कोई निदान न चाहिए तो `-w` चलाएँ।

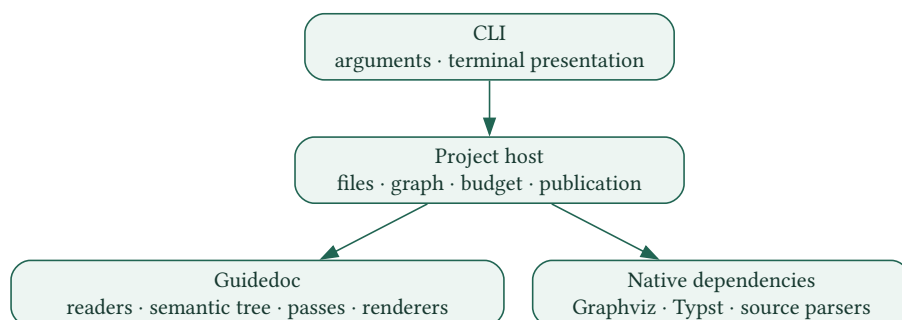
पुराने माप और स्रोत संस्करण `docs/manual/evidence/manuals.md` में हैं। नवीनतम दूरस्थ समीक्षा `build/review-remote-20261001/report.txt` में है। ये माप दर्ज काम का वर्णन करते हैं, सार्वभौमिक गति या मेमोरी सीमा का नहीं।

आंतरिक कार्यप्रणाली

ये अध्याय कार्यान्वयन के पीछे का तर्क और वर्तमान अनुबंध समझाते हैं। GDS अलग प्रोजेक्ट में निर्णय और समीक्षा की स्थिति रखता है।

Guidedoc इंजन

Guidedoc रूपांतरण का कोर है। वह कॉल करने वाले का स्टोरेज उधार लेता है। सोर्स पढ़ता, अर्थ-वृक्ष बनाता, पास चलाता, आउटपुट मापता और बाइट निकालता है। फ़ाइल खोलना या परिणाम प्रकाशित करना उसका काम नहीं।



चित्र 4 बाहरी प्रभावों की ज़िम्मेदारी होस्ट की है; दस्तावेज़ के अर्थ की ज़िम्मेदारी Guidedoc की।

सीमा

कोर रूपांतरण आवंटन नहीं करता। कॉल करने वाला नोड कॉलम, पाठ स्टोरेज, निदान और आउटपुट क्षमता देता है। होस्ट इन बफ़रों की तैयारी में आवंटन कर सकता है। बाहरी कंपाइलर के अपने नियम हैं। गारंटी रूपांतरण सीमा की है, पूरे प्रक्रम की नहीं।

वृक्ष

नोड के ID में प्रकार होता है। बार-बार उपयोग होने वाले कॉलम में प्रकार और उपवृक्ष का अंतराल है; अन्य तालिकाओं में विशिष्ट नोड का पेलोड है। समतल प्रीऑर्डर रूप से हीप पॉइंटर की कड़ी चले बिना पूरा उपवृक्ष छोड़ा जा सकता है।

नोड i का अपवर्जित उपवृक्ष अंत e_i मानें। उपवृक्ष $[i, e_i)$ में है। पत्ती में $e_i = i + 1$ है। जनक अपने वंशजों के बाद बंद होता है। अंतराल का प्रमाण एल्गोरिद्म से पहले इनवेरिएंट में है।

API

कोर API स्पष्ट है: स्टोरेज तैयार करें, रूपांतरण करें, परिणाम देखें, रीसेट करें। रजिस्ट्री रीडर, पास और रेंडर के विवरण रखने वाले मान हैं। रनटाइम प्लगइन खोज या वैश्विक पंजीकरण के दुष्प्रभाव नहीं। कम क्रियाएँ दूसरे Odin प्रोग्राम से उपयोग सरल रखती हैं।

स्वामित्व मुख्य नियम है। वर्कस्पेस के व्यू रीसेट या पुनः उपयोग पर अमान्य होते हैं। होस्ट का परिणाम सत्र से अधिक जीवित पाठ का मालिक हो। सटीक सिग्नेचर `core` में हैं; वे अलग `docs/api` जैसे Odin सोर्स से बनते हैं। मैनुअल PDF में संदर्भ उसी पुस्तक में आंतरिक लिंक सहित है। डिज़ाइन रिकॉर्ड `docs/gds` का GDS 0003 है।

एल्गोरिद्म से पहले इनवेरिएंट

उपयोगी अपरिवर्तनीय शर्त बताती है कि एल्गोरिद्म के चलते क्या सत्य रहता है और समाप्ति पर कॉल करने वाला किस पर भरोसा कर सकता है। लूप से पहले इसे लिखें।

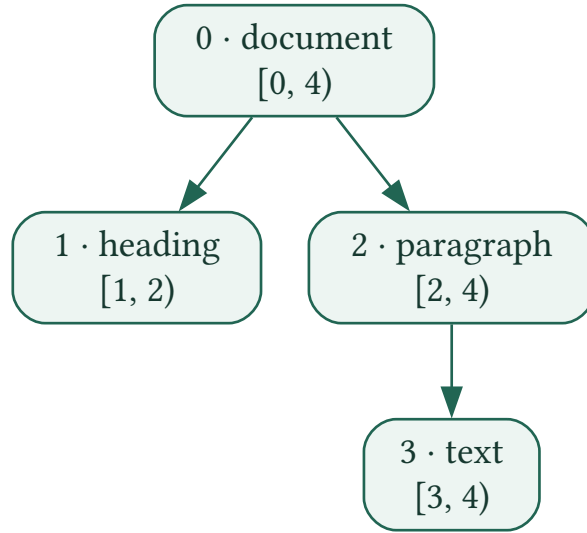
प्रीऑर्डर में उपवृक्ष के अंतराल

मानें कि वृक्ष प्रीऑर्डर में निकलता है: हर नोड अपने बच्चों से पहले और हर उपवृक्ष लगातार है। e_i नोड i और उसके सभी वंशजों के ठीक बाद का सूचकांक हो।

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

कथन। सूचकांक को e_i तक बढ़ाकर उपवृक्ष छोड़ा जा सकता है।

प्रमाण। लगातार होने से सभी वंशज e_i के पहले हैं। अपवर्जित अंत की परिभाषा से बाद के नोड e_i पर या उसके बाद हैं। वहाँ बढ़ना ठीक उसी उपवृक्ष को छोड़ता है। रेंडर के भरोसा करने से पहले वैलिडेटर धारणाएँ जाँचता है।



चित्र 5 माता-पिता का अंतराल अपने बच्चों के अंतरालों को समाहित करता है।

पूरा स्कैन हर नोड को एक बार देखता है; लागत $O(N)$ है। उपवृक्ष छोड़ने में एक सूचकांक बदलता है। पेलोड का काम अपनी लागत लेता है। स्थिर समय में छोड़ना, स्थिर समय में रेंडर करना नहीं है।

आउटपुट से पहले मापें

रैंडर पहले बाइटों की संख्या m मापता है। कॉल करने वाले की आउटपुट क्षमता c मानें।

$$m \leq c \implies \text{आउटपुट शुरू हो सकता है}.$$

मापने और निकालने के क्रम समान नियम अपनाते हैं। सफल आउटपुट ठीक m बाइट लिखे। विफलता वैध अधूरा परिणाम नहीं लौटाती। प्रकाशन होस्ट का काम है, इसलिए अधूरे बाइट पुराना आउटपुट नहीं बदल सकते।

कैश की वैधता

कैश कुंजी परिणाम बदल सकने वाले सभी डेटा को समेटे। केवल सोर्स काफ़ी नहीं: शामिल फ़ाइलें, टेम्पलेट, कॉन्फ़िगरेशन, संदर्भ खोज, संसाधन और अडैप्टर संस्करण भी असर डालते हैं। दर्ज निर्भरताएँ समान हों तभी पुनः उपयोग करें। कुंजी मिले पर आउटपुट न हो तो पुनः उपयोग अमान्य है।

टेस्ट कोर की सीमा पर विफल होने वाले एलोकैटर लगाकर निकली सामग्री मिलाते हैं। इंपोर्ट की भी जाँच होती है। केवल आवंटन गिनना किसी अस्वीकृति से चुपचाप खोए दस्तावेज़ हिस्से को छोड़ सकता है।

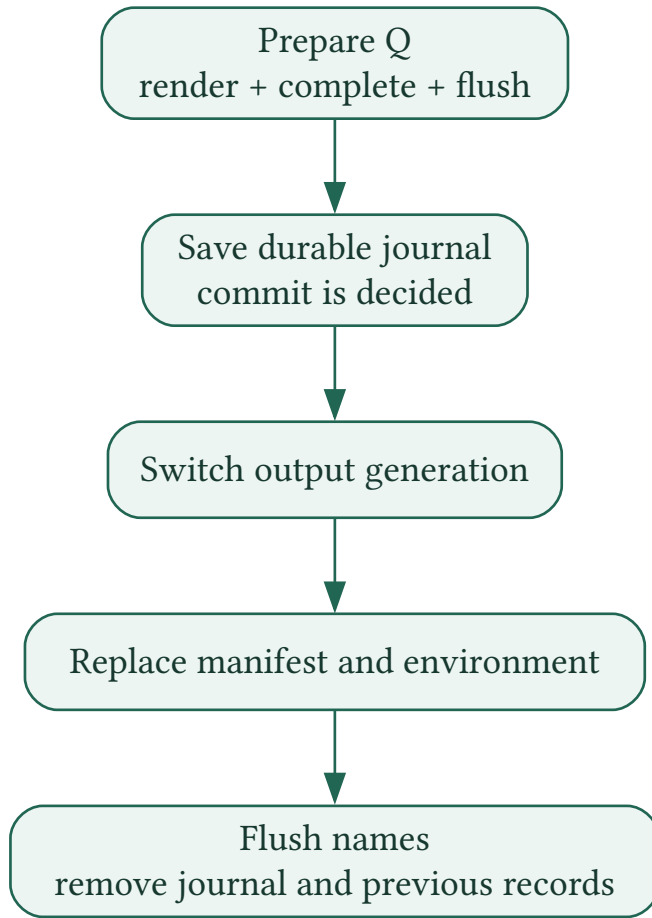
एक पूरी आउटपुट पीढ़ी प्रकाशित करें

पाठक को आधी बदली साइट नहीं दिखनी चाहिए। यह प्रकाशन का गुण है, पार्सिंग और रेंडरिंग से अलग।

इनवेरिगेंट

प्रकाशित आउटपुट वृक्ष P हो। तैयारी में P नहीं बदलता। अगला वृक्ष Q कमिट के निर्णय से पहले पूरा होता है। कमिट जर्नल आउटपुट और उसके रिकॉर्ड बताता है।

तैयारी विफल हो सकती है। होस्ट अधूरा संस्करण छोड़ता है। **तय कमिट बीच में रुक सकता है।** अगली बिल्ड जर्नल से उसे पूरा या बहाल करती है। स्टेजिंग फ़ोल्डर के नाम से अनुमान नहीं लगाती।



चित्र 6 स्थायी जर्नल तैयारी को तय हो चुके कमिट से अलग रखता है।

कैश आउटपुट के बाहर क्यों है

फ़ोल्डर बदलने पर उसकी पूरी सामग्री चलती है। अंदर रखा जर्नल उसी क्रिया में हिलेगा जिसका वर्णन करता है; रिकवरी के समय उसकी जगह स्थिर नहीं रहेगी। Guidedog रिकॉर्ड प्रकाशन वृक्ष के बाहर स्थिर फ़ोल्डर में रखता है। इसलिए कस्टम आउटपुट समान स्तर का कैश इस्तेमाल करता है।

टिकाऊपन और दृश्यता

फ़ाइल सामग्री डिस्क पर भेजना और फ़ोल्डर नाम भेजना अलग हैं। जर्नल हटाने से पहले एल्गोरिदम दोनों सँभालता है। प्लेटफ़ॉर्म के फ़ोल्डर बदलने के तरीके अलग हैं; रिकवरी प्रोटोकॉल इन्हें ध्यान में रखता है। कार्यान्वयन `lib/project/commit.odin` और `lib/project/publish.odin` में है।

यह प्रोजेक्ट का प्रकाशन सुरक्षित करता है। बाहरी होस्टिंग सेवा के साथ वितरित ट्रांज़ैक्शन नहीं है। सफल बिल्ड सर्वर पर कॉपी करना अलग क्रिया है, जिसकी ज़िम्मेदारी डिप्लॉयमेंट टूल की है।

मेमोरी और स्वामित्व

बजट हिसाब का अनुबंध है। खर्च के दर्ज रहने की अवधि उसके स्टोरेज की अवधि से मिले, तभी इसका अर्थ है।

गिनती स्टोरेज के साथ चलती है

आवंटन से पहले बाइट आरक्षित करें, स्टोरेज मुक्त करने के बाद खर्च हटाएँ। बफ़र बदलते समय पुराना और नया साथ रह सकते हैं। उस समय पूरा नया आवंटन गिनें। केवल बढ़ोतरी गिनना अधिकतम उपयोग कम दिखाएगा।

बदलते समय अधिकतम उपयोग = अन्य सक्रिय स्टोरेज + पुराना बफ़र + नया बफ़र .

सत्र के स्वामित्व वाला इनपुट सत्र वापस लेता है। कॉल करने वाले का इनपुट उसी की ज़िम्मेदारी है। `read_owned` और `read_kept` यह अंतर स्पष्ट करते हैं। कम पढ़ने से पहले से आवंटित जगह नहीं घटती।

ज्ञात शून्य अज्ञात नहीं है

मेमोरी जाँच पता लगा सकती है कि जगह नहीं बची, या जगह का पता लगाने में असफल हो सकती है। ये अलग स्थितियाँ हैं। नीति दोनों के लिए शून्य नहीं, अलग ज्ञात-स्थिति फ़्लैग रखती है। Linux पर दिखने वाले पूर्वज स्तरों के पठनीय cgroup प्रतिबंध लागू होते हैं। कंटेनर में केवल होस्ट की खाली मेमोरी भ्रामक हो सकती है।

रिपोर्ट कार्यक्षेत्र से अधिक समय तक रहती है

सत्र के बाद दिखने वाली रिपोर्ट अपने शीर्षक, सोर्स और सुधार पाठ की मालिक हो; मुक्त बफ़र उधार न ले। आंतरिक स्थान बाइट ऑफ़सेट हैं। उपयोगकर्ता के कॉलम Unicode स्केलर गिनते हैं। टर्मिनल निशान अलग से दृश्य चौड़ाई देखते हैं।

शेष सीमा

Guidedog अपना प्रबंधित कार्यशील स्टोरेज मापता है। Typst और मूल निर्भरताएँ अपने आवंटन सँभालती हैं। बड़े PDF होस्ट बजट से बहुत अधिक मेमोरी ले सकते हैं। दूरस्थ CPython परीक्षण में Typst सहित अधिकतम RSS लगभग 14.5 GiB था। यह उस काम का प्रमाण है, सार्वभौमिक सीमा नहीं। बजट पूरा होने के विकल्प त्रुटि संदेश में हैं।

इस मैनुअल के बारे में

यह मैनुअल लागू किए गए टूल और उसकी सोच-समझकर तय की गई सीमाओं को समझाता है। अलग GDS परियोजना में प्रस्ताव, निर्णय और समीक्षा का इतिहास दर्ज होता है। बेहतर टाइपोग्राफी से किसी प्रस्ताव की स्थिति नहीं बदलती।

इसका क्रम व्यावहारिक है: ट्यूटोरियल से सीखें, विषयवार मार्गदर्शन से समझें, काम करने के निर्देश अपनाएँ और संदर्भ पृष्ठों में बारीकियाँ जाँचें। आंतरिक संरचना वाले अध्याय अपरिवर्तनीय नियमों और प्रमाणों को समझाते हैं।

लेखन में छोटे उदाहरण, स्पष्ट धारणाएँ और सीधे कथन हैं। मकसद ऐसी व्याख्या देना है जिसे पाठक खुद जाँच सके। डिज़ाइन में सीमित रंग, स्पष्ट शीर्षक, पढ़ने योग्य serif फ़ॉन्ट और सुलभ नेविगेशन हैं। यह किसी लेखक की भाषा या पृष्ठ-सज्जा की नकल नहीं करता।

यह परियोजना `guidedog quickstart` से बनाई गई है और Guidedog से बिल्ड होती है। गद्य RST में, आरेख Graphviz में और समीकरण LaTeX गणितीय संकेतों में लिखे हैं। Typst का उपयोग केवल PDF टाइपसेटिंग टेम्पलेट में होता है।

बिल्ड के सत्यापन का रिकॉर्ड

2026-10-01 के रिमोट सत्यापन में CPython, Django और Flask के लिए साफ़ और incremental HTML तथा PDF बिल्ड चलाए गए। सभी बारह कमांड सफल रहे। निदान और लिंक-जाँच के नतीजे स्थापित आधाररेखा से मेल खाते थे। Linux के 1,039 टेस्ट और चुने गए sanitizer के 94 टेस्ट पास हुए। यह जाँचे गए संस्करण का प्रमाण है, पूर्ण Sphinx अनुकूलता का दावा नहीं। बची हुई सीमाओं के लिए Sphinx के साथ संगतता देखें।

दस्तावेज़ों के रखरखाव के कमांड इन दस्तावेज़ों का रखरखाव में हैं।

Guidedog

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

एक बार लिखें। साफ़ समझाएँ। वेब और कागज़ पर प्रकाशित करें।

Guidedog reStructuredText और Markdown से HTML दस्तावेज़ साइट और PDF पुस्तकें बनाता है। यह Odin में लिखा गया है। इसके रूपांतरण इंजन का नाम **Guidedoc** है और Typst PDF की पृष्ठ-सज्जा करता है। Python वातावरण या LaTeX स्थापित करने की ज़रूरत नहीं है।

लेखन की प्रक्रिया Sphinx से प्रेरित है, पर Guidedog एक स्वतंत्र प्रणाली है। कॉन्फ़िगरेशन घोषणात्मक TOML में होता है और API का स्थिर विश्लेषण होता है। Python कॉन्फ़िगरेशन या एक्सटेंशन कभी नहीं चलते। Sphinx से पूरी अनुकूलता लक्ष्य नहीं है।

परियोजना शुरू करें

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

परियोजना में `conf.toml`, `index.rst`, `_templates/layout.html`, `_templates/book.typ` और `_static/` होते हैं। टेम्पलेट सामान्य फ़ाइलें हैं जिन्हें सीधे बदल सकते हैं। HTML में नेविगेशन, खोज, कोड हाइलाइटिंग, आरेख और समीकरण मिलते हैं। PDF में अध्याय, पार-संदर्भ, चित्र, इंडेक्स और भाषा के अनुसार टाइपसेटिंग मिलती है।

डिफ़ॉल्ट थीम हल्की है; गहरी या सिस्टम थीम भी चुन सकते हैं। मैनुअल के मुख्य पृष्ठ और रिपोर्टरी परिचय पर ब्राउज़र की भाषा चुनी जाती है। सहेजी गई पसंद पहले लागू होती है। किसी भाषा या अध्याय के सीधे लिंक की पसंद नहीं बदली जाती।

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

स्थापित करें

हर रिलीज़ में Linux (x86-64 और ARM64), macOS (Apple silicon और Intel) और Windows (x86-64) के लिए एक आर्काइव है। हर आर्काइव में Typst, Graphviz और tree-sitter शामिल हैं, इसलिए साइट और पुस्तकें बनाने के लिए और कुछ स्थापित नहीं करना पड़ता।

Linux को glibc 2.35 या नया और libcurl चाहिए; macOS को संस्करण 12 या नया। अपने सिस्टम का आर्काइव और SHA256SUMS डाउनलोड करें, फिर जाँचकर स्थापित करें:

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

macOS पर आधिकारिक [Guidedog Homebrew tap](#) से स्थापित करें। Homebrew आपके Apple silicon या Intel प्रोसेसर के लिए सही पैकेज चुनता है। यह tap Linux पर भी काम करता है:

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

Windows पर आर्काइव को अनज़िप करें और उसके फ़ोल्डर को PATH में जोड़ें; Graphviz की DLL फ़ाइलें और config8 को guidedog.exe के पास ही रहने दें। प्रोग्रामों पर कोड-हस्ताक्षर नहीं है: macOS पर ब्राउज़र से डाउनलोड की गई फ़ाइल का क्वारंटीन xattr -d com.apple.quarantine guidedog से हटाएँ। हर प्लेटफ़ॉर्म की सीमाएँ रिलीज़ नोट्स में लिखी हैं।

स्रोत से बिल्ड करें

Odin dev-2026-10 (tools/install-toolchain.sh odin DIR इसे डाउनलोड करके जाँचता है), एक C कंपाइलर, और Linux पर libcurl, expat और zlib के डेवलपमेंट पैकेज स्थापित करें। पहले नेटिव लाइब्रेरी बनाएँ, फिर Guidedog:

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

इसके बाद build/guidedog PDF के लिए स्थापित Typst 0.15.1 का उपयोग करता है। रिलीज़ की तरह Typst कंपाइलर को अंतर्निहित करने के लिए Rust और Cargo स्थापित करके चलाएँ:

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

Windows पर Visual Studio के C++ टूल और LLVM के साथ Git Bash का उपयोग करें। Graphviz का Windows के लिए स्थिर बिल्ड नहीं है, इसलिए build.sh की जगह native/graphviz/windows.sh चलाएँ: यह Graphviz 16.1.0 की आधिकारिक DLL फ़ाइलें स्थापित करता है, जिन्हें tools/release/build.sh build/guidedog.exe के पास कॉपी करता है। tools/release/suites.sh टेस्ट सूट चलाता है, और [RELEASING.md](#) रिलीज़ की प्रक्रिया बताता है।

Guidedog से अनुवाद

```
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 फ़ाइलों में `msgstr` का अनुवाद लिखें और जाँचें। Guidedog POT निकालता है, कैटलॉग मिलाता है, प्रगति बताता है और हर भाषा का संस्करण बनाता है; अनुवाद खुद नहीं लिखता। कमांड, पथ, कोड, API नाम और संदर्भ लक्ष्य न बदलें। केवल मैनुअल और रिपॉज़िटरी परिचय का अनुवाद होता है; API और GDS अंग्रेज़ी में रहते हैं। सारे संस्करण और PDF जोड़ने के लिए [बिल्ड निर्देश](#) देखें।

GitHub पर ऊपर के लिंक से README की भाषा चुनें। भाषा का स्वतः चयन बनी हुई दस्तावेज़ वेबसाइट पर होता है।

पढ़ें और एम्बेड करें

- **मैनुअल:** ट्यूटोरियल, विषय, व्यावहारिक काम, संदर्भ और आंतरिक संरचना।
- **API संदर्भ:** Odin स्रोत से बनता है और मैनुअल PDF के परिशिष्ट में भी रहता है।
- **लाइब्रेरी गाइड:** कॉलर के कार्यक्षेत्र, मेमोरी और रूपांतरण उदाहरण।
- **टेम्पलेट:** Jinja से HTML और Typst से पुस्तक का रूप बदलें।
- **GDS:** क्रमांकित डिज़ाइन अभिलेख और कार्यान्वयन के निर्णय।
- **बीटा समीक्षा:** सुधार, सत्यापन, समर्थन की सीमा और रिलीज़ से पहले बचा काम।

Guidedoc कॉलर की दी हुई मेमोरी इस्तेमाल करता है और संरचित निदान लौटाता है। होस्ट फ़ाइल, बजट, कैश, प्रकाशन और मूल कंपाइलर सँभालता है। `--untrusted` क्षमताएँ सीमित करता है; यह प्रक्रिया का सैंडबॉक्स या सभी मूल संसाधनों की सीमा नहीं है। बीटा macOS और Linux पर भरोसेमंद दस्तावेज़ परियोजनाओं के लिए है।

श्रेय और लाइसेंस

Vikrant Rathore ने इसे बनाया, Ronak Rathore और Kanak Rathore ने सहयोग दिया। इश्यू और पुल रिक्वेस्ट के ज़रिए योगदान का स्वागत है। दस्तावेज़ में नाम या हैंडल दिए जाते हैं, निजी ईमेल पते नहीं। Git की पहचान संबंधी सेटिंग अलग सँभाली जाती हैं।

Sphinx और Georg Brandl, Docutils और David Goodger, Typst, Tree-sitter और Max Brunsfeld, Graphviz तथा Ginger Bill और Odin समुदाय का धन्यवाद।

Guidedog [GNU Affero General Public License v3.0](#) या उसके बाद के संस्करण (AGPL-3.0-or-later) के अंतर्गत जारी किया गया है। अन्य लाइसेंस शर्तों के लिए लेखकों से संपर्क करें। संपर्क का तरीका [NOTICE](#) में दिया गया है, साथ ही उन तृतीय-पक्ष घटकों की सूची भी, जिनके अपने लाइसेंस रहते हैं, जैसे शामिल फ़ॉन्ट।

एक अतिरिक्त अनुमति Guidedog को Graphviz के साथ लिंक करने और वितरित करने देती है; देखें [NOTICE](#)।

API Reference

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

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

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

autodoc

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

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

Types

autodoc.Config :: struct

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

autoclass_content: string

"class", "init", "both".

autodoc_class_signature: string

"mixed", "separated".

autodoc_default_options: []Default_Option

autodoc_docstring_signature: bool

autodoc_inherit_docstrings: bool

autodoc_member_order: string

"alphabetical", "bysource", "groupwise".

autodoc_mock_imports: []string

autodoc_preserve_defaults: bool

autodoc_typehints: string

"signature", "description", "none", "both".

autodoc_typehints_description_target: string

"all", "documented", "documented_params".

autodoc_typehints_format: string

"short", "fully-qualified".

autodoc_type_aliases: []Type_Alias

autodoc_use_type_comments: bool

python_display_short_literal_types: bool

strip_signature_backslash: bool

tab_width: int

typehint_fields: bool

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

autodoc.**Default_Option** :: struct

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

name: string

value: string

flag: bool

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

autodoc.**Message** :: struct

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

severity: Severity

code: string

title: string

text: string

hint: string

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

Option is a directive option as written; value is “” for a flag.

```
name: string
```

```
value: string
```

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

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

```
directive: string
```

```
argument: string
```

```
options: []Option
```

```
content: []string
```

```
module: string
```

```
class: string
```

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

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

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

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

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

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

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

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

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

```
Note
```

```
Warning
```

```
Error
```

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

```
name: string
```

```
target: string
```

Procedures

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

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

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

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

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

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

cli

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

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

Procedures

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

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

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

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

Constants

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

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

```
cli.LICENSE_NOTICE
```

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

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

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

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

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

core

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

Types

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

Generic

Attention

Caution

Danger

Error

Hint

Important

Note

Tip

Warning

Seealso

Todo

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

None

Left

Center

Right

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

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

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

```
owner: Node_Id
```

```
item: Node_Id
```

```
note: Node_Id
```

```
span: u32
```

```
number: i32
```

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

```
span: u32
```

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

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

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

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

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

index: []u32

per node: its first span, or NONE.

spans: []Text

ranges of bytes.

next: []u32

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

count: int

bytes: Buffer

prefix: string

prefix_length: int

the bytes the prefix takes before an id.

names: Symbol_Table

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

counter: int

labels: []u32

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

own: []u8

room for the longest name a node carries by itself.

extras: []Anchor_Extra

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

extra_count: int


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

None

Int

Span

Static

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

key: Attribute_Key

name: Text

the key text when key is Custom.

value: Text

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

Custom

Class

Id

Name

Language

Alt

Width

Height

Scale

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

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

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

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

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

```
snap: ^Snapshot
```

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

```
reports: ^Reports
```

```
limits: Limits
```

```
status: Status
```

```
need: Capacity_Need
```

```
work: u64
```

```
cancel: Cancel_Probe
```

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

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

```
user: rawptr
```

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

```
region: Region
```

```
minimum: u64
```

```
exact: bool
```

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

```
row_span: u16
```

```
col_span: u16
```

```
align: Align
```

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

```
next: u32
```

```
end: u32
```

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

```
Planned
```

```
Partial
```

```
Conformant
```

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

```
Nothing
```

```
Blocks
```

```
Inlines
```

```
Structure
```

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

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

```
Refuse
```

an error that names the construct and the policy.

```
Omit
```

a warning; the content is left out.

Allow

trusted content is emitted.

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

```
using msg: Message
```

```
span: Source_Span
```

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

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

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

```
kind: Arg_Kind
```

```
int: i64
```

```
span: Source_Span
```

```
static: string
```

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

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

```
ws: ^Workspace
```

```
generation: u32
```

```
revision: u32
```

the bank's revision when the document was built.

```
bank: u8
```

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

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

Docutils

Sphinx

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

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

Measure

Write

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

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

mode: Emit_Mode

out: []u8

written: int

source_map: []Map_Entry

mapped: int

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

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

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

```
user: rawptr
```

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

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

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

```
Attention
```

```
Caution
```

```
Danger
```

```
Error
```

```
Hint
```

```
Important
```

```
Note
```

```
Tip
```

```
Warning
```

```
See_Also
```

```
Todo
```

```
Contents
```

```
Version_Added
```

Version_Changed

Version_Deprecated

Version_Removed

Section_Author

Module_Author

Code_Author

Link_Heading

Link_Definition

Link_Image

Link_Table

Link_Code

Link_Equation

Link_Term

Parameters

Variables

Raises

Returns

Return_Type

Return_Values

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

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

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

```
max_depth: int
```

nested blocks and inline markup.

```
max_nodes: int
```

```
max_work: u64
```

reader and resolution steps.

```
max_include_depth: int
```

```
max_included_bytes: int
```

```
max_expansion: int
```

substitution nesting.

```
stack_bytes: int
```

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

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

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

```
text: string
```

offset: u32

core.Link_Info :: struct

destination: Text

title: Text

name: Text

normalized reference name, when named.

resolved: Node_Id

meaningful only when kind is Internal.

kind: Link_Kind

core.Link_Kind :: enum u8

External

destination is a URL or path.

Email

destination is an address; HTML adds mailto:.

Internal

resolved is a node in this document.

Unresolved

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

Project

under Policy.keep_documents_with_errors.

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

core.List_Info :: struct

style: List_Style

delimiter: u8

‘, ‘), or ‘(for “(1)”; bullet character for bullets.

start: i32

core.List_Style :: *enum* u8

Bullet

Arabic

Lower_Alpha

Upper_Alpha

Lower_Roman

Upper_Roman

core.Map_Entry :: *struct*

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

offset: u32

span: Source_Span

core.Math_Style :: *enum* u8

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

Plain

class “math”, the text in $(\)$ or $[\]$, as Docutils writes it.

Sphinx

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

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

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

Image

Video

Audio

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

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

code: string

severity: Severity

status: Status

title: string

message: string

“{0}” and “{1}” name the arguments.

hint: string

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

kind: Node_Kind

flags: Node_Flags

subtree_end: u32

value: i32

text: Text

payload: u32

span: Source_Span

attributes: Attribute_Range

core.Node_Flag :: *enum* u8

Tight

List: items render without paragraph spacing.

Anonymous

Reference or Target: anonymous hyperlink.

Auto

Footnote or reference: auto-numbered.

Symbol

Footnote or reference: auto-symbol.

Local

Contents: only the enclosing section.

Header

Row or Cell: belongs to a header.

Referenced

Target or Footnote: something refers to it.

Generated

Created by resolution rather than markup.

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

core.Node_Id :: *distinct* u32

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

Document

Title

Block structure.

Subtitle

Section

Heading

Paragraph

Block_Quote

Attribution

List

List_Item

Definition_List

Definition_Item

Term

Classifier

Definition

Field_List

Field

Field_Name

Field_Body

Option_List

Option_Item

Option_Group

Option

Option_Argument

Option_Description

Line_Block

Line

Code_Block

Preformatted

Doctest_Block

Math_Block

Raw_Block

Thematic_Break

Table

Table_Head

Table_Body

Row

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

Sphinx layer (GDS “Guidedog: everything Sphinx does, the Odin way”).

Toctree_Entry

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

Inline content.

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

Substitution_Reference**Inline_Target****Math****Raw_Inline****Soft_Break****Hard_Break****Subscript****Superscript****Title_Reference****Abbreviation****Role****Xref**

Sphinx layer inlines.

Desc_Name**Desc_Addname****Desc_Annotation****Desc_Parameterlist****Desc_Parameter****Desc_Optional****Desc_Returns**

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

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

```
name: string
```

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

```
user: rawptr
```

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

```
Unchanged
```

```
Rebuilt
```

```
Failed
```

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

```
raw: Content_Policy
```

raw HTML, raw Typst, raw LaTeX.

```
unsafe_urls: Content_Policy
```

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

```
math: Content_Policy
```

math the target cannot translate.

```
includes: bool
```

whether include directives may request resources.

```
strict: bool
```

warnings fail the conversion.

```
keep_documents_with_errors: bool
```

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

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

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

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

```
ws: ^Workspace
```

```
b: ^Builder
```

```
source: Source_Id
```

```
options: Read_Options
```

```
policy: Policy
```

```
limits: Limits
```

```
resources: Resource_Provider
```

```
reports: ^Reports
```

```
scratch: ^Scratch
```

```
included: int
```

bytes of included sources, charged against `max_included_bytes`.

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

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

```
data: rawptr
```

id: typeid

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

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

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

promote_title: bool

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

tab_width: int

RST tab stops; zero means 8.

today: string

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

extension: Read_Extension

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

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

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

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

using result: Result

document: Document

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

id: string

name: string

version: string

conformance: Conformance

read: Read_Proc

read_inline: Read_Inline_Proc

optional; see Read_Inline_Proc.

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

None

Nodes

Text

Attributes

Lists

Links

Cells

Sources

Scratch

Reports

Output

Source_Map

Second_Bank

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

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

readers: []Reader

```
renderers: []Renderer
```

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

```
view: View
```

```
doc: Document
```

```
out: ^Emitter
```

```
reports: ^Reports
```

```
options: Render_Options
```

```
policy: Policy
```

```
limits: Limits
```

```
scratch: ^Scratch
```

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

```
standalone: bool
```

a complete document rather than a body fragment.

```
reference_profile: bool
```

CommonMark reference HTML, for conformance tests.

```
title: string
```

overrides the document's own title.

```
language: string
```

BCP 47 tag; "en" when empty.

```
stylesheet: string
```

standalone HTML: CSS emitted in a style element.

```
section_numbers: bool
```

anchor_prefix: string

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

permalinks: bool

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

permalink_icon: string

“¶” when empty.

secnumber_suffix: string

after section numbers; “. “ when empty.

highlighter: Highlighter

labels: ^Labels

the words renderers write, translated; nil: English.

math: Math_Style

math_inline: [2]string

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

math_display: [2]string

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

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

core.Renderer :: struct

id: string

name: string

version: string

media_type: string

conformance: Conformance

render: Render_Proc

core.Reports :: struct

items: []Diagnostic

used: int

suppressed: int

truncated: bool

muted: bool

write mode repeats the checked encoding without reporting.

primary: Diagnostic

has_primary: bool

errors: int

warnings: int

fatal: Status

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

core.Request :: struct

source: Source

reader: string

renderer: string

read: Read_Options

render: Render_Options

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

optional; filled by renderers that support it.

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

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

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

user: rawptr

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

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

the host's memory budget cannot hold it.

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

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

```
status: Status
```

```
written: int
```

```
reports: []Diagnostic
```

```
primary: Diagnostic
```

```
need: Capacity_Need
```

```
mapped: int
```

source_map entries written.

```
suppressed: int
```

```
truncated: bool
```

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

```
bytes: []u8
```

```
used: int
```

```
high: int
```

high-water mark, for sizing evidence.

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

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

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

```
Note
```

```
Warning
```

```
Error
```

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

```
nodes: int
```

```
attributes: int
```

```
lists: int
```

```
links: int
```

```
cells: int
```

```
text: int
```

```
scratch: int
```

```
sources: int
```

```
reports: int
```

```
output: int
```

```
second_bank: bool
```

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

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

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

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

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

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

```
text: []u8
```

```
text_used: int
```

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

```
nodes: []Node
```

```
attributes: []Attribute
```

```
lists: []List_Info
```

```
links: []Link_Info
```

```
cells: []Cell_Info
```

```
text: []u8
```

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

```
name: string
```

```
text: string
```

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

```
using source: Source
```

```
parent: Source_Id
```

the including source; itself for the main source.

```
included_at: Source_Span
```

the include directive, when included.

```
use: Source_Use
```

how an included source is read.

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

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

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

source: Source_Id

start: u32

end: u32

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

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

Parsed

Literal

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

Stack is a bounded LIFO over caller-provided items.

items: []T

used: int

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

base: uintptr

limit: uintptr

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

Ok

Invalid_Input

Unsupported

Policy

Capacity

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

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

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

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

hash: u64**key:** string

borrowed; must outlive the table.

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

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

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

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

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

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

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

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

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

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

banks: [2]Snapshot

revisions: [2]u32

raised whenever a bank is cleared to be built again.

active: int

generation: u32

scratch: Scratch

sources: []Source_Entry

sources_used: int

reports: Reports

keep_errors: bool

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

output: []u8

optional; set by init_workspace_from_bytes for convert_text.

ready: bool

Procedures

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

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

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

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

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

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

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

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

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

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

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

add_text copies s into the text pool.

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

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

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

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

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

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

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

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

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

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

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

arg_int makes an integer diagnostic argument.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

buffer_write_byte appends one byte, as buffer_write does.

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

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

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

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

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

```
core.builder_fail :: proc(b: ^Builder, d: Diagnostic)
```

builder_fail reports d and poisons the builder with its status (the first failure wins); later builder calls do nothing.

```
core.builder_finish :: proc(b: ^Builder) -> Status
```

builder_finish closes the root. Every other node must already be closed.

```
core.builder_init :: proc(b: ^Builder, snap: ^Snapshot,
scratch: ^Scratch, reports: ^Reports, limits: Limits,
root_span: Source_Span) -> Status
```

builder_init starts a builder over snap with its open-node stack taken from scratch, and opens the root Document node at root_span. The engine calls it for every read and pass; tests use it to drive a reader's pieces. It returns Capacity, with b.need set, when scratch cannot hold the stack. The builder borrows every argument for one stage.

```
core.cell_info :: proc(v: View, id: Node_Id) -> Cell_Info
```

cell_info returns the cell payload of node id, or a cell spanning one row and one column when it has none.

```
core.check_render :: proc(ws: ^Workspace, doc: Document,
renderer: Renderer, request: Request) -> Result
```

check_render runs the renderer in measure mode and reports its exact output size, as need, and every support problem, without writing. doc must be a valid document of ws.

```
core.child_count :: proc(v: View, id: Node_Id) -> int
```

child_count counts the children of node id, in time linear in their number.

```
core.children :: proc(v: View, id: Node_Id) -> Children
```

children starts an iteration over the children of node id, for next_child.

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

code_caption is a code block's Caption child (its parsed Custom caption), or INVALID_NODE when it has none.

```
core.content_error :: proc(status: Status) -> bool
```

content_error tells whether an error's status describes the content (an unknown directive or role, a bad option, an unresolved reference, content refused by policy) rather than a tree that cannot be used: capacity, limits, cancellation, or a bug. Policy.keep_documents_with_errors keeps documents whose errors are all content errors.

```
core.content_model :: proc(k: Node_Kind) -> Content
```

content_model tells what children nodes of kind k may have; see child_allowed. The switch is exhaustive, so a new kind does not compile until it says.

```
core.contents_scope :: proc(v: View, contents: Node_Id) ->
(first, end: Node_Id, base: int)
```

contents_scope is the range of nodes a table of contents lists and the number of sections around that range.

```
core.convert :: proc(ws: ^Workspace, registry: Registry,
request: Request, output: [u8]) -> Result
```

convert reads, optionally transforms, and renders one source into output, which the caller owns; on success output[:written] holds it, and on failure written is 0. It starts a new workspace generation, which invalidates every earlier document, even on failure. It checks the registry first (validate_registry). The result borrows the workspace: its reports are valid until the next conversion or reset. It never allocates; every failure is a status with a primary diagnostic.

```
core.convert_text :: proc(ws: ^Workspace, registry: Registry,
reader, renderer, text: string, name := "input") -> (string,
Result)
```

convert_text converts text with default_request and the workspace's own output region (set by the slab initializers; convert with an output slice otherwise). The returned string borrows that region and is valid until the next conversion or reset of the workspace; on failure it is "" and the result says why.

```
core.copy_payload :: proc(b: ^Builder, v: View, from, to: Node_Id)
```

copy_payload copies a node's list, cell, or link row, with its texts, to the node to of b's document; passes that rebuild a document share it.

```
core.copy_subtree :: proc(b: ^Builder, v: View, id: Node_Id)
-> Node_Id
```

copy_subtree copies a node and its descendants from a document view into a builder, including text, attributes, and payload rows, and returns the copy of id. Copies are appended in order, one per node, so the copy of a descendant d is copy + (d - id): a pass can adjust copies with set_value, set_text, add_flags, and set_kind. Internal link targets keep their numbering only when the pass copies the whole document in order. On failure the builder is poisoned and the result is INVALID_NODE.

```
core.current :: proc(b: ^Builder) -> Node_Id
```

current returns the innermost open node, or INVALID_NODE when none is open.

```
core.current_kind :: proc(b: ^Builder) -> Node_Kind
```

current_kind returns the kind of the innermost open node; Document when none is open.

```
core.custom_attribute :: proc(v: View, id: Node_Id, name: string) -> (string, bool)
```

custom_attribute returns the value of node id's Custom attribute named name, borrowed from the view, or false when it has none.

```
core.decode_last_rune :: proc(s: string, end: int) -> (rune, int)
```

decode_last_rune decodes the UTF-8 character ending at byte end of s, as decode_rune.

```
core.decode_rune :: proc(s: string, i: int) -> (rune, int)
```

decode_rune decodes the UTF-8 character starting at byte i of s and returns it with its width; an invalid byte decodes as the replacement character, width 1.

```
core.default_limits :: proc() -> Limits
```

default_limits returns the documented limits by value; zero limits never mean unlimited.

```
core.default_policy :: proc() -> Policy
```

default_policy returns the safe policy by value: raw content, unsafe URLs, and math the target cannot translate are refused; includes may be requested.

```
core.default_read_options :: proc() -> Read_Options
```

default_read_options returns the read options by value: RST title promotion on, tab width 8, and no reader settings.

```
core.default_request :: proc(name, text, reader, renderer:
string) -> Request
```

default_request assembles a complete Request with the documented defaults: default_read_options, a standalone render, default_policy, and default_limits, with no passes, resource provider, or cancellation probe. It borrows name and text. Change fields afterwards; lib/defaults has typed read options for the built-in readers.

```
core.diagnostic :: proc(m: Message, span: Source_Span, a :=
Diagnostic_Arg{}, b := Diagnostic_Arg{}) -> Diagnostic
```

diagnostic builds a Diagnostic by value from a static message, a span, and up to two arguments; it copies nothing.

```
core.document_valid :: proc(doc: Document) -> bool
```

document_valid reports whether doc still names the snapshot it was built as: the same generation, the active bank, and the bank's revision at that time.

```
core.dump_tree :: proc(v: View, e: ^Emitter)
```

dump_tree writes a readable, deterministic outline of a document. Tests use it as a renderer-independent golden format:

```
document
  section value=1
    heading value=1
      text "Title"
```

```
core.emit :: proc(e: ^Emitter, s: string)
```

emit writes s: Measure mode counts it, Write mode also stores it in the output.

```
core.emit_byte :: proc(e: ^Emitter, c: u8)
```

emit_byte writes one byte, as emit does.

```
core.emit_int :: proc(e: ^Emitter, value: i64)
```

emit_int writes value in decimal, as emit does.

```
core.emit_mark :: proc(e: ^Emitter, span: Source_Span)
```

emit_mark records that output from here on originates at span.

```
core.emit_rune :: proc(e: ^Emitter, r: rune)
```

emit_rune writes r UTF-8 encoded, as emit does.

```
core.encode_rune :: proc(r: rune) -> ([4]u8, int)
```

encode_rune returns r encoded as UTF-8 and the number of bytes used.

```
core.end_node :: proc(b: ^Builder, id: Node_Id)
```

end_node closes id, which must be the innermost open node, and records its subtree end; closing out of order poisons the builder.

```
core.extension_as :: proc(e: Read_Extension, $T: typeid) ->
(settings: ^T, ok: bool)
```

extension_as returns the settings e carries as ^T: nil and true when e is empty, nil and false when it carries another type. It neither reports nor allocates.

```
core.extra_anchor :: proc(a: ^Anchors, owner, item: Node_Id)
-> (string, bool)
```

extra_anchor returns the id of an element generated for owner and item.

```
core.extra_text :: proc(a: ^Anchors, e: Anchor_Extra) ->
string
```

extra_text is the id of a generated element.

```
core.find_reader :: proc(r: Registry, id: string) -> (Reader,
bool)
```

find_reader returns the reader of r with id, by value, or false.

```
core.find_renderer :: proc(r: Registry, id: string) ->
(Renderer, bool)
```

find_renderer returns the renderer of r with id, by value, or false.

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

first_child returns the first child of node id, or INVALID_NODE when it has none.


```
core.first_problem :: proc(r: ^Reports) -> Diagnostic
```

first_problem returns the primary error, or the first warning when there is none.

```
core.has_children :: proc(v: View, id: Node_Id) -> bool
```

has_children tells whether node id has any child.

```
core.has_class :: proc(v: View, id: Node_Id, class: string) -> bool
```

has_class looks through every class attribute; a value may hold several classes.

```
core.id_fold :: proc(r: rune) -> string
```

id_fold is the ASCII transliteration make_id gives a lowercase rune (“é” gives “e”, “2” gives “2”), from Unicode’s decompositions, or “” when it has none. The string is static.

```
core.include_source :: proc(ctx: ^Read_Context, from: Source_Id, name: string, span: Source_Span, use := Source_Use.Parsed) -> (Source_Id, bool)
```

include_source asks the host for a resource named by a directive in source from, records it in the source table, and returns its ID. Every refusal is reported at span. use says whether the reader parses the text or shows it as it is.

```
core.init_workspace :: proc(ws: ^Workspace, storage: Storage) -> Result
```

init_workspace borrows storage. It checks alignment-free invariants: every region is disjoint, and bank A can hold at least a root node. Initializing a workspace again invalidates every document of its earlier storage.

```
core.init_workspace_from_bytes :: proc(ws: ^Workspace, slab: []u8) -> Result
```

init_workspace_from_bytes carves the regions of slab_plan(estimate) from slab, one bank, for the largest input estimate whose plan fits, and initializes the workspace. The workspace borrows slab, which must outlive it; a slab too small for any plan is a capacity result. For passes that rebuild, use init_workspace_with_plan.

```
core.init_workspace_with_plan :: proc(ws: ^Workspace, slab: []u8, p: Slab_Plan) -> Result
```

init_workspace_with_plan carves exactly the regions of p from slab, and the rest of the slab becomes scratch. It is the authoritative path:

`init_workspace_from_bytes` only chooses a plan. A plan that cannot be sized is `Invalid_Input`; a slab smaller than `plan_bytes(p)` is a capacity result.

```
core.is_ascii_alnum :: proc(c: u8) -> bool
```

`is_ascii_alnum` tells whether `c` is an ASCII letter or digit.

```
core.is_ascii_digit :: proc(c: u8) -> bool
```

`is_ascii_digit` tells whether `c` is an ASCII digit.

```
core.is_ascii_punct :: proc(c: u8) -> bool
```

`is_ascii_punct` tells whether `c` is ASCII punctuation, as `CommonMark` defines it.

```
core.is_ascii_space :: proc(c: u8) -> bool
```

`is_ascii_space` tells whether `c` is ASCII white space: space, tab, LF, CR, FF, or VT.

```
core.is_unicode_punctuation :: proc(r: rune) -> bool
```

`is_unicode_punctuation` follows `CommonMark 0.31`: general categories P and S.

```
core.is_unicode_whitespace :: proc(r: rune) -> bool
```

`is_unicode_whitespace` follows `CommonMark`: Zs characters, tab, LF, FF, and CR.

```
core.kept_message :: proc(policy: Policy, m: Message) -> Message
```

`kept_message` is `m` as a renderer reports it: a content error becomes a warning when the policy keeps documents with errors, so the page is still written.

```
core.kind :: proc(v: View, id: Node_Id) -> Node_Kind
```

`kind` returns the kind of node `id`.

```
core.kind_name :: proc(k: Node_Kind) -> string
```

`kind_name` returns the static name of a node kind, as dumps and diagnostics write it.

```
core.label :: proc(o: Render_Options, l: Label) -> string
```

`label` returns a label in the render's language, or in English when it has none.

```
core.line_bounds :: proc(text: string, offset: int) ->
(start, end: int)
```

line_bounds returns the byte range of the line containing offset, without the break.

```
core.line_column :: proc(text: string, offset: int) -> (line,
column: int)
```

line_column converts a byte offset into a 1-based line and byte column.

```
core.link_info :: proc(v: View, id: Node_Id) -> (Link_Info,
bool)
```

link_info returns the link payload of node id, or false and an unresolved link when it has none.

```
core.list_info :: proc(v: View, id: Node_Id) -> List_Info
```

list_info returns the list payload of node id, or a bullet list starting at 1 when it has none.

```
core.load_object :: proc(ws: ^Workspace, bytes: []u8, limits:
Limits) -> Read_Result
```

load_object rebuilds a document from bytes written by write_object into bank A of a new workspace generation. bytes must outlive the document: source names and texts are views into them.

```
core.make_id :: proc(out: ^Buffer, name: string)
```

make_id turns a name into an identifier matching `[a-z](-?[a-z0-9]+)*`: letters are lowercased and transliterated to ASCII (ID_FOLDS, derived from Unicode data), every other run of characters becomes one hyphen, and leading digits and hyphens are dropped, as published Sphinx URLs expect. The result may be empty.

```
core.media_kind :: proc(path: string) -> Media_Kind
```

media_kind reads a path's extension, ignoring case, a query, and a fragment.

```
core.next_anchor_id :: proc(a: ^Anchors, it: ^Anchor_Ids) ->
(string, bool)
```

next_anchor_id returns the next id of an anchor_ids iteration, borrowed from the anchor table; false ends it.

```
core.next_attribute :: proc(v: View, it: ^Attributes) ->
(Attribute, bool)
```

next_attribute returns the next attribute of an attributes iteration, whose Text values index the view's pool; false ends it.

```
core.next_child :: proc(v: View, it: ^Children) -> (Node_Id,
bool)
```

next_child returns the next child of a children iteration; false ends it.

```
core.nfd_first :: proc(s: string) -> rune
```

nfd_first returns the first character of s in NFD, as Sphinx takes an index entry's letter: `unicodedata.normalize("NFD", s[o])[o]`.

```
core.no_reader_settings :: proc(ctx: ^Read_Context) -> bool
```

no_reader_settings is reader_settings for a reader that takes no settings: any extension is reported (CORE_SETTINGS_NONE) and the result is false.

```
core.node :: proc(v: View, id: Node_Id) -> Node
```

node returns node id by value.

```
core.node_count :: proc(v: View) -> int
```

node_count returns the number of nodes in the document, the root included.

```
core.node_text :: proc(v: View, id: Node_Id) -> string
```

node_text returns the text of node id, borrowed from the view.

```
core.normalize_label :: proc(out: ^Buffer, label: string)
```

normalize_label writes the case-folded label with runs of white space collapsed to one space and the ends trimmed. It is used for CommonMark labels and RST names.

```
core.normalize_nfd_into :: proc(dst: []u8, s: string) -> (n:
int, ok: bool) #optional_ok
```

normalize_nfd_into writes s in Normalization Form D into dst, the caller's storage, and returns the length of the whole normal form. It never allocates and never borrows s: the result is always `dst[:n]`. When dst is too small, ok is false, n is the size dst needs, and dst holds only a prefix; a nil dst measures.

```
core.offset_in :: proc "contextless" (outer, inner: string) -> (int, bool)
```

offset_in returns where inner starts in outer when inner is a slice of outer's bytes, and false otherwise (an empty inner included). It compares addresses only and reads neither string, so it is safe for any two strings.

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

parent_of scans backwards; it is linear and meant for diagnostics, not traversal.

```
core.plain_text :: proc(v: View, id: Node_Id, out: ^Buffer)
```

plain_text writes the concatenated text of a subtree, as used for alt text and ids.

```
core.plan_bytes :: proc(p: Slab_Plan) -> (bytes: int, ok: bool) #optional_ok
```

plan_bytes returns the slab size a plan needs. ok is false when a capacity is negative or the size overflows, which init_workspace_with_plan reports.

```
core.pool_string :: proc(b: ^Builder, t: Text) -> string
```

pool_string returns the text t of the builder's pool; it borrows the bank for the stage.

```
core.pop :: proc(s: ^Stack($T)) -> (v: T, ok: bool)
```

pop removes and returns the top item; false when the stack is empty.

```
core.push :: proc(s: ^Stack($T), v: T) -> bool
```

push appends v; false, changing nothing, when the stack is full.

```
core.read_document :: proc(ws: ^Workspace, reader: Reader, request: Request) -> Read_Result
```

read_document parses the request's source into bank A of a new generation and validates it. The document borrows the workspace and the request's source text, which must outlive it; it is valid until the next conversion, reset, or rebuild of bank A, and is returned only on success.

```
core.read_extension :: proc(settings: ^$T) -> Read_Extension
```

read_extension wraps a pointer to a reader's settings with their type. It borrows settings, which must outlive the read.

```
core.reader_settings :: proc(ctx: ^Read_Context, $T: typeid)
-> (settings: ^T, ok: bool)
```

reader_settings is extension_as at a reader's entry: settings of another type are reported (CORE_SETTINGS_TYPE, naming both types' packages) and ok is false; the reader then returns Invalid_Input before reading anything. nil and true mean the defaults.

```
core.render_document :: proc(ws: ^Workspace, doc: Document,
renderer: Renderer, request: Request, output: [u8] -> Result
```

render_document checks and measures, then writes into the caller's output only if it fits; otherwise the result is Capacity with the exact size needed, and output is untouched. doc must be a valid document of ws, and stays valid.

```
core.report :: proc(r: ^Reports, d: Diagnostic)
```

report records a diagnostic. The first error is kept by value in primary even when storage is full, so capacity failures remain explainable.

```
core.report_message :: proc(r: ^Reports, m: Message, span:
Source_Span, a := Diagnostic_Arg{}, b := Diagnostic_Arg{})
```

report_message reports the message m at span with its arguments, as report does.

```
core.reset_workspace :: proc(ws: ^Workspace)
```

reset_workspace invalidates every document and clears all used-counts.

```
core.resolve_link :: proc(b: ^Builder, row: u32, target:
Node_Id, kind: Link_Kind, destination := Text{})
```

resolve_link records the resolution of a link row during a reader's resolve stage.

```
core.scratch_buffer :: proc(s: ^Scratch, n: int) -> (b:
Buffer, missing: u64, ok: bool)
```

scratch_buffer takes a Buffer of n bytes from scratch, usually the wanted count of a measuring pass into a zero Buffer; ok is false, with the bytes missing, when scratch cannot hold it. The caller releases the scratch.

```
core.scratch_mark :: proc(s: ^Scratch) -> Scratch_Mark
```

scratch_mark records how much scratch is taken, for scratch_release.

```
core.scratch_release :: proc(s: ^Scratch, m: Scratch_Mark)
```

scratch_release gives back every scratch item taken since m; slices taken after m must not be used afterwards.

```
core.scratch_take :: proc(s: ^Scratch, $T: typeid, n: int) ->
(items: []T, missing: u64, ok: bool)
```

scratch_take returns n zeroed elements of T, or ok = false with the bytes missing.

```
core.set_cell :: proc(b: ^Builder, id: Node_Id, info:
Cell_Info)
```

set_cell stores info as node id's cell payload; a full cells region poisons the builder with Capacity.

```
core.set_kind :: proc(b: ^Builder, id: Node_Id, k: Node_Kind)
```

set_kind changes the kind of node id, as a pass does to a copy; validation checks the result. Nothing happens once the builder has failed.

```
core.set_link :: proc(b: ^Builder, id: Node_Id, info:
Link_Info) -> u32
```

set_link stores info as node id's link payload and returns its row, for resolve_link, or NONE when the builder has failed or the links region is full (Capacity).

```
core.set_list :: proc(b: ^Builder, id: Node_Id, info:
List_Info)
```

set_list stores info as node id's list payload; a full lists region poisons the builder with Capacity.

```
core.set_span_end :: proc(b: ^Builder, id: Node_Id, end: u32)
```

set_span_end moves the end of node id's source span, as a reader learns where a construct ends; nothing happens once the builder has failed.

```
core.set_text :: proc(b: ^Builder, id: Node_Id, t: Text)
```

set_text sets the text of node id; nothing happens once the builder has failed.

```
core.set_value :: proc(b: ^Builder, id: Node_Id, v: i32)
```

set_value sets the value of node id (a heading level, a list count, and so on); nothing happens once the builder has failed.

```
core.slab_plan :: proc(input_bytes: int, second_bank :=
false) -> (p: Slab_Plan, ok: bool) #optional_ok
```

`slab_plan` estimates capacities for an input of the given size. The estimate is generous for prose and typical technical documents; tables and heavy markup may need more, which a capacity result reports. `ok` is false when a capacity would overflow.

```
core.source_entry :: proc(v: View, id: Source_Id) ->
Source_Entry
```

`source_entry` returns the source-table entry of `id` (the main source is 0), 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"
```

उपलब्धता: 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++"}
```

उपलब्धता: 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"}
```

उपलब्धता: 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"}
```

उपलब्धता: Windows.

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

उपलब्धता: 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)
```

उपलब्धता: 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)
```

उपलब्धता: 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
```

उपलब्धता: 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
```

उपलब्धता: 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
```

उपलब्धता: 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
```

उपलब्धता: 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"
```

उपलब्धता: Windows.

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

उपलब्धता: 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

```
il8n.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.

```
il8n.destroy_template :: proc(t: ^Template)
```

destroy_template frees a template and everything added to it, and clears it.

```
il8n.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.

```
il8n.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 (booleans and integral floats accepted).

```
jinja.autoescaping :: proc(c: ^Call) -> bool
```

autoescaping reports whether the call happens while autoescape is on.

```
jinja.boolean :: proc(b: bool) -> Value
```

boolean makes a boolean value.

```
jinja.call_filter :: proc(s: ^State, name: string, args:
[]Value, kwargs: []Keyword) -> Value
```

call_filter runs a registered filter by name (used by map, and by hosts).

```
jinja.call_test :: proc(s: ^State, name: string, args:
[]Value, kwargs: []Keyword) -> bool
```

call_test runs a registered test by name (used by select and reject).

```
jinja.check_args :: proc(c: ^Call, who: string, names:
[]string, required := 0) -> bool
```

check_args enforces a Python signature: at most len(names) positional arguments and only the named keywords, each at most once. who names the callable in errors.

```
jinja.clear_cache :: proc(env: ^Environment)
```

clear_cache forgets parsed templates so the next load reads them again.

```
jinja.create_environment :: proc(options: Options = {}) -  
> Environment
```

create_environment makes an environment with options, zero fields taking their defaults. It keeps its own arena, backed by context allocator, for parsed templates and registrations; destroy_environment frees it. When that memory cannot hold the environment's tables, out_of_memory is set and every parse fails with a Memory error.

```
jinja.destroy_environment :: proc(env: ^Environment)
```

destroy_environment frees the environment's arena, with every template parsed in it, and its loader's state (Loader.destroy).

```
jinja.destroy_error :: proc(err: Error, allocator :=  
context.allocator)
```

destroy_error frees an error returned by this package.

```
jinja.dict :: proc() -> Value
```

dict makes an empty dict, allocated with context.allocator (Undefined when it refuses the memory); fill it with set().

```
jinja.escape :: proc(v: Value, allocator :=  
context.allocator) -> Markup
```

escape converts a value to Markup: Markup stays as it is, everything else is converted to a string and escaped (MarkupSafe's escape()), allocating with allocator when it must; the result may borrow v, so it belongs in an arena, as values do.

```
jinja.escape_string :: proc(s: string, allocator :=  
context.allocator) -> string
```

escape_string replaces the five HTML special characters as MarkupSafe does. When s has none it is returned as it is, borrowed; otherwise the result is allocated with allocator, which the caller owns (or, as usual, an arena holds).

```
jinja.evaluate :: proc(env: ^Environment, expression: string,  
ctx: Value = {}) -> (Value, Error)
```

evaluate evaluates one expression against ctx (Jinja's compile_expression). The value and an error's message are allocated with context.allocator, the caller's; the lists and dicts of the value grow through a small allocator

record kept there too. When that allocator refuses memory the error is a Memory error.

```
jinja.fail :: proc(c: ^Call, format: string, args: ..any) -> Value
```

fail records a runtime error from a native procedure; return its result. The message is formatted with context.allocator, the render's arena.

```
jinja.filesystem_loader :: proc(directories: []string) -> Loader
```

filesystem_loader serves templates from directories, searched in order; the first match wins. Names use "/" and may not climb out of a directory with "...". The loader copies directories with context.allocator and frees them when its environment is destroyed (Loader.destroy). When that allocator refuses the copy, the loader finds nothing.

```
jinja.float :: proc(f: f64) -> Value
```

float makes a floating-point value.

```
jinja.format_error :: proc(err: Error, allocator := context.allocator) -> string
```

format_error renders an error as a friendly report in the style of Elm's compiler, in a string allocated with allocator that the caller owns (" for no error):

```
-- UNDEFINED VALUE ----- page.html:3
'user' is undefined
3| <h1>{{ user.name }}</h1>
Hint: Pass it in the render context, or test it with `is defined`.
```

```
jinja.function :: proc(name: string, procedure: Native, user: rawptr = nil) -> Value
```

function wraps a native procedure as a callable value for globals or context data, allocated with context.allocator (Undefined when it refuses the memory); name and user are borrowed.

```
jinja.get :: proc(d: Value, key: string) -> Value
```

get reads a string key from a dict value; missing keys give Undefined.

```
jinja.has_arg :: proc(c: ^Call, index: int, name: string) -> bool
```

has_arg reports whether the argument was passed at all.

```
jinja.integer :: proc(i: i64) -> Value
```

integer makes an integer value (int would shadow Odin's type).

```
jinja.is_markup :: proc(v: Value) -> bool
```

is_markup reports a safe-HTML string.

```
jinja.is_none :: proc(v: Value) -> bool
```

is_none reports None.

```
jinja.is_undefined :: proc(v: Value) -> bool
```

is_undefined reports an undefined value.

```
jinja.keys :: proc(d: Value, allocator := context.allocator)
-> []string
```

keys returns the string keys of a dict value, in insertion order, in a slice allocated with allocator that the caller owns; the keys borrow the dict.

```
jinja.length :: proc(v: Value) -> int
```

length is the number of items of a list or dict, or of runes of a string.

```
jinja.list :: proc(items: ..Value) -> Value
```

list makes a list holding the given values, allocated with context.allocator; when it refuses the memory, the result is Undefined.

```
jinja.load_template :: proc(env: ^Environment, name: string)
-> (Template, Error)
```

load_template loads a template through the environment's loader, cached in the environment's arena until destroy_environment (or clear_cache). An error is allocated with context.allocator; the caller frees it with destroy_error.

```
jinja.markup :: proc(s: string) -> Value
```

markup makes a string value that is safe HTML, which autoescaping leaves alone; the value borrows s.

```
jinja.memory_loader :: proc(sources: ^map[string]string) -
> Loader
```

memory_loader serves templates from a map the host owns (name -> source).

```
jinja.none :: proc() -> Value
```

none makes Python's None.

```
jinja.parse_template :: proc(env: ^Environment, name, source:
string) -> (Template, Error)
```

parse_template parses source directly (not cached; not visible to includes). The template lives in the environment's arena, which copies source; an error is allocated with context allocator (destroy_error).

```
jinja.render :: proc(t: ^Template, ctx: Value = {}) ->
(string, Error)
```

render renders a template with ctx (a dict value, or Undefined for none). The result and any error message are allocated with context allocator; everything else lives in a scratch arena freed before returning. When that allocator refuses memory the render stops with a Memory error at the line it was rendering.

```
jinja.render_position :: proc() -> (template: string, line:
int, excerpt: string, ok: bool)
```

render_position is where the render running on this thread is: its template's name, the line being rendered, and that line's text; false when none runs. A host asked about a render while it runs, as when the render's allocator asks what to do about memory, names the place with it. The strings borrow the template.

```
jinja.render_source :: proc(env: ^Environment, source:
string, ctx: Value = {}, name := "<template>") -> (string,
Error)
```

render_source parses and renders source in one call (nothing is cached), in a scratch arena it frees before returning. The output, and an error, are allocated with context allocator; the caller owns them.

```
jinja.repr :: proc(v: Value, allocator := context.allocator)
-> string
```

repr converts a value as Python's repr() does, in a string allocated with allocator, which the caller owns.

```
jinja.set :: proc(d: Value, key: string, value: Value)
```

set stores value under the string key in a dict value, growing it with the dict's own allocator; nothing happens when d is not a dict or that allocator refuses.

```
jinja.set_key :: proc(d: Value, key: Value, value: Value)
```

set_key stores value under any key in a dict value.

```
jinja.str :: proc(s: string) -> Value
```

str makes a string value; the value borrows s.

```
jinja.to_string :: proc(v: Value, allocator :=
context.allocator) -> string
```

to_string converts a value as Python's str() does (Undefined gives ""). A string or Markup value is returned as it is, borrowed; anything else is written into a string allocated with allocator, so the result belongs in an arena, as values do.

```
jinja.truthy :: proc(v: Value) -> bool
```

truthy is Python truthiness (Undefined is false).

```
jinja.tuple :: proc(items: ..Value) -> Value
```

tuple makes a tuple holding the given values, allocated as list allocates.

```
jinja.type_name :: proc(v: Value) -> string
```

type_name is the Python type name used in Jinja's error messages.

```
jinja.undefined :: proc(name := "") -> Value
```

undefined makes an undefined value; name, borrowed, is what error messages call it.

```
jinja.values_equal :: proc(a, b: Value) -> bool
```

values_equal compares as Python's == does: lists and dicts by content, 200 levels deep.

odindoc

Package odindoc reads Odin packages by static analysis, with Odin's own parser (core:odin/parser), and writes the reStructuredText of the Guidedoc `odin` domain that documents them, as sphinx.ext.autodoc writes the Python domain's: packages, procedures, procedure groups, types and their fields or enumerators, constants, variables, and foreign imports, each with its doc comment turned into reST.

It never runs the compiler: types are shown as written. A package's files are read the way Odin builds them for every target at once: file-name suffixes (`_windows`, `_linux_arm64`) and `#+build` tags choose the targets a file builds for, and file-scope `when ODIN_OS == ...` blocks are evaluated when their conditions are that simple. A declaration made for every target the package builds for is documented once; one made for some targets only says which (see `platform.odin`).

It is a host library: it reads files through a `File_Reader` and allocates with the context allocator, freeing nothing itself, so a host runs it in an arena: a Project, the packages it loads, and the lines document returns live until the arena is freed.

Types

```
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_entry: 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_permaLinks: 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's getslots`: the `__slots__` the class has (its own, or the nearest base's), as a list, tuple, dictionary, or single string of names. `ok` is `false` when there are none, or they are computed.

```
pyscan.split_lines :: proc(s: string) -> [dynamic]string
```

`split_lines` is `str.splitlines` for `\n`, `\r\n`, `\r` and the other line boundaries Python knows; a final line break does not start another line.

```
pyscan.str_value :: proc(m: ^Module, n: ts.Node) -> (string,
bool)
```

`str_value` is `string_value` for `str` constants only.

```
pyscan.string_value :: proc(m: ^Module, n: ts.Node) ->
(value: string, kind: String_Kind, ok: bool)
```

string_value evaluates a string or concatenated_string node as Python does: prefixes, escape sequences, and adjacent literals joined. ok is false for other nodes.

```
pyscan.text :: proc(m: ^Module, n: ts.Node) -> string
```

text is the source a node spans.

```
pyscan.try_unparse :: proc(m: ^Module, n: ts.Node, mode :=
Unparse_Mode.Sphinx) -> (string, bool)
```

try_unparse is unparse that reports expressions Sphinx’s unparser refuses (comparisons, comprehensions, f-strings, conditional expressions...).

```
pyscan.unparse :: proc(m: ^Module, n: ts.Node, mode :=
Unparse_Mode.Sphinx) -> string
```

unparse writes an expression back as Sphinx’s unparser does (Sphinx mode).

readers/commonmark

Package commonmark reads Markdown into a Guidedoc document without calling an allocator, as two readers with distinct ids: “commonmark” (reader, read, read_inline) is pure CommonMark 0.31.2 and takes no settings; “myst” (myst_reader, read_myst, read_myst_inline) is the MyST Markdown that Sphinx projects write (myst-parser 0.16.1): directives, roles, targets, comments, front matter, footnotes, GFM tables, and the optional syntax that Settings.enable_extensions turns on.

Block structure is parsed line by line into a skeleton in scratch; link reference definitions are collected over the whole document; then the skeleton is emitted in preorder, with inline content parsed per paragraph and heading. Directive content is parsed the same way when the directive asks for it (see extension.odin), so a parse works on line views of any source.

Types

```
commonmark.Directive :: struct
```

Directive is a directive the reader has split into its parts. Lines are views of source.

```
name: string
```

```
name_span: gd.Source_Span
```

span: `gd.Source_Span`

source: `gd.Source_Id`

arg: `[]Line`

the argument; empty when there is none.

options: `[]Option`

content: `[]Line`

sections: `bool`

the directive stands where sections are allowed.

classes: `gd.Text`

classes a handler leaves for the next element (Docutils' class directive without content); the reader gives them to the next block it emits.

`commonmark.Directive_Handler` *:: struct*

name: `string`

compared case-insensitively.

run: `Directive_Proc`

`commonmark.Directive_Proc` *:: #type proc(h: ^Host, d: ^Directive, user: rawptr) -> Directive_Result*

`Directive_Proc` runs a parsed directive; `user` is `Extension.user`. The directive and its lines are borrowed for the call.

`commonmark.Directive_Result` *:: enum u8*

Ran

Unknown

the handler does not know the name; the reader reports it.

Failed

the handler refused the directive and reported why.

```
commonmark.Directive_Shape :: struct
```

Directive_Shape tells the reader how a directive takes its first line: as an argument, or, for a directive without arguments, as the first line of content; and whether it takes options at all.

```
argument: bool
```

```
options: bool
```

it has options; without, an option block stays content.

```
commonmark.Extension :: struct
```

```
directives: []Directive_Handler
```

```
directive: Directive_Proc
```

directive is offered every directive the table does not list.

```
shape: proc(h: ^Host, name: string, user: rawptr) ->
(Directive_Shape, bool)
```

shape describes a directive by name; found is false for an unknown directive.

```
roles: []Role_Handler
```

```
role: Role_Proc
```

role is offered every role the table does not list.

```
table: proc(h: ^Host, id: gd.Node_Id, span:
gd.Source_Span, columns: int, user: rawptr) -> bool
```

table runs after the reader opens a Table node, before its rows, so a table directive around it can add its title and attributes; it returns true when it did, and the reader then adds no attributes of its own.

```
begin: proc(h: ^Host, user: rawptr)
```

before the document's blocks.

```
end: proc(h: ^Host, user: rawptr)
```

after them.

resolve: `proc(h: ^Host, user: rawptr) -> gd.Status`

resolve, when set, links references, targets, and footnotes instead of the reader's own resolution (resolve.odin); the Sphinx layer resolves as for reStructuredText.

user: `rawptr`

`commonmark.Host` *:: struct*

Host is the reader's document-wide state, shared by the parse of the document and the parses of nested content. Directive and role handlers receive it and treat it as opaque: they reach the read through context_of and the procedures below.

ctx: `^gd.Read_Context`

b: `^gd.Builder`

settings: `Settings`

ext: `^Extension`

myst: `bool`

the MyST syntax is on: the "myst" reader, not "commonmark".

disabled: `Rules`

myst_disable_syntax.

syntax: `Syntax_Extensions`

myst_enable_extensions in force.

refs: `[]Ref`

Link reference definitions, collected over the whole document.

nrefs: `int`

names: `gd.Symbol_Table`

labels: `gd.Buffer`


```
label_tmp: []u8
```

```
html_format: gd.Text
```

```
rel_images: bool
```

MyST include options in force: rewrite image paths, and link paths starting with `rel_docs`, of included sources to be relative to the main source.

```
rel_docs: string
```

```
using state: Myst_State
```

```
commonmark.Line :: gd.Line_View
```

Line is a view of one input line: its text without the line break, and the source offset of `text[o]`. Directive handlers pass nested content as lines.

```
commonmark.Meta :: struct
```

Meta is one HTML meta tag: key as MyST writes it (“description lang=en”, “http-equiv=Content-Type”, “keywords”) and its content.

```
key: string
```

```
content: string
```

```
commonmark.Option :: struct
```

Option is one directive option: its name and value lines. A value written as a YAML scalar has its quoting and folding processed; its lines may then be copies in scratch, whose offset is where the value starts in the source.

```
name: string
```

```
value: []Line
```

```
span: gd.Source_Span
```

```
commonmark.Role :: struct
```

Role is one role, `{name}`content``. lines hold the whole construct, split at line breaks; content is the range of the content in the lines joined by “\n”.

```
name: string
```

```
lines: []Line
```

```
content: [2]int
```

```
span: gd.Source_Span
```

```
source: gd.Source_Id
```

```
commonmark.Role_Handler :: struct
```

```
name: string
```

compared case-insensitively.

```
run: Role_Proc
```

```
commonmark.Role_Proc :: #type proc(h: ^Host, r: ^Role, user: rawptr) -> bool
```

Role_Proc handles a role; it returns false for a role it does not know.

```
commonmark.Rule :: enum u8
```

Rule names a syntax element that `myst_disable_syntax` can turn off, by markdown-it's rule name.

```
Table
```

```
Code
```

```
Fence
```

```
Blockquote
```

```
Hr
```

```
List
```

```
Reference
```

Html_Block

Heading

Lheading

Emphasis

Backticks

Link

Image

Autolink

Html_Inline

Entity

Escape

Newline

Front_Matter

Myst_Target

Myst_Line_Comment

Myst_Block_Break

Myst_Role

Math_Block

Math_Inline

Footnote_Def**Footnote_Ref****Myst_Directive**

not a markdown-it rule: {name} fences (off with `commonmark_only`).

```
commonmark.Settings :: struct
```

Settings are the “myst” reader’s settings: myst-parser 0.16.1’s `myst_*` configuration values, named without the prefix. They reach the reader through `gd.Read_Options.extension`, made with `gd.read_extension(&settings)`; an empty extension means `default_settings()`. The “commonmark” reader (pure CommonMark) takes none. A caller that supplies Settings starts from `default_settings()`, because several defaults are not zero.

```
extension: ^Extension
```

Directive and role handlers (the Sphinx layer provides them); nil means none, and directives and roles are then reported as unknown.

```
commonmark_only: bool
```

`myst_commonmark_only`: CommonMark syntax only (no directives, roles, or other MyST syntax), but a document’s tree as for MyST: headings make sections.

```
enable_extensions: Syntax_Extensions
```

`myst_enable_extensions`: the optional syntax. Default `{.Dollarmath}`.

```
disable_syntax: []string
```

`myst_disable_syntax`: markdown-it rule names whose syntax is not parsed, e.g. “emphasis”, “table”, “html_block” (see `syntax_rule`).

```
url_schemes: []string
```

`myst_url_schemes`: schemes of `[text](scheme:...)` links that are external; other destinations are cross-references. nil recognizes every scheme. Default `http, https, mailto, and ftp`.

```
linkify_fuzzy_links: bool
```

`myst_linkify_fuzzy_links`: linkify recognizes URLs without a scheme. Default `true`.

heading_anchors: int

myst_heading_anchors: headings up to this level get a GitHub-style slug anchor that `[](#slug)` and `[](file.md#slug)` reach. 0 means none.

heading_slug_func: string

myst_heading_slug_func is a Python callable and cannot be supported; a non-empty value is reported and the built-in slugs are used.

substitutions: []Substitution

myst_substitutions: global substitutions; front matter overrides them.

sub_delimiters: [2]u8

myst_sub_delimiters: the characters doubled around substitution references. Default `{‘, ’}`, giving `{{ name }}`.

html_meta: []Meta

myst_html_meta: HTML meta tags for every document; front matter `html_meta` overrides entries with the same key.

footnote_transition: bool

myst_footnote_transition: a transition of class “footnotes” precedes the footnotes at the end of the document. Default true.

dmath_allow_labels: bool

myst_dmath_allow_labels: `$$...$$ (label)`. Default true.

dmath_allow_space: bool

myst_dmath_allow_space: inline math may start or end with a space. Default true.

dmath_allow_digits: bool

myst_dmath_allow_digits: inline math may follow or precede a digit. Default true.

dmath_double_inline: bool

myst_dmath_double_inline: `$$...$$` inside a paragraph. Default false.

words_per_minute: int

myst_words_per_minute: for the wordcount-minutes substitution. Default 200.

docname: string

The document’s name (“usage/intro”) and configuration values, which substitutions reach as env.docname and env.config.<name>.

env_config: []Substitution

today: string

The {sub-ref} today: the project’s formatted date; empty is Read_Options.today.

commonmark.**Substitution** :: struct

A substitution’s value is Markdown, or a mapping written as nested YAML that dotted names reach (“key.sub”).

name: string

value: string

commonmark.**Syntax_Extension** :: enum u8

Syntax_Extension names an entry of myst_enable_extensions.

Amsmath

Colon_Fence

Deflist

Dollarmath

Fieldlist

Html_Admonition

Html_Image

Linkify**Replacements****Smartquotes****Substitution****Tasklist**

```
commonmark.Syntax_Extensions :: bit_set[Syntax_Extension;
u16]
```

Procedures

```
commonmark.context_of :: proc(h: ^Host) -> ^gd.Read_Context
```

context_of returns the read context, for handlers that build nodes.

```
commonmark.default_settings :: proc() -> Settings
```

default_settings returns myst-parser 0.16.1's defaults. The value borrows only static data (the default URL schemes); fields a caller sets must outlive the read.

```
commonmark.include_paths :: proc(h: ^Host, images: bool,
docs: string) -> (bool, string)
```

include_paths sets MyST's include options :relative-images: and :relative-docs: for the content of the include directive being run; it returns the previous values.

```
commonmark.list_items :: proc(h: ^Host, lines: []Line,
source: gd.Source_Id) -> (items: [][]Line, starts: []int,
ok: bool)
```

list_items splits lines that hold one bullet list into the content lines of its items, and where each item starts in lines (for list-table). ok is false when the lines are anything but one bullet list. The results live in scratch.

```
commonmark.myst_reader :: proc() -> gd.Reader
```

myst_reader returns the “myst” reader by value: MyST Markdown with ^Settings.

```
commonmark.parse_body :: proc(h: ^Host, lines: []Line,
source: gd.Source_Id, sections := false, classes :=
gd.Text{})
```

`parse_body` parses lines of a source as Markdown body content of the builder's open node. `sections` lets headings open sections, as at the top of a document; `classes`, when set, go to every top-level element.

```
commonmark.parse_inline :: proc(h: ^Host, lines: []Line,
source: gd.Source_Id)
```

`parse_inline` parses lines of a source as Markdown inline content of the builder's open node.

```
commonmark.read :: proc(ctx: ^gd.Read_Context) -> gd.Status
```

`read` reads `ctx.source` as pure CommonMark 0.31.2 (`gd.Read_Proc`): headings, no sections. Any settings in the read options are refused with `Invalid_Input`. It borrows `ctx` for the call; its skeleton lives in scratch, released before it returns. Problems go to `ctx.reports`; the result is the builder's status (`.Capacity` when a table is full).

```
commonmark.read_inline :: proc(ctx: ^gd.Read_Context, span:
gd.Source_Span) -> gd.Status
```

`read_inline` parses a span of `ctx.source` as CommonMark inline content of the builder's open node, for a translated message (`gd.Read_Inline_Proc`). Link reference definitions and footnote definitions of the surrounding document are not known to it.

```
commonmark.read_myst :: proc(ctx: ^gd.Read_Context) ->
gd.Status
```

`read_myst` reads `ctx.source` as MyST Markdown (`gd.Read_Proc`), with the `^Settings` the read options carry (`gd.read_extension`), or `default_settings` when they carry none. Settings of another type are refused with `Invalid_Input`. Ownership and failures are `read`'s; the settings are borrowed for the call.

```
commonmark.read_myst_inline :: proc(ctx: ^gd.Read_Context,
span: gd.Source_Span) -> gd.Status
```

`read_myst_inline` is `read_inline` for MyST, with the settings of `ctx.options.extension` (roles included), as `read_myst` takes them.

```
commonmark.reader :: proc() -> gd.Reader
```

`reader` returns the “commonmark” reader by value: pure CommonMark 0.31.2, no settings.


```
commonmark.syntax_extension_named :: proc(name: string) ->
(Syntax_Extension, bool)
```

syntax_extension_named finds a myst_enable_extensions entry by its name.

```
commonmark.syntax_rule :: proc(name: string) -> (Rule, bool)
```

syntax_rule finds a myst_disable_syntax entry by its markdown-it rule name.

Constants

```
commonmark.CONFORMANCE :: gd.Conformance.Conformant
```

```
commonmark.MYST_CONFORMANCE :: gd.Conformance.Partial
```

```
commonmark.MYST_VERSION :: "0.16.1"
```

MyST follows myst-parser 0.16.1, with the differences CONFORMANCE.md documents.

```
commonmark.VERSION :: "0.31.2"
```

readers/rst

Package rst reads reStructuredText as the Docutils markup specification defines it, implemented independently. It owns lines, block structure, and directives; inline markup and reference resolution live in the inline subpackage.

Types

```
rst.Arg_Rule :: enum u8
```

None

Optional

Required

```
rst.Content_Rule :: enum u8
```

None

Optional

Required

```
rst.Dir_Opt :: struct
```

```
spec: Opt_Spec
```

```
lines: []Line
```

the value: the rest of the marker line, then continuation lines.

```
span: gd.Source_Span
```

```
rst.Dir_Spec :: struct
```

```
name: string
```

```
kind: Dir_Kind
```

```
arg: Arg_Rule
```

```
final_ws: bool
```

```
content: Content_Rule
```

```
opts: Opt_Set
```

```
value: gd.Admonition_Kind
```

```
rst.Directive :: struct
```

```
spec: ^Dir_Spec
```

```
span: gd.Source_Span
```

```
name_span: gd.Source_Span
```

```
arg: []Line
```

argument lines, trimmed; empty when there is no argument.

```
opts: [MAX_OPTIONS]Dir_Opt
```

```
nopts: int
```

content: []Line

sections: bool

subst: bool

inside a substitution definition.

ext: ^Directive_Handler

a dialect's handler, or nil for a standard directive.

arg_block: []Line

the argument lines with their relative indentation.

subst_name: string

the name of the substitution definition it is in, as written.

rst.Directive_Handler :: struct

name: string

compared case-insensitively, as directive names are.

arg: Arg_Rule

final_ws: bool

the last argument may contain white space.

content: Content_Rule

options: []Opt_Spec

run: Directive_Proc

rst.Directive_Proc :: #type proc(p: ^Parser, d: ^Directive, user: rawptr)

Directive_Proc runs a parsed directive. user is Extension.user.

rst.Directive_Shape :: struct

Directive_Shape tells a host how a directive takes its parts.

arg: Arg_Rule

final_ws: bool

the last argument may contain white space.

content: Content_Rule

options: []Opt_Spec

rst.Extension :: struct

directives: []Directive_Handler

lookup: proc(name: string, user: rawptr) ->
^Directive_Handler

lookup, when set, finds handlers the table does not list, such as names that depend on reader state (a default domain). It returns nil for unknown names.

roles: []Role_Handler

role: Role_Proc

role, when set, is offered every role the table does not list.

begin: Hook_Proc

before the main source's blocks (reader state).

prologue: Hook_Proc

prologue runs after the main source's leading field lines, where Sphinx inserts rst_prolog: after a document's bibliographic fields, else before everything.

end: Hook_Proc

after them, before sections close (e.g. an epilogue).

substitution: proc(name: string, user: rawptr) -> (text:
string, ok: bool)

substitution, when set, supplies default substitutions: resolution offers it each reference whose name no definition in the document has (see

`rst_inline.Substitution_Hook`). It runs after parsing, with the extension's user.

```
user: rawptr
```

```
rst.Frame :: struct
```

```
src: gd.Source_Id
```

```
text: string
```

```
tab: int
```

```
rst.Hook_Proc :: #type proc(p: ^Parser, user: rawptr)
```

`Hook_Proc` runs at a fixed point of the read, with the reader's state.

```
rst.Host :: struct
```

Host procedures receive `Host.user`. Lines are views of the current source (`p.src`), as the block parser's are, so every node keeps exact spans.

```
body: proc(p: ^Parser, lines: []Line, sections: bool,  
each: gd.Text, user: rawptr)
```

`body` parses nested body content of the open node; `sections` allows headings to make sections, and `each`, when set, gives these classes to every top-level element (the class directive).

```
inline: proc(p: ^Parser, lines: []Line, user: rawptr)
```

`inline` parses inline content of the open node.

```
items: proc(p: ^Parser, lines: []Line, user: rawptr) ->  
(items: [][]Line, starts: []int, ok: bool)
```

`items` splits lines that hold one bullet list into its items' content, with where each item starts in lines (list-table, with the host's list syntax); `ok` is false when the lines are not one bullet list.

```
user: rawptr
```

```
rst.Host_Directive :: struct
```

`Host_Directive` is a directive the host has split into its parts.

name: string

name_span: gd.Source_Span

span: gd.Source_Span

arg: []Line

the argument lines; empty when there is none.

options: []Host_Option

content: []Line

sections: bool

the directive stands where sections are allowed.

rst.Host_Option :: struct

Host_Option is a directive option as the host wrote it: its name and value lines.

name: string

value: []Line

span: gd.Source_Span

rst.Hosted_Result :: enum u8

Ran

Unknown

no directive of that name; nothing was reported.

Failed

the arguments, options, or content were refused and reported.

rst.Line :: struct

Line is a view of one input line in columns: tabs are expanded to tab stops and trailing white space is stripped. A line that held a tab, form feed, or vertical tab is a copy in scratch; raw then keeps the source line so columns map back

to source bytes. Nested blocks use cut views of their parent's lines, so every node keeps exact spans.

text: string

raw: string

the source line of an expanded copy, else "".

offset: u32

source offset of text[o]; of raw[o] when raw is set.

col: i32

column of text[o] in the expanded line when raw is set.

rst.Opt_Spec :: struct

name: string

type: Opt_Type

choices: string

space-separated, for Choice.

rst.Opt_Type :: enum u8

Flag

Text

Text_Required

Int

Nonneg_Int

Pos_Int

Opt_Int

Length**Length_Percent****Figwidth****Percentage****Class****Name****Choice****Encoding****Char****Widths****Path****rst.Parser** *:: struct***ctx:** ^gd.Read_Context**b:** ^gd.Builder**inl:** rst_inline.Inline_State**src:** gd.Source_Id

the source the current lines come from.

text: string**tab:** int**depth:** int

stack: `gd.Stack_Guard`

bounds the machine stack of the recursion (enter).

styles: `[MAX_LEVELS]Style`

styles_used: `int`

sections: `[MAX_LEVELS]Section_Entry`

sections_used: `int`

section_base: `int`

sections open before titled content (`parse_titled`) began.

level_base: `int`

the level of the last of them: titled content's sections nest in it.

top_count: `int`

top-level elements since the current section began.

top_transition: `bool`

the last top-level element was a transition.

transition_span: `gd.Source_Span`

pending: `gd.Text`

classes from a class directive, for the next element.

pending_span: `gd.Source_Span`

has_pending: `bool`

in_sidebar: `int`

ext: `^Extension`

a dialect's directives and roles; see `extension.odin`.

host: `^Host`

another markup that uses the directives; see `host.odin`.

hosted: `bool`

the running directive was written in the host markup.

table_extra: `^Table_Extra`

a table directive's title and options for the host.

`rst.Role_Handler` `:: struct`

name: `string`

compared case-insensitively.

run: `Role_Proc`

`rst.Role_Proc` `:: #type proc(p: ^Parser, call: ^rst_inline.Role_Call, user: rawptr) -> bool`

`Role_Proc` handles interpreted text; it returns false for a role it does not know.

`rst.Settings` `:: struct`

Settings are the reader's dialect settings. They reach the reader through `gd.Read_Options.extension`, made with `gd.read_extension(&settings)`; settings of another type are refused. A dialect's own configuration embeds `Settings` and passes a pointer to that embedded field when it runs this reader. An empty extension means defaults (`default_settings`); a caller that supplies `Settings` starts from `default_settings`, because the zero value turns bibliographic fields off.

extension: `^Extension`

Directive and role handlers consulted before the standard ones.

host: `^Host`

Another markup (MyST Markdown) that reads the main source and nested directive content with these directives and roles; see `host.odin`.

pep_references: `bool`

Implicit "PEP nnn" and "RFC nnn" references in running text become links.

rfc_references: bool

pep_base_url: string

“” means <https://peps.python.org/>

rfc_base_url: string

“” means <https://datatracker.ietf.org/doc/html/>

trim_footnote_reference_space: bool

White space before a footnote reference is removed.

smart_quotes: Smart_Quotes

Typographic quotation marks, apostrophes, dashes, and ellipses in text.

smart_quotes_locales: string

Quotation marks per language, overriding the built-in ones, as “tag:“””,tag2:«»<>”: primary open and close, then secondary open and close.

language: string

BCP 47 tag of the document language: bibliographic field names, generated titles, and quotation marks. “” means “en”.

docinfo: bool

A field list first in the document becomes bibliographic data (docinfo).

sectsubtitle: bool

A lone subsection title becomes the section’s subtitle.

tab_width: int

Tab stops; zero means `gd.Read_Options.tab_width`, which zero makes 8.

character_level_inline_markup: bool

Inline markup is recognized inside words (Docutils’ `character_level_inline_markup`).

`rst.Smart_Quotes :: rst_inline.Smart_Quotes`

```
rst.Table_Extra :: struct
```

Table_Extra carries what table directives add to a table: a title and attributes.

```
title: []Line
```

```
classes: gd.Text
```

```
name: gd.Text
```

```
align: string
```

```
width: string
```

```
widths: []int
```

from :widths:: nil keeps the table's own widths.

```
auto_widths: bool
```

:widths: auto leaves widths to the renderer.

```
width_class: string
```

“colwidths-auto” or “colwidths-given” from :widths:.

```
found: bool
```

set by the table parser when it emits a table.

Procedures

```
rst.add_common :: proc(p: ^Parser, id: gd.Node_Id, d:  
^Directive, class_opt := "class")
```

add_common adds the :class: and :name: options as attributes. Directive nodes keep their Name attribute for the directive name, so :name: becomes Custom “name”.

```
rst.blank :: proc(l: Line) -> bool
```

blank tells whether l has no text (white space is stripped when lines are split).

```
rst.bullet_width :: proc(t: string) -> int
```

bullet_width returns the byte length of a bullet that starts t, or 0.

```
rst.class_directive :: proc(p: ^Parser, d: ^Directive)
```

class_directive runs the class directive: its classes go to the content's elements, or, without content, to the next element; invalid names are reported.

```
rst.close :: proc(p: ^Parser, id: gd.Node_Id)
```

close ends the element open returned id for.

```
rst.code_text :: proc(p: ^Parser, lines: []Line) -> gd.Text
```

code_text writes lines to the text pool, each ending in “\n”.

```
rst.custom :: proc(p: ^Parser, id: gd.Node_Id, name, value: string)
```

custom adds a Custom attribute name=value to node id; both strings are copied to the text pool. A full pool poisons the builder, as every builder write does.

```
rst.cut :: proc(l: Line, k: int) -> Line
```

cut drops the first k columns.

```
rst.dedent :: proc(lines: []Line)
```

dedent strips the least indentation of the non-blank lines, in place.

```
rst.default_settings :: proc() -> Settings
```

default_settings returns Docutils' defaults: bibliographic fields on, everything else off or empty. The value owns nothing.

```
rst.directive_node :: proc(p: ^Parser, d: ^Directive, name: string) -> gd.Node_Id
```

directive_node opens a generic Directive node named after the directive.

```
rst.directive_shape :: proc(p: ^Parser, name: string) -> (shape: Directive_Shape, found: bool)
```

directive_shape looks a directive up as the reader would, a dialect's handlers first.

```
rst.embed_begin :: proc(p: ^Parser, ctx: ^gd.Read_Context)
```

embed_begin prepares p for a host that reads the main source itself: the settings of ctx.options.extension apply as for read, and the dialect's begin hook runs.

```
rst.embed_end :: proc(p: ^Parser) -> gd.Status
```

`embed_end` runs the dialect's end hook and resolves the document as `read` does.

```
rst.enter_source :: proc(p: ^Parser, id: gd.Source_Id, tab: int) -> Frame
```

`enter_source` makes lines of another source current, for spans and inline parsing.

```
rst.find_role :: proc(p: ^Parser, name: string) -> int
```

`find_role` returns the index of the role the document defined with the role directive under `name`, in `p.inl.roles`, or -1. The search spends work budget.

```
rst.get_opt :: proc(d: ^Directive, name: string) -> ^Dir_Opt
```

`get_opt` returns the option called `name` that the directive was given, or nil. The result points into `d` and lives as long as it.

```
rst.has_option :: proc(d: ^Directive, name: string) -> bool
```

`has_option` reports whether a directive was given an option, flags included.

```
rst.head :: proc(l: Line, n: int) -> Line
```

`head` keeps the first `n` columns, without trailing spaces.

```
rst.host_lines :: proc(p: ^Parser, views: []rst_inline.Line) -> ([]Line, bool)
```

`host_lines` makes lines of views of source text (a host's lines): tabs are expanded to the parser's tab stops, as `split_lines` does.

```
rst.host_views :: proc(p: ^Parser, lines: []Line) -> ([]rst_inline.Line, bool)
```

`host_views` gives the host the source text of lines, tabs unexpanded (`inline_view`).

```
rst.indent_of :: proc(s: string) -> int
```

`indent_of` counts the spaces `s` starts with.

```
rst.inline :: proc(p: ^Parser, lines: []Line)
```

`inline` parses lines as inline content of the currently open node.

```
rst.inline_node :: proc(p: ^Parser, kind: gd.Node_Kind,
lines: []Line)
```

inline_node adds a node of kind holding lines parsed as inline content (inline); nothing for no lines.

```
rst.is_indented :: proc(l: Line) -> bool
```

is_indented tells whether l starts with a space.

```
rst.join_lines :: proc(p: ^Parser, lines: []Line, sep:
string) -> string
```

join_lines joins lines into a scratch copy, or returns the single line's text.

```
rst.leave_source :: proc(p: ^Parser, f: Frame)
```

leave_source makes the source that enter_source returned f for current again.

```
rst.line_span :: proc(p: ^Parser, l: Line) -> gd.Source_Span
```

line_span is the source span of one line (span_of).

```
rst.lines_span :: proc(p: ^Parser, lines: []Line) ->
gd.Source_Span
```

lines_span is the source span of lines without trailing blank lines; empty lines give an empty span at the start of the source.

```
rst.mark :: proc(p: ^Parser) -> gd.Scratch_Mark
```

mark records the scratch position; release returns scratch to it. Pair them with defer around work that takes scratch.

```
rst.open :: proc(p: ^Parser, kind: gd.Node_Kind, span:
gd.Source_Span) -> gd.Node_Id
```

open begins an element and gives it the classes of a preceding class directive.

```
rst.opt_value :: proc(p: ^Parser, d: ^Directive, name:
string) -> (string, bool)
```

opt_value returns an option's value lines joined by spaces and trimmed; ok is false when the option was not given. The text is a view of the source or a scratch copy, valid until the caller's scratch mark is released.

```
rst.option_lines :: proc(d: ^Directive, name: string) ->
([]Line, bool)
```

option_lines returns the value lines of a directive option, or nil.

```
rst.parse_blocks :: proc(p: ^Parser, lines: []Line,
sections := false, each := gd.Text{})
```

`parse_blocks` parses body elements. `sections` allows titles and transitions; each gives classes to every element parsed at this level (class directive content).

```
rst.parse_body :: proc(p: ^Parser, lines: []Line, sections :=
false, each := gd.Text{})
```

`parse_body` parses nested body content of the open node in the markup of the running directive (see `Host`). Outside hosted directives it is `parse_blocks`.

```
rst.parse_source :: proc(p: ^Parser, id: gd.Source_Id,
sections: bool)
```

`parse_source` parses a whole registered source (`gd.add_source`) as body content at the current position, as if its lines stood there. `sections` allows titles.

```
rst.parse_titled :: proc(p: ^Parser, lines: []Line)
```

`parse_titled` parses content in which titles begin sections, in a title context of its own (Sphinx's nested parse with titles): adornments take levels afresh, the first nesting in the innermost open section, and the sections close when the content ends. The surrounding adornments are kept for what follows.

```
rst.path_of :: proc(p: ^Parser, lines: []Line) -> string
```

`path_of` joins path lines without white space, as the specification treats paths.

```
rst.place_titled :: proc(p: ^Parser, lines: []Line)
```

`place_titled` prepares for titled content (`parse_titled`) as Sphinx's only directive does: when the content's first title has an adornment the document already uses, the content stands where a title of that level would, so deeper sections close first.

```
rst.pool_classes :: proc(p: ^Parser, s: string) -> (gd.Text,
bool)
```

`pool_classes` writes white-space separated class names as identifiers, one space apart. It fails when a name has no letters.

```
rst.pool_lines :: proc(p: ^Parser, lines: []Line, sep:
string) -> gd.Text
```

`pool_lines` writes the lines joined by `sep` into the text pool and returns the text, which lives as long as the document; a full pool poisons the builder.

```
rst.pool_lower :: proc(p: ^Parser, s: string) -> string
```

pool_lower writes s in lower case to the text pool and returns the pooled string.

```
rst.push_lines :: proc(p: ^Parser, lines: []Line, sep: string)
```

push_lines writes the lines joined by sep into the text pool.

```
rst.read :: proc(ctx: ^gd.Read_Context) -> gd.Status
```

read reads ctx.source as reStructuredText into ctx.b (gd.Read_Proc). Settings of another type than ^Settings are refused with Invalid_Input before anything is read. It borrows ctx for the call and keeps nothing; its tables live in scratch, released before it returns. Problems go to ctx.reports; the result is the builder's status (.Capacity when a table is full, .Limit past the nesting or stack limits), or that of reference resolution.

```
rst.read_inline :: proc(ctx: ^gd.Read_Context, span: gd.Source_Span) -> gd.Status
```

read_inline parses a span of ctx.source as inline content of the builder's open node, with the settings of ctx.options.extension, as Sphinx parses a translated message (gd.Read_Inline_Proc). Each line's leading white space is dropped, as the lines of a paragraph lose their indentation. References are left for the caller to resolve.

```
rst.reader :: proc() -> gd.Reader
```

reader returns the "rst" reader descriptor, for a gd.Registry. It is a value with no state: registering it twice or copying it is harmless.

```
rst.release :: proc(p: ^Parser, m: gd.Scratch_Mark)
```

release frees the scratch taken since mark m; views into it become invalid.

```
rst.report :: proc(p: ^Parser, m: gd.Message, span: gd.Source_Span, a := gd.Diagnostic_Arg{}, b := gd.Diagnostic_Arg{})
```

report adds diagnostic m at span, with its arguments, to the read's reports; a full reports region counts it as suppressed (gd.report) rather than failing the read.

```
rst.run_hosted :: proc(p: ^Parser, h: Host_Directive) ->
Hosted_Result
```

run_hosted checks and runs a directive written in the host markup. Options are checked as in reStructuredText; a host with other rules checks them first (directive_shape, option_problem) and passes only the ones it keeps.

```
rst.span_of :: proc(p: ^Parser, first, last: Line) ->
gd.Source_Span
```

span_of is the source span from the start of first to the end of last, in the current source, clamped to its text.

```
rst.standard_role :: proc(name: string) ->
(rst_inline.Role_Base, bool)
```

standard_role finds a standard role or alias by name, compared case-insensitively; ok is false (base .Custom) for another name.

```
rst.table_decorate :: proc(p: ^Parser, id: gd.Node_Id,
span: gd.Source_Span, cols: int, widths: []int, extra:
^Table_Extra)
```

table_decorate gives an open Table, before its rows, its widths and a table directive's attributes and title. A host calls it for the table it emits inside a table directive (Parser.table_extra).

```
rst.take :: proc(p: ^Parser, $T: typeid, n: int) -> ([]T,
bool)
```

take returns n zeroed items of T from scratch, valid until the enclosing mark is released. When scratch is short it records the capacity need on the builder (poisoning it) and returns false.

```
rst.trim_blank_end :: proc(lines: []Line) -> []Line
```

trim_blank_end returns lines without its trailing blank lines (a subslice).

Constants

```
rst.CONFORMANCE :: gd.Conformance.Partial
```

```
rst.DIRECTIVE_UNKNOWN
```

DIRECTIVE_UNKNOWN is what the reader reports for a directive no handler knows; a handler that finds it cannot serve a directive it was given reports it too.

```
rst.VERSION :: "0.1.0"
```

readers/rst/inline

Package `rst_inline` parses reStructuredText inline markup and resolves references. The block reader in `lib/readers/rst` owns lines, directives, and document structure; it calls `parse_inline` for every inline context and resolve once parsing ends.

Types

```
inline.Inline_State :: struct
```

`Inline_State` is shared, long-lived reader state for inline parsing.

```
ctx: ^gd.Read_Context
```

```
source: gd.Source_Id
```

the source the current lines come from.

```
roles: []Role
```

custom roles; storage owned by the block reader.

```
roles_used: int
```

```
default_role: string
```

“” means title-reference.

```
hook: Role_Hook
```

a dialect's roles, consulted first; see `extension.odin`.

```
options: Options
```

settings of the block reader; see `options.odin`.

```
inline.Line :: gd.Line_View
```

`Line` is a view of one line of text. `offset` is the source byte offset of `text[o]`; for a line copied during tab expansion it is the offset of the original line start.

```
inline.Options :: struct
```

`Options` are the settings of the block reader (`rst.Settings`) that inline parsing and resolution need.

```
pep_references: bool
```

rfc_references: bool

pep_base_url: string

rfc_base_url: string

trim_footnote_reference_space: bool

smart_quotes: Smart_Quotes

smart_quotes_locales: string

language: string

docinfo: bool

sectsubtitle: bool

character_level: bool

tab_width: int

tab stops of the main source; 0 means 8.

default_substitution: Substitution_Hook

inline.Role :: struct

Role is a role defined by the role directive, or a standard role.

name: string

normalized (lowercase) role name.

base: Role_Base

classes: string

space-separated, from :class:.

format: string

raw roles: the output format, e.g. "html".

language: string

code-derived roles.

`inline.Role_Base :: enum u8`

Emphasis

Strong

Literal

Code

Math

Subscript

Superscript

Title_Reference

Pep_Reference

Rfc_Reference

Abbreviation

Acronym

Raw

Custom

a Role node carrying the role name and classes.

`inline.Role_Call :: struct`

Role_Call describes one interpreted text for an extension’s role handler. The strings live in scratch and are valid only during the call.

state: ^Inline_State

b: `^gd.Builder`

name: `string`

the role name as written, or the default role's name.

marked: `string`

the content with every escaping backslash as NUL; breaks as “\n”.

text: `string`

the content with escapes processed; line breaks become spaces.

span: `gd.Source_Span`

the whole interpreted text, role included.

content: `gd.Source_Span`

the content between the backquotes.

inline.Role_Hook *:: struct*

Role_Hook lets a dialect handle interpreted text before the document's and the standard roles are consulted. The procedure returns false for a role it does not know, and the standard lookup continues.

procedure: `proc(user: rawptr, call: ^Role_Call) -> bool`

user: `rawptr`

inline.Smart_Quotes *:: enum u8*

Smart_Quotes selects typographic quotation marks, apostrophes, dashes, and ellipses.

Off

On

the language's primary quotation marks outside, secondary inside.

Alt

the language's alternative marks, where it has them.

```
inline.Substitution_Hook :: struct
```

Substitution_Hook supplies substitutions a dialect defines by default, such as Sphinx’s `|version|`, `|release|`, and `|today|`. Resolution offers it every reference whose name no definition in the document has exactly (the case-insensitive match comes after it); a procedure returning true replaces the reference with that text.

```
procedure: proc(user: rawptr, name: string) -> (text:
string, ok: bool)
```

```
user: rawptr
```

Procedures

```
inline.apply_host_role :: proc(s: ^Inline_State, name:
string, lines: []Line, q, j: int) -> gd.Status
```

`apply_host_role` runs the role called `name` on interpreted text written in another markup, such as MyST’s `{name}`content``, as the dialect’s hook and the standard roles would for `reStructuredText`. `lines` hold the whole construct; the content is the joined text’s range `[q, j]` (lines joined by “\n”) and is taken literally: a backslash is an ordinary character. An unknown role is reported, and the construct stays as text.

```
inline.footnote_symbol :: proc(n: int, out: ^gd.Buffer)
```

`footnote_symbol` writes the label of symbol footnote `n` (1-based) into `out`; nothing for `n < 1`. A label that does not fit sets `out`’s overflow, as every `Buffer` write does.

```
inline.parse_inline :: proc(s: ^Inline_State, lines: []Line)
-> gd.Status
```

`parse_inline` emits inline nodes for the lines as children of the builder’s currently open node. Markup may span lines; a line break outside inline literals and roles becomes a `Soft_Break` node, as in `CommonMark`. Markup that cannot be completed is kept as text with a warning; unknown or misused roles are errors. It borrows `s` and `lines` for the call; its working memory is scratch it releases. Problems go to the reports; the result is the builder’s status, which a full table or spent work budget sets.

```
inline.python_space :: proc(r: rune) -> bool
```

`python_space` reports whether `r` is white space as Python’s `str.isspace` and the `\s` of Python’s regular expressions define it, for readers that follow Python patterns.

```
inline.resolve :: proc(ctx: ^gd.Read_Context, options :=
Options{docinfo = true}) -> gd.Status
```

resolve links references to targets, numbers footnotes, checks substitutions, and promotes the document title when ctx.options.promote_title is set. It edits the builder's nodes in place, with scratch it releases; problems go to ctx.reports; the result is the builder's status.

```
inline.unescape :: proc(out: ^gd.Buffer, marked: string)
```

unescape writes marked text (escapes as NUL, see Role_Call.marked) with escapes processed: an escaped space or line break disappears, any other escaped character stays, and an unescaped line break becomes a space.

readers/sphinx

Package sphinx reads reStructuredText with the Sphinx layer: Sphinx's directives, roles, and domains (GDS "Guidedog: everything Sphinx does, the Odin way"), implemented independently on the base reader's extension interface. It produces the shapes of lib/core/AST.md, "Sphinx layer"; cross-document work (toctrees, cross-references, numbering) is left to the project host. The same handlers serve Markdown (MyST) read by the commonmark reader: see markdown.odin.

Types

```
sphinx.Autodoc_Message :: struct
```

Autodoc_Message is a problem the host found, reported at the directive.

```
severity: gd.Severity
```

```
code: string
```

```
title: string
```

```
message: string
```

```
hint: string
```

```
sphinx.Autodoc_Option :: struct
```

```
name: string
```

```
value: string
```

“” for a flag.


```
sphinx.Autodoc_Provider :: struct
```

Autodoc_Provider belongs to the caller’s trust boundary and may allocate.

```
procedure: proc(user: rawptr, request: ^Autodoc_Request) -
> Autodoc_Result
```

```
user: rawptr
```

```
sphinx.Autodoc_Request :: struct
```

Autodoc_Request is one autodoc directive and the context it needs: the current py:module and py:class (ref_context) and the document.

```
directive: string
```

“autoclass”

```
argument: string
```

```
options: []Autodoc_Option
```

```
content: string
```

the directive’s content, lines joined with “\n”.

```
module: string
```

```
class: string
```

```
docname: string
```

```
sphinx.Autodoc_Result :: struct
```

Autodoc_Result borrows host memory valid for the whole conversion. name names the generated text as a source (for diagnostics), such as “<autodoc flask.Flask>”.

```
name: string
```

```
text: string
```

```
messages: []Autodoc_Message
```

```
sphinx.Config :: struct
```

Config holds the conf.toml settings that reading needs, with Sphinx's names. It reaches the reader through `gd.Read_Options.extension`. Start from `default_config`: several defaults are true. Settings that the base reader owns (smart quotes, PEP and RFC base URLs, trimmed footnote reference space, language) live in settings.

```
using settings: rst.Settings
```

```
docname: string
```

the document's name, e.g. "usage/intro".

```
default_role: string
```

" " means Docutils' title-reference.

```
primary_domain: string
```

"py"; " " means none.

```
highlight_language: string
```

"default".

```
extlinks: []Extlink
```

```
rst_prolog: string
```

parsed before every document.

```
rst_epilog: string
```

parsed after every document.

```
numfig: bool
```

```
math_number_all: bool
```

```
todo_include_todos: bool
```

```
show_authors: bool
```

add_function_parentheses: bool

true.

add_module_names: bool

true.

toc_object_entries: bool

true: objects have table of contents entries.

toc_object_entries_show_parents: Toc_Parents

manpages_url: string

“{page}”, “{section}”, “{path}” substituted.

option_emphasise_placeholders: bool

trim_doctest_flags: bool

true: doctest flags and <BLANKLINE> are hidden.

version: string

The default substitutions |version|, |release|, and |today|, for documents that do not define them: the short version, the full release, and today’s date already formatted (Sphinx’s today, or today_fmt applied).

release: string

today: string

intersphinx: bool

sphinx.ext.intersphinx: the :external: roles are read when it is on, and an external role naming intersphinx_resolve_self is an ordinary reference.

intersphinx_resolve_self: string

labels: ^gd.Labels

The group names of info fields (“Parameters”, “Returns”) in the document’s language, as a host translates its labels; nil for English.

object_types: `[]Object_Type`

Directives and roles the project declares (see `object_types.odin`), and other names for directives.

directive_aliases: `[]Directive_Alias`

autodoc: `Autodoc_Provider`

`sphinx.ext.autodoc`: the host that generates what autodoc directives stand for; `nil` when the extension is off. See `autodoc.odin`.

odin_autodoc: `Autodoc_Provider`

The `odin` domain’s generating directives (`odin:autopackage`, ...): the host that reads the `Odin` source; `nil` leaves them unknown.

`sphinx.Directive_Alias :: struct`

name: `string`

“django-admin-option” names “option”; a target may name

target: `string`

“django-admin-option” names “option”; a target may name

`sphinx.Extlink :: struct`

`Extlink` is one `sphinx.ext.extlinks` entry: `:name:part` links to `url` with `%s` replaced by `part`; `caption`, when set, is the link text with `%s` replaced likewise.

name: `string`

url: `string`

caption: `string`

`sphinx.Markdown :: struct`

`Markdown` serves the `commonmark` reader with this layer’s directives and roles, the same handlers that serve `reStructuredText`: the reader parses `MyST Markdown` and hands each directive and role here (`commonmark.Extension`); they run on an embedded `reStructuredText` parser whose host is the `Markdown` reader (`rst.Host`), so their nested content is parsed as `Markdown` again. `eval-rst` parses its content as `reStructuredText`. The document is

resolved as reStructuredText is (references, footnotes, targets, bibliographic fields, smart quotes).

Use: `markdown_settings(&m, config, myst)` gives the reader settings to pass as `gd.Read_Options.extension` to the “commonmark” reader; `m` must outlive the read.

config: Config

settings: cm.Settings

ext: cm.Extension

state: State

parser: rst.Parser

host: rst.Host

reader: ^cm.Host

sphinx. Object_Type :: struct

directive: string

“setting”

role: string

“setting”; “” for none.

index: string

Sphinx’s `indextemplate`, “pair: %s; setting”; “” for none.

target: bool

`add_crossref_type`: a target and an index entry, no description.

first_word: bool

the object’s name is the signature up to its first space.

display: string

how the signature is shown, “%s” for it; “” shows it as written.

program: bool

the name becomes the program options that follow belong to.

sphinx.Toc_Parents :: enum u8

Toc_Parents is toc_object_entries_show_parents: how much of an object's hierarchy its table of contents entry shows.

Domain

as the domain names it: Python's Class.method, C++'s enclosing objects.

Hide

the object's own name.

All

every part: module.Class.method, ns::Class::f.

Procedures

sphinx.default_config :: proc() -> Config

default_config returns Sphinx's defaults, with the base reader's default_settings and smart quotes on. The value owns nothing; fields a caller sets must outlive the read.

sphinx.markdown_settings :: proc(m: ^Markdown, config: Config, myst: cm.Settings) -> ^cm.Settings

markdown_settings prepares m for one read of a Markdown document with the Sphinx configuration config and the MyST settings myst, and returns the reader settings. It copies both into m; the result points into m, so m must outlive the read and serves one read at a time. It cannot fail.

sphinx.read :: proc(ctx: ^gd.Read_Context) -> gd.Status

read reads reStructuredText with the Sphinx layer. ctx.options.extension is a ^Config (gd.read_extension), or empty for default_config; another type is refused with Invalid_Input. The read copies the Config and keeps nothing of it afterwards; its tables live in scratch, released before it returns. Otherwise it fails as rst.read.

sphinx.read_inline :: proc(ctx: ^gd.Read_Context, span: gd.Source_Span) -> gd.Status

read_inline parses a span of ctx.source as reStructuredText inline content with Sphinx's roles, for a translated message (gd.Read_Inline_Proc). rst_prolog

and `rst_epilog` do not apply to a message. Settings, ownership, and failures are read's.

```
sphinx.reader :: proc() -> gd.Reader
```

`reader` returns the “sphinx” reader descriptor, for a `gd.Registry`. It is a value with no state: registering it twice or copying it is harmless.

Constants

```
sphinx.CONFORMANCE :: gd.Conformance.Partial
```

```
sphinx.MAX_OBJECT_TYPES :: 64
```

```
sphinx.VERSION :: "0.1.0"
```

render/html

Package `html` renders a Guidedoc document as HTML without calling an allocator. The default output is semantic HTML with anchors, notes, and tables; the reference profile reproduces the CommonMark reference renderer byte for byte for conformance tests. Raw content and unsafe URLs are governed by the conversion policy.

Procedures

```
html.render :: proc(ctx: ^gd.Render_Context) -> gd.Status
```

`render` writes `ctx.view` as HTML through `ctx.out`: a whole page when `ctx.options.standalone`, else a fragment, and the CommonMark reference renderer's bytes when `ctx.options.reference_profile`. It borrows the context for the call and keeps nothing; its scratch is released before it returns. Problems go to `ctx.reports`; the result is `.Ok`, the status of the first error (a raw block or URL the policy refuses, an unresolved reference, nesting past the limits), or `.Capacity` when scratch runs out.

```
html.renderer :: proc() -> gd.Renderer
```

`renderer` returns the “html” renderer descriptor, for a `gd.Registry`. It is a value with no state: registering it twice or copying it is harmless.

Constants

```
html.CONFORMANCE :: gd.Conformance.Partial
```

```
html.VERSION :: "0.1.0"
```

render/text

Package `text_render` renders a Guidedoc document as plain text without calling an allocator, following the conventions of Sphinx’s text builder: paragraphs wrapped at 70 columns, titles underlined with `= ~` and the other section characters, `“* “` and `“1. “` list markers with hanging indents, definitions and literal blocks indented by three spaces, grid tables drawn with `+—+` borders, and `“Note:”` admonitions.

Inline content is laid out twice, once to measure and once into scratch, and then wrapped; table cells are laid out into scratch at trial widths before the table is drawn. Output goes only through the emitter, so the engine’s measure and write runs produce identical bytes.

Procedures

```
text.render :: proc(ctx: ^gd.Render_Context) -> gd.Status
```

`render` writes `ctx.view` as plain text through `ctx.out`. It borrows the context for the call and keeps nothing; its scratch is released before it returns. Problems go to `ctx.reports`; the result is `.Ok`, the status of the first error (raw content the policy refuses, an unresolved reference, nesting past the limits), or `.Capacity` when scratch cannot hold the indentation frames or a laid-out paragraph.

```
text.renderer :: proc() -> gd.Renderer
```

`renderer` returns the “text” renderer descriptor, for a `gd.Registry`. It is a value with no state: registering it twice or copying it is harmless.

Constants

```
text.CONFORMANCE :: gd.Conformance.Partial
```

```
text.VERSION :: "0.1.0"
```

render/typst

Package `typst_render` encodes a validated Guidedoc document as one Typst source file.

The output is a fixed, trusted preamble of `gd-` helper definitions followed by the body. Source text never becomes executable Typst: markup text escapes every ASCII punctuation character, and code, URLs, labels, and paths are string literals. Working memory comes only from the render context’s scratch; output goes only through the emitter, so the Measure and Write runs produce identical bytes.

Procedures

```
typst.emit_preamble :: proc(e: ^gd.Emitter)
```

emit_preamble writes the helper definitions every body relies on. A host composing a book from body-only chapters writes it once, before the chapters. The text is a constant: it cannot fail, and the emitter counts what does not fit.

```
typst.render :: proc(ctx: ^gd.Render_Context) -> gd.Status
```

render writes ctx.view as Typst source through ctx.out: the preamble, page settings, and body when ctx.options.standalone, else the body alone (see emit_preamble). It borrows the context for the call and keeps nothing but scratch taken from ctx, which the caller's scratch mark releases. Problems go to ctx.reports; the result is .Ok, the status of the first error (raw content the policy refuses, nesting past the limits), or .Capacity when scratch cannot hold the per-node marks.

```
typst.renderer :: proc() -> gd.Renderer
```

renderer returns the “typst” renderer descriptor, for a gd.Registry. It is a value with no state: registering it twice or copying it is harmless.

Constants

```
typst.CONFORMANCE :: gd.Conformance.Partial
```

```
typst.VERSION :: "0.1.1"
```

treesitter

Package treesitter is a thin binding of the tree-sitter C library and its Python grammar: parsers, trees, nodes, and cursors, and nothing more. It imports only Odin's own packages, so any Odin program can use it; Guidedog's autodoc reads Python source through it.

Types

```
treesitter.Cursor :: struct
```

Cursor is TSTreeCursor; cursor_delete frees it.

```
tree: rawptr
```

```
id: rawptr
```

```
ctx: [3]u32
```

```
tree sitter.Field_Id :: u16
```

```
tree sitter.Iterator :: struct
```

Iterator visits a node's children in order, named or all, for for child in

```
node: Node
```

```
index: int
```

```
count: int
```

```
named: bool
```

```
tree sitter.Language :: struct
```

```
tree sitter.Node :: struct
```

Node is TSNODE, a value that is valid while its tree lives.

```
ctx: [4]u32
```

```
id: rawptr
```

```
tree: ^Tree
```

```
tree sitter.Parser :: struct
```

Opaque handles.

```
tree sitter.Point :: struct
```

```
row: u32
```

```
column: u32
```

in bytes

```
tree sitter.Symbol :: u16
```

```
tree sitter.Tree :: struct
```

Procedures

```
tree-sitter.abi_version :: proc(l: ^Language) -> int
```

abi_version is the grammar's ABI, which the runtime must support.

```
tree-sitter.child :: proc(n: Node, i: int) -> Node
```

child is the node's child i of child_count, or a null node.

```
tree-sitter.child_count :: proc(n: Node) -> int
```

child_count counts the node's children, named and anonymous.

```
tree-sitter.children :: proc(n: Node, named := true) ->
Iterator
```

children iterates n's children, named ones only unless named is false, with next_child.

```
tree-sitter.cursor_delete :: proc(c: ^Cursor)
```

cursor_delete frees what cursor_new allocated.

```
tree-sitter.cursor_field_name :: proc(c: ^Cursor) -> string
```

cursor_field_name names the field the cursor's node fills in its parent, or "".

```
tree-sitter.cursor_first_child :: proc(c: ^Cursor) -> bool
```

cursor_first_child moves to the first child; false, and no move, when there is none.

```
tree-sitter.cursor_new :: proc(n: Node) -> Cursor
```

cursor_new starts a cursor at n, for walks that allocate nothing per step. tree-sitter allocates the cursor's stack; the caller frees it with cursor_delete.

```
tree-sitter.cursor_next_sibling :: proc(c: ^Cursor) -> bool
```

cursor_next_sibling moves to the next sibling; false, and no move, when there is none.

```
tree-sitter.cursor_node :: proc(c: ^Cursor) -> Node
```

cursor_node is the node the cursor is at.

```
treesitter.cursor_parent :: proc(c: ^Cursor) -> bool
```

`cursor_parent` moves to the parent; false, and no move, at the node the cursor started at.

```
treesitter.deeper_than :: proc(n: Node, limit: int) -> bool
```

`deeper_than` reports whether the tree below `n` nests more than `limit` levels (`n`'s children are level 1). Parsing needs no recursion however deep the source nests, but recursive walks over the tree do: a program checks first, and refuses trees deeper than its walks can hold. It walks with a cursor, so it needs no recursion itself.

```
treesitter.end_byte :: proc(n: Node) -> int
```

`end_byte` is the byte offset just past the node in the parsed text.

```
treesitter.end_point :: proc(n: Node) -> Point
```

`end_point` is the 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"}
```

उपलब्धता: Windows.

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

उपलब्धता: 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"}
```

उपलब्धता: Linux, FreeBSD, OpenBSD, NetBSD, WASI, JS, Orca, Freestanding.

इंडेक्स

A

- abi_version (procedure in treesitter), 335
- absolute (procedure in project), 271
- add (procedure in i18n), 222
 - (procedure in inventory), 227
- add_attribute (procedure in core), 117
- add_attribute_string (procedure in core), 117
- add_catalog (procedure in gettext), 154
- add_common (procedure in readers/rst), 312
- add_filter (procedure in jinja), 236
- add_flags (procedure in core), 118
- add_function (procedure in jinja), 236
- add_global (procedure in jinja), 236
- add_leaf (procedure in core), 118
- add_message (procedure in gettext), 154
- add_source (procedure in core), 118
- add_template (procedure in jinja), 236
- add_test (procedure in jinja), 237
- add_text (procedure in core), 118
- Admonition_Kind (enum in core), 82
- Admonition_Kind.Attention (enumerator in core), 82
- Admonition_Kind.Caution (enumerator in core), 82
- Admonition_Kind.Danger (enumerator in core), 82
- Admonition_Kind.Error (enumerator in core), 82
- Admonition_Kind.Generic (enumerator in core), 82
- Admonition_Kind.Hint (enumerator in core), 82
- Admonition_Kind.Important (enumerator in core), 82
- Admonition_Kind.Note (enumerator in core), 82
- Admonition_Kind.Seealso (enumerator in core), 82
- Admonition_Kind.Tip (enumerator in core), 82
- Admonition_Kind.TODO (enumerator in core), 82
- Admonition_Kind.Warning (enumerator in core), 82
- admonition_label (procedure in core), 118
- admonition_name (procedure in core), 118
- Align (enum in core), 82
- Align.Center (enumerator in core), 82
- Align.Left (enumerator in core), 82
- Align.None (enumerator in core), 82
- Align.Right (enumerator in core), 82
- All (struct in pyscan), 274
- All.defined (field in pyscan), 274
- All.known (field in pyscan), 274
- All.names (field in pyscan), 274
- Analysis (struct in pyscan), 275
- Analysis.annotation_order (field in pyscan), 275
- Analysis.annotations (field in pyscan), 275
- Analysis.attr_docs (field in pyscan), 275
- Analysis.attr_order (field in pyscan), 275
- Analysis.counter (field in pyscan), 275
- Analysis.finals (field in pyscan), 275
- Analysis.overloads (field in pyscan), 275
- Analysis.tagorder (field in pyscan), 275
- Anchor_Extra (struct in core), 83
- Anchor_Extra.item (field in core), 83
- Anchor_Extra.note (field in core), 83
- Anchor_Extra.number (field in core), 83
- Anchor_Extra.owner (field in core), 83
- Anchor_Extra.span (field in core), 83
- anchor_extras (procedure in core), 118
- anchor_ids (procedure in core), 118
- Anchor_Ids (struct in core), 83
- Anchor_Ids.span (field in core), 83
- anchor_of (procedure in core), 118
- anchors (struct in core), 83
- anchors.bytes (field in core), 84
- anchors.count (field in core), 84
- anchors.counter (field in core), 84
- anchors.extra_count (field in core), 84
- anchors.extras (field in core), 84
- anchors.index (field in core), 84
- anchors.labels (field in core), 84
- anchors.names (field in core), 84
- anchors.next (field in core), 84
- anchors.own (field in core), 84
- anchors.prefix (field in core), 84
- anchors.prefix_length (field in core), 84
- anchors.spans (field in core), 84
- append_value (procedure in jinja), 237
- apply (procedure in gds), 142
- apply_host_role (procedure in readers/rst/inline), 323
- Arch (type in odindoc), 243
- Archs (type in odindoc), 243

- Arena (struct in host), 189
- Arena.backing (field in host), 189
- Arena.block (field in host), 189
- Arena.blocks (field in host), 189
- Arena.bytes (field in host), 189
- Arena.last (field in host), 189
- arena_allocator (procedure in host), 202
- ARENA_BLOCK_BYTES (constant in host), 215
- arena_destroy (procedure in host), 202
- arena_init (procedure in host), 202
- arg (procedure in jinja), 237
- arg_int (procedure in core), 119
- (procedure in jinja), 237
- Arg_Kind (enum in core), 85
- Arg_Kind.Int (enumerator in core), 85
- Arg_Kind.None (enumerator in core), 85
- Arg_Kind.Span (enumerator in core), 85
- Arg_Kind.Static (enumerator in core), 85
- Arg_Rule (enum in readers/rst), 301
- Arg_Rule.None (enumerator in readers/rst), 301
- Arg_Rule.Optional (enumerator in readers/rst), 301
- Arg_Rule.Required (enumerator in readers/rst), 301
- arg_span (procedure in core), 119
- arg_static (procedure in core), 119
- Artifact_Request (struct in host), 189
- Artifact_Request.allocator (field in host), 190
- Artifact_Request.budget (field in host), 190
- Artifact_Request.emit_typst (field in host), 190
- Artifact_Request.force (field in host), 190
- Artifact_Request.from (field in host), 190
- Artifact_Request.input (field in host), 189
- Artifact_Request.limits (field in host), 190
- Artifact_Request.output (field in host), 190
- Artifact_Request.policy (field in host), 190
- Artifact_Request.read (field in host), 190
- Artifact_Request.render (field in host), 190
- Artifact_Request.roots (field in host), 190
- Artifact_Request.to (field in host), 190
- Artifact_Request.typst_options (field in host), 190
- Artifact_Request.typst_program (field in host), 190
- Artifact_Result (struct in host), 190
- Artifact_Result.exit (field in host), 191
- Artifact_Result.problem (field in host), 191
- Artifact_Result.reports (field in host), 191
- Artifact_Result.stdout (field in host), 191
- Artifact_Result.written (field in host), 191
- Attr_Key (struct in pyscan), 275
- Attr_Key.name (field in pyscan), 275
- Attr_Key.scope (field in pyscan), 275
- attribute (procedure in core), 119
- Attribute (struct in core), 85
- (struct in graphdog), 161
- Attribute.key (field in core), 85
- Attribute.kind (field in graphdog), 161
- Attribute.name (field in core), 85
- (field in graphdog), 161
- Attribute.value (field in core), 85
- (field in graphdog), 161
- Attribute_Key (enum in core), 85
- Attribute_Key.Align (enumerator in core), 86
- Attribute_Key.Alt (enumerator in core), 85
- Attribute_Key.Class (enumerator in core), 85
- Attribute_Key.Custom (enumerator in core), 85
- Attribute_Key.Format (enumerator in core), 86
- Attribute_Key.Height (enumerator in core), 85
- Attribute_Key.Id (enumerator in core), 85
- Attribute_Key.Language (enumerator in core), 85
- Attribute_Key.Name (enumerator in core), 85
- Attribute_Key.Scale (enumerator in core), 85
- Attribute_Key.Source (enumerator in core), 86
- Attribute_Key.Start (enumerator in core), 86
- Attribute_Key.Target (enumerator in core), 86
- Attribute_Key.Title (enumerator in core), 86
- Attribute_Key.Width (enumerator in core), 85
- Attribute_Key.Widths (enumerator in core), 86
- Attribute_Kind (enum in graphdog), 161
- Attribute_Kind.Edge (enumerator in graphdog), 162
- Attribute_Kind.Graph (enumerator in graphdog), 161
- Attribute_Kind.Node (enumerator in graphdog), 161
- Attribute_Range (struct in core), 86
- Attribute_Range.count (field in core), 86
- Attribute_Range.first (field in core), 86
- attributes (procedure in core), 119
- Attributes (struct in core), 86
- Attributes.end (field in core), 86
- Attributes.next (field in core), 86
- autodoc (package), 76
- Autodoc_Message (struct in readers/sphinx), 324
- Autodoc_Message.code (field in readers/sphinx), 324
- Autodoc_Message.hint (field in readers/sphinx), 324
- Autodoc_Message.message (field in readers/sphinx), 324
- Autodoc_Message.severity (field in readers/sphinx), 324
- Autodoc_Message.title (field in readers/sphinx), 324
- Autodoc_Option (struct in readers/sphinx), 324
- Autodoc_Option.name (field in readers/sphinx), 324
- Autodoc_Option.value (field in readers/sphinx), 324

Autodoc_Provider (struct in readers/
sphinx), 325
Autodoc_Provider.procedure (field in
readers/sphinx), 325
Autodoc_Provider.user (field in readers/
sphinx), 325
Autodoc_Request (struct in readers/
sphinx), 325
Autodoc_Request.argument (field in
readers/sphinx), 325
Autodoc_Request.class (field in readers/
sphinx), 325
Autodoc_Request.content (field in readers/
sphinx), 325
Autodoc_Request.directive (field in
readers/sphinx), 325
Autodoc_Request.docname (field in
readers/sphinx), 325
Autodoc_Request.module (field in readers/
sphinx), 325
Autodoc_Request.options (field in readers/
sphinx), 325
Autodoc_Result (struct in readers/sphinx),
325
Autodoc_Result.messages (field in readers/
sphinx), 325
Autodoc_Result.name (field in readers/
sphinx), 325
Autodoc_Result.text (field in readers/
sphinx), 325
autoescaping (procedure in jinja), 237
available_memory (procedure in host), 202
AVAILABLE_OVERRIDE (constant in
host), 215

B

Backend (struct in typst), 340
Backend.embedded (field in typst), 340
Backend.program (field in typst), 340
Backend.version (field in typst), 340
backend_problem (procedure in host), 203
base_objects (procedure in pyscan), 286
begin_node (procedure in core), 119
bind_text_domain (procedure in gettext),
154
blank (procedure in readers/rst), 312
boolean (procedure in jinja), 237
Bound_Catalog (struct in gettext), 145
Bound_Catalog.catalog (field in gettext),
145
Bound_Catalog.domain (field in gettext),
145
bridge (foreign import in typst), 343
Budget (struct in host), 191
Budget.active (field in host), 191
Budget.freed (field in host), 192
Budget.limit (field in host), 191
Budget.mutex (field in host), 192
Budget.peak (field in host), 191
Budget.policy (field in host), 192
Budget.refused_need (field in host), 192
Budget.refused_used (field in host), 192
Budget.stopped (field in host), 192
Budget.used (field in host), 191

Budget.waiting (field in host), 191
budget_advice (procedure in host), 203
budget_hint (procedure in host), 203
budget_join (procedure in host), 203
budget_leave (procedure in host), 203
budget_limit (procedure in host), 203
Budget_Policy (struct in host), 192
Budget_Policy.disk (field in host), 192
Budget_Policy.locate (field in host), 192
Budget_Policy.locate_user (field in host),
192
Budget_Policy.on_refusal (field in host),
192
Budget_Policy.spill (field in host), 192
Budget_Policy.user (field in host), 192
budget_problem (procedure in host), 203
budget_report (procedure in host), 203
Buffer (struct in core), 86
Buffer.bytes (field in core), 86
Buffer.overflow (field in core), 86
Buffer.used (field in core), 86
Buffer.wanted (field in core), 86
buffer_string (procedure in core), 119
buffer_write (procedure in core), 119
buffer_write_byte (procedure in core), 119
build (procedure in gds), 142
(procedure in project), 271
build_anchors (procedure in core), 119
BUILD_BUDGET (constant in project), 274
build_index (procedure in gettext), 154
build_limits (procedure in project), 272
Build_Options (struct in project), 250
Build_Options.all (field in project), 250
Build_Options.arena_block (field in
project), 252
Build_Options.budget (field in project), 251
Build_Options.fresh (field in project), 250
Build_Options.heap (field in project), 251
Build_Options.jobs (field in project), 251
Build_Options.keep_going (field in
project), 251
Build_Options.max_depth (field in
project), 251
Build_Options.max_nodes (field in
project), 251
Build_Options.memory (field in project),
251
Build_Options.nitpicky (field in project),
251
Build_Options.now (field in project), 252
Build_Options.output_dir (field in project),
252
Build_Options.stack_bytes (field in
project), 251
Build_Options.tags (field in project), 251
Build_Options.untrusted (field in project),
251
Build_Options.warnings_as_errors (field in
project), 251
Build_Options.year (field in project), 252
Build_Result (struct in project), 252
Build_Result.budget (field in project), 253
Build_Result.exit (field in project), 252
Build_Result.memory (field in project), 253
Build_Result.output (field in project), 252
Build_Result.owner (field in project), 253
Build_Result.peak (field in project), 253

Build_Result.problem (field in project), 252
 Build_Result.published (field in project), 253
 Build_Result.reached (field in project), 253
 Build_Result.read (field in project), 252
 Build_Result.reports (field in project), 252
 Build_Result.reused (field in project), 252
 Build_Result.unchanged (field in project), 252
 Build_Result.unread (field in project), 252
 Build_Result.unsettled (field in project), 253
 Build_Result.written (field in project), 252
 Build_Stage (enum in project), 253
 Build_Stage.Preparing (enumerator in project), 253
 Build_Stage.Reading (enumerator in project), 253
 Build_Stage.Writing (enumerator in project), 253
 Build_Stage.Written (enumerator in project), 253
 Builder (struct in core), 87
 Builder.cancel (field in core), 87
 Builder.limits (field in core), 87
 Builder.need (field in core), 87
 Builder.open (field in core), 87
 Builder.reports (field in core), 87
 Builder.snap (field in core), 87
 Builder.status (field in core), 87
 Builder.work (field in core), 87
 builder_capacity (procedure in core), 120
 builder_fail (procedure in core), 120
 builder_finish (procedure in core), 120
 builder_init (procedure in core), 120
 Builder_Kind (enum in project), 254
 Builder_Kind.Dirhtml (enumerator in project), 254
 Builder_Kind.Dummy (enumerator in project), 254
 Builder_Kind.Gettext (enumerator in project), 254
 Builder_Kind.Html (enumerator in project), 254
 Builder_Kind.Pdf (enumerator in project), 254
 Builder_Kind.Singlehtml (enumerator in project), 254
 Builder_Kind.Text (enumerator in project), 254
 builder_named (procedure in project), 272
 BUILDER_NAMES (variable in project), 274
 BUILDER_OPEN (constant in core), 135
 bullet_width (procedure in readers/rst), 312

C

Call (struct in jinja), 228
 Call.args (field in jinja), 228
 Call.kwargs (field in jinja), 228
 Call.name (field in jinja), 228
 Call.self (field in jinja), 228
 Call.state (field in jinja), 228
 Call.user (field in jinja), 228

call_filter (procedure in jinja), 237
 call_test (procedure in jinja), 237
 Callable (struct in jinja), 228
 Callable.macro (field in jinja), 229
 Callable.name (field in jinja), 228
 Callable.procedure (field in jinja), 228
 Callable.self (field in jinja), 228
 Callable.user (field in jinja), 228
 cancel_probe (procedure in host), 204
 Cancel_Probe (struct in core), 87
 Cancel_Probe.procedure (field in core), 87
 Cancel_Probe.user (field in core), 87
 canonical_roots (procedure in host), 204
 Capacity_Need (struct in core), 87
 Capacity_Need.exact (field in core), 87
 Capacity_Need.minimum (field in core), 87
 Capacity_Need.region (field in core), 87
 Catalog (struct in gds), 139
 (struct in gettext), 145
 Catalog allocator (field in gettext), 146
 Catalog.allow_fuzzy (field in gettext), 146
 Catalog.arena (field in gettext), 146
 Catalog.has_header (field in gettext), 145
 Catalog.header (field in gettext), 145
 Catalog.index (field in gettext), 146
 Catalog.messages (field in gettext), 145
 Catalog.name (field in gettext), 145
 Catalog.owner (field in gettext), 146
 Catalog.plural (field in gettext), 145
 Catalog.records (field in gds), 139
 Catalog.reserved (field in gds), 139
 Catalog_Arena (struct in gettext), 146
 Catalog_Arena.refused (field in gettext), 146
 Catalog_Arena.using (field in gettext), 146
 catalog_bytes (procedure in gettext), 154
 catalog_refused (procedure in gettext), 154
 catch_interrupts (procedure in host), 204
 cell_info (procedure in core), 120
 Cell_Info (struct in core), 88
 Cell_Info.align (field in core), 88
 Cell_Info.col_span (field in core), 88
 Cell_Info.row_span (field in core), 88
 Change (struct in gds), 139
 Change.after (field in gds), 140
 Change.before (field in gds), 139
 Change.existed (field in gds), 140
 Change.path (field in gds), 139
 Change.present (field in gds), 140
 charge (procedure in host), 204
 charset (procedure in gettext), 155
 check_args (procedure in jinja), 237
 check_code (procedure in highlight), 184
 check_index (procedure in gds), 143
 check_render (procedure in core), 120
 check_sources (procedure in gds), 143
 check_style (procedure in highlight), 184
 child (procedure in treesitter), 335
 child_count (procedure in core), 120
 (procedure in treesitter), 335
 children (procedure in core), 120
 (procedure in treesitter), 335
 Children (struct in core), 88
 Children.end (field in core), 88
 Children.next (field in core), 88
 Class (struct in pycan), 275
 Class.bases (field in pycan), 276
 Class.decorators (field in pycan), 276
 Class.doc (field in pycan), 276
 Class.keywords (field in pycan), 276

- Class.module (field in pyscan), 275
- Class.mro_complete (field in pyscan), 276
- Class.mro_done (field in pyscan), 276
- Class.mro_order (field in pyscan), 276
- Class.name (field in pyscan), 275
- Class.node (field in pyscan), 276
- Class.outer (field in pyscan), 276
- Class.qualname (field in pyscan), 276
- Class.scope (field in pyscan), 276
- class_attr (procedure in pyscan), 286
- class_directive (procedure in readers/rst), 313
- clean (procedure in project), 272
- clear_cache (procedure in jinja), 238
- cli (package), 80
- clone_message (procedure in gettext), 155
- close (procedure in graphdog), 165
 - (procedure in readers/rst), 313
- code_caption (procedure in core), 121
- code_text (procedure in readers/rst), 313
- Color (struct in highlight), 169
- Color.hex (field in highlight), 169
- Color.set (field in highlight), 169
- Compact (struct in i18n), 217
- Compact.mode (field in i18n), 217
- Compact.name (field in i18n), 217
- compile_file (procedure in typst), 342
- compile_plural (procedure in gettext), 155
- compile_text (procedure in typst), 342
- Config (struct in autodoc), 77
 - (struct in odindoc), 243
 - (struct in project), 254
 - (struct in readers/sphinx), 326
- Config.add_function_parentheses (field in project), 256
 - (field in readers/sphinx), 327
- Config.add_module_names (field in project), 256
 - (field in readers/sphinx), 327
- Config.author (field in project), 254
- Config.autoclass_content (field in autodoc), 77
 - (field in project), 258
- Config.autodoc (field in readers/sphinx), 328
- Config.autodoc_class_signature (field in autodoc), 77
 - (field in project), 258
- Config.autodoc_default_options (field in autodoc), 77
 - (field in project), 259
- Config.autodoc_docstring_signature (field in autodoc), 77
 - (field in project), 259
- Config.autodoc_inherit_docstrings (field in autodoc), 77
 - (field in project), 259
- Config.autodoc_member_order (field in autodoc), 77
 - (field in project), 259
- Config.autodoc_mock_imports (field in autodoc), 77
 - (field in project), 259
- Config.autodoc_preserve_defaults (field in autodoc), 77
 - (field in project), 259
- Config.autodoc_source_paths (field in project), 258
- Config.autodoc_type_aliases (field in autodoc), 77
 - (field in project), 259
- Config.autodoc_typehints (field in autodoc), 77
 - (field in project), 259
- Config.autodoc_typehints_description_target (field in autodoc), 77
 - (field in project), 259
- Config.autodoc_typehints_format (field in autodoc), 77
 - (field in project), 259
- Config.autodoc_use_type_comments (field in autodoc), 78
 - (field in project), 259
- Config.autodoc_warningiserror (field in project), 259
- Config.autosectionlabel_maxdepth (field in project), 257
- Config.autosectionlabel_prefix_document (field in project), 257
- Config.build_dir (field in project), 254
- Config.copyright (field in project), 254
- Config.default_options (field in odindoc), 243
- Config.default_role (field in project), 255
 - (field in readers/sphinx), 326
- Config.directive_aliases (field in project), 257
 - (field in readers/sphinx), 328
- Config.docname (field in readers/sphinx), 326
- Config.exclude_patterns (field in project), 255
- Config.extensions (field in project), 255
- Config.extlinks (field in project), 257
 - (field in readers/sphinx), 326
- Config.extlinks_detect_hardcoded_links (field in project), 257
- Config.figure_language_filename (field in project), 264
- Config.gettext_additional_targets (field in project), 264
- Config.gettext_allow_fuzzy_translations (field in project), 264
- Config.gettext_auto_build (field in project), 264
- Config.gettext_compact (field in project), 264
- Config.gettext_exclude_patterns (field in project), 264
- Config.gettext_language_team (field in project), 264
- Config.gettext_last_translator (field in project), 264
- Config.gettext_location (field in project), 264
- Config.gettext_uuid (field in project), 264
- Config.graphviz_dot (field in project), 262
- Config.graphviz_dot_args (field in project), 262

- Config.graphviz_output_format (field in project), 262
- Config.highlight_language (field in project), 256
- Config.html_additional_pages (field in project), 261
- Config.html_baseurl (field in project), 261
- Config.html_context (field in project), 261
- Config.html_copy_source (field in project), 261
- Config.html_css_files (field in project), 260
- Config.html_domain_indices (field in project), 261
- Config.html_extra_path (field in project), 260
- Config.html_favicon (field in project), 260
- Config.html_file_suffix (field in project), 261
- Config.html_js_files (field in project), 260
- Config.html_last_updated_fmt (field in project), 260
- Config.html_link_suffix (field in project), 261
- Config.html_logo (field in project), 260
- Config.html_permalinks (field in project), 260
- Config.html_permalinks_icon (field in project), 261
- Config.html_secnumber_suffix (field in project), 261
- Config.html_short_title (field in project), 260
- Config.html_short_title_derived (field in project), 264
- Config.html_show_copyright (field in project), 261
- Config.html_show_sourcelink (field in project), 261
- Config.html_show_sphinx (field in project), 261
- Config.html_sourcelink_suffix (field in project), 261
- Config.html_split_index (field in project), 261
- Config.html_static_path (field in project), 260
- Config.html_theme (field in project), 260
- Config.html_theme_options (field in project), 260
- Config.html_title (field in project), 260
- Config.html_title_derived (field in project), 264
- Config.html_use_index (field in project), 261
- Config.include_patterns (field in project), 255
- Config.include_roots (field in project), 255
- Config.intersphinx (field in readers/sphinx), 327
- Config.intersphinx_cache_limit (field in project), 258
- Config.intersphinx_disabled_reftypes (field in project), 258
- Config.intersphinx_mapping (field in project), 258
- Config.intersphinx_resolve_self (field in project), 258
- Config.intersphinx_sphinx (field in readers/sphinx), 327
- Config.intersphinx_timeout (field in project), 258
- Config.keep_warnings (field in project), 257
- Config.labels (field in readers/sphinx), 327
- Config.language (field in project), 255
- Config.locale_dirs (field in project), 263
- Config.manpages_url (field in project), 257
- Config.math_eqref_format (field in project), 256
- Config.math_number_all (field in project), 256
- Config.math_number_all (field in readers/sphinx), 326
- Config.math_numfig (field in project), 256
- Config.mathjax3_config (field in project), 258
- Config.mathjax4_config (field in project), 258
- Config.mathjax_config_path (field in project), 258
- Config.mathjax_display (field in project), 258
- Config.mathjax_inline (field in project), 258
- Config.mathjax_options (field in project), 258
- Config.mathjax_path (field in project), 257
- Config.member_order (field in odindoc), 243
- Config.modindex_common_prefix (field in project), 261
- Config.myst_amsmath_enable (field in project), 263
- Config.myst_commonmark_only (field in project), 263
- Config.myst_disable_syntax (field in project), 263
- Config.myst_dmath_allow_digits (field in project), 263
- Config.myst_dmath_allow_labels (field in project), 262
- Config.myst_dmath_allow_space (field in project), 263
- Config.myst_dmath_double_inline (field in project), 263
- Config.myst_dmath_enable (field in project), 263
- Config.myst_enable_extensions (field in project), 262
- Config.myst_footnote_transition (field in project), 263
- Config.myst_heading_anchors (field in project), 262

- Config.myst_html_meta (field in project), 263
- Config.myst_linkify_fuzzy_links (field in project), 263
- Config.myst_sub_delimiters (field in project), 263
- Config.myst_substitutions (field in project), 263
- Config.myst_url_schemes (field in project), 263
- Config.myst_words_per_minute (field in project), 263
- Config.nitpick_ignore (field in project), 256
- Config.nitpick_ignore_regex (field in project), 256
- Config.nitpick_patterns (field in project), 256
- Config.nitpicky (field in project), 256
- Config.numfig (field in project), 256 (field in readers/sphinx), 326
- Config.numfig_format (field in project), 256
- Config.numfig_secnum_depth (field in project), 256
- Config.object_types (field in project), 257 (field in readers/sphinx), 328
- Config.odin_autoapi_add_toctree_entry (field in project), 260
- Config.odin_autoapi_dirs (field in project), 259
- Config.odin_autoapi_generate_api_docs (field in project), 260
- Config.odin_autoapi_member_order (field in project), 260
- Config.odin_autoapi_options (field in project), 260
- Config.odin_autoapi_root (field in project), 259
- Config.odin_autodoc (field in readers/sphinx), 328
- Config.odin_collections (field in project), 259
- Config.option_emphasise_placeholders (field in project), 257 (field in readers/sphinx), 327
- Config.pdf_documents (field in project), 261
- Config.pdf_font_paths (field in project), 262
- Config.pdf_fonts (field in project), 262
- Config.pdf_logo (field in project), 262
- Config.pdf_packages (field in project), 262
- Config.pdf_paper_size (field in project), 262
- Config.pdf_preamble (field in project), 262
- Config.pdf_show_urls (field in project), 262
- Config.pdf_toplevel_sectioning (field in project), 262
- Config.primary_domain (field in project), 255 (field in readers/sphinx), 326
- Config.project (field in project), 254
- Config.pygments_dark_style (field in project), 256
- Config.pygments_style (field in project), 256
- Config.python_display_short_literal_types (field in autodoc), 78 (field in project), 259
- Config.release (field in project), 255 (field in readers/sphinx), 327
- Config.root_doc (field in project), 255
- Config.rst_epilog (field in project), 255 (field in readers/sphinx), 326
- Config.rst_prolog (field in project), 255 (field in readers/sphinx), 326
- Config.show_authors (field in project), 256 (field in readers/sphinx), 326
- Config.smartquotes (field in project), 257
- Config.source_dir (field in project), 254
- Config.source_suffix (field in project), 255
- Config.strip_signature_backslash (field in autodoc), 78 (field in project), 259
- Config.suppress_warnings (field in project), 257
- Config.tab_width (field in autodoc), 78
- Config.templates_path (field in project), 255
- Config.toc_object_entries (field in project), 256 (field in readers/sphinx), 327
- Config.toc_object_entries_show_parents (field in project), 257 (field in readers/sphinx), 327
- Config.today (field in project), 255 (field in readers/sphinx), 327
- Config.today_fmt (field in project), 255
- Config.todo_include_todos (field in project), 257 (field in readers/sphinx), 326
- Config.translation_progress_classes (field in project), 264
- Config.trim_doctest_flags (field in project), 257 (field in readers/sphinx), 327
- Config.trim_footnote_reference_space (field in project), 257
- Config.typehint_fields (field in autodoc), 78
- Config.typst (field in project), 262
- Config.using (field in readers/sphinx), 326
- Config.version (field in project), 255 (field in readers/sphinx), 327
- CONFIG_FILE (constant in project), 274
- confined (procedure in host), 204
- Confinement (enum in host), 193
- confinement (procedure in host), 204
- Confinement.Inside (enumerator in host), 193
- Confinement.Outside (enumerator in host), 193
- Confinement.Through_Link (enumerator in host), 193
- CONFORMANCE (constant in readers/commonmark), 301 (constant in readers/rst), 318

- (constant in readers/sphinx), 331
- (constant in render/html), 331
- (constant in render/text), 332
- (constant in render/typst), 333
- Conformance (enum in core), 88
- Conformance.Conformant (enumerator in core), 88
- Conformance.Partial (enumerator in core), 88
- Conformance.Planned (enumerator in core), 88
- containing_root (procedure in host), 204
- Content (enum in core), 88
- Content.Blocks (enumerator in core), 88
- Content.Inlines (enumerator in core), 88
- Content.Nothing (enumerator in core), 88
- Content.Structure (enumerator in core), 88
- content_error (procedure in core), 121
- content_model (procedure in core), 121
- Content_Policy (enum in core), 88
- Content_Policy.Allow (enumerator in core), 89
- Content_Policy.Omit (enumerator in core), 88
- Content_Policy.Refuse (enumerator in core), 88
- Content_Rule (enum in readers/rst), 301
- Content_Rule.None (enumerator in readers/rst), 301
- Content_Rule.Optional (enumerator in readers/rst), 301
- Content_Rule.Required (enumerator in readers/rst), 301
- contents_scope (procedure in core), 121
- Context (struct in graphdog), 162
- Context.gvc (field in graphdog), 162
- context_of (procedure in readers/commonmark), 299
- convert (procedure in core), 121
- convert_artifact (procedure in host), 205
- convert_text (procedure in core), 121
- copy_payload (procedure in core), 122
- copy_subtree (procedure in core), 122
- core (package), 82
- CORE_CAPACITY (constant in core), 135
- CORE_DEPTH (constant in core), 135
- CORE_NODES (constant in core), 135
- CORE_STACK (constant in core), 136
- CORE_UNHANDLED_KIND (constant in core), 136
- CORE_WORK (constant in core), 136
- count_tokens (procedure in highlight), 184
- CRASH_EXIT (constant in host), 215
- crash_on_assertion (procedure in host), 205
- create_environment (procedure in jinja), 238
- curl (foreign import in fetch), 139
- current (procedure in core), 122
- current_kind (procedure in core), 122
- Cursor (struct in treesitter), 333
- Cursor.ctx (field in treesitter), 333
- Cursor.id (field in treesitter), 333
- Cursor.tree (field in treesitter), 333
- cursor_delete (procedure in treesitter), 335

- cursor_field_name (procedure in treesitter), 335
- cursor_first_child (procedure in treesitter), 335
- cursor_new (procedure in treesitter), 335
- cursor_next_sibling (procedure in treesitter), 335
- cursor_node (procedure in treesitter), 335
- cursor_parent (procedure in treesitter), 336
- custom (procedure in readers/rst), 313
- custom_attribute (procedure in core), 122
- cut (procedure in readers/rst), 313
- Cycler (struct in jinja), 229
- Cycler.items (field in jinja), 229
- Cycler.pos (field in jinja), 229

D

- Decl (struct in odindoc), 243
- Decl.attributes (field in odindoc), 243
- Decl.availability (field in odindoc), 244
- Decl.docs (field in odindoc), 243
- Decl.file (field in odindoc), 244
- Decl.foreign_lib (field in odindoc), 243
- Decl.kind (field in odindoc), 243
- Decl.line (field in odindoc), 244
- Decl.members (field in odindoc), 243
- Decl.name (field in odindoc), 243
- Decl.order (field in odindoc), 244
- Decl.private (field in odindoc), 243
- Decl.signature (field in odindoc), 243
- Decl.targets (field in odindoc), 243
- Decl.variant (field in odindoc), 244
- decode_last_rune (procedure in core), 122
- decode_rune (procedure in core), 122
- dedent (procedure in readers/rst), 313
- deeper_than (procedure in treesitter), 336
- Def (struct in pyscan), 276
- Def.annotation (field in pyscan), 277
- Def.class (field in pyscan), 277
- Def.function (field in pyscan), 277
- Def.kind (field in pyscan), 276
- Def.line (field in pyscan), 277
- Def.member (field in pyscan), 277
- Def.name (field in pyscan), 276
- Def.node (field in pyscan), 276
- Def.runtime (field in pyscan), 276
- Def.target (field in pyscan), 277
- Def.tuple (field in pyscan), 277
- Def.value (field in pyscan), 277
- Def_Kind (enum in pyscan), 277
- Def_Kind.Annotation (enumerator in pyscan), 277
- Def_Kind.Assign (enumerator in pyscan), 277
- Def_Kind.Class (enumerator in pyscan), 277
- Def_Kind.Function (enumerator in pyscan), 277
- Def_Kind.Import (enumerator in pyscan), 277
- Def_Kind.Import_From (enumerator in pyscan), 278
- Def_Kind.Type_Alias (enumerator in pyscan), 278
- def_object (procedure in pyscan), 286
- DEFAULT_BUDGET (constant in host), 215

- default_config (procedure in autodoc), 80
- (procedure in readers/sphinx), 330
- DEFAULT_FIGURE_FILENAME (constant in i18n), 225
- default_limits (procedure in core), 122
- default_merge_options (procedure in gettext), 155
- Default_Option (struct in autodoc), 78
- Default_Option.flag (field in autodoc), 78
- Default_Option.name (field in autodoc), 78
- Default_Option.value (field in autodoc), 78
- default_policy (procedure in core), 123
- default_read_options (procedure in core), 123
- default_request (procedure in core), 123
- default_settings (procedure in readers/commonmark), 299
- (procedure in readers/rst), 313
- default_stack_budget (procedure in host), 205
- DEFAULT_STACK_BYTES (constant in core), 136
- default_write_options (procedure in gettext), 155
- defaults (package), 136
- delete_error (procedure in graphdog), 165
- describe (procedure in host), 205
- describe_all (procedure in host), 205
- destroy (procedure in fetch), 139
- (procedure in gettext), 155
- (procedure in pyscan), 286
- destroy_environment (procedure in jinja), 238
- destroy_error (procedure in gettext), 155
- (procedure in jinja), 238
- destroy_storage (procedure in host), 205
- destroy_template (procedure in i18n), 222
- destroy_translator (procedure in gettext), 155
- diagnostic (procedure in core), 123
- Diagnostic (struct in core), 89
- (struct in typst), 340
- Diagnostic.args (field in core), 89
- Diagnostic.column (field in typst), 340
- Diagnostic.file (field in typst), 340
- Diagnostic.line (field in typst), 340
- Diagnostic.message (field in typst), 340
- Diagnostic.severity (field in typst), 340
- Diagnostic.span (field in core), 89
- Diagnostic.stdin (field in typst), 340
- Diagnostic.using (field in core), 89
- Diagnostic_Arg (struct in core), 89
- Diagnostic_Arg.int (field in core), 89
- Diagnostic_Arg.kind (field in core), 89
- Diagnostic_Arg.span (field in core), 89
- Diagnostic_Arg.static (field in core), 89
- DIAGNOSTICS_SCHEMA (constant in host), 215
- dict (procedure in jinja), 238
- Dict (struct in jinja), 229
- Dict allocator (field in jinja), 229
- Dict.index (field in jinja), 229
- Dict.keys (field in jinja), 229
- Dict.kind (field in jinja), 229
- Dict.name (field in jinja), 229
- Dict.values (field in jinja), 229
- Dict_Kind (enum in jinja), 229
- Dict_Kind.Dict (enumerator in jinja), 229
- Dict_Kind.Module (enumerator in jinja), 229
- Dict_Kind.Namespace (enumerator in jinja), 229
- Dir_Opt (struct in readers/rst), 302
- Dir_Opt.lines (field in readers/rst), 302
- Dir_Opt.span (field in readers/rst), 302
- Dir_Opt.spec (field in readers/rst), 302
- Dir_Spec (struct in readers/rst), 302
- Dir_Spec.arg (field in readers/rst), 302
- Dir_Spec.content (field in readers/rst), 302
- Dir_Spec.final_ws (field in readers/rst), 302
- Dir_Spec.kind (field in readers/rst), 302
- Dir_Spec.name (field in readers/rst), 302
- Dir_Spec.opts (field in readers/rst), 302
- Dir_Spec.value (field in readers/rst), 302
- Directive (struct in readers/commonmark), 289
- (struct in readers/rst), 302
- Directive.arg (field in readers/commonmark), 290
- (field in readers/rst), 302
- Directive.arg_block (field in readers/rst), 303
- Directive.classes (field in readers/commonmark), 290
- Directive.content (field in readers/commonmark), 290
- (field in readers/rst), 303
- Directive.ext (field in readers/rst), 303
- Directive.name (field in readers/commonmark), 289
- Directive.name_span (field in readers/commonmark), 289
- (field in readers/rst), 302
- Directive.nopts (field in readers/rst), 302
- Directive.options (field in readers/commonmark), 290
- Directive.opts (field in readers/rst), 302
- Directive.sections (field in readers/commonmark), 290
- (field in readers/rst), 303
- Directive.source (field in readers/commonmark), 290
- Directive.span (field in readers/commonmark), 290
- (field in readers/rst), 302
- Directive.spec (field in readers/rst), 302
- Directive.subst (field in readers/rst), 303
- Directive.subst_name (field in readers/rst), 303
- Directive_Alias (struct in readers/sphinx), 328
- Directive_Alias.name (field in readers/sphinx), 328
- Directive_Alias.target (field in readers/sphinx), 328
- Directive_Handler (struct in readers/commonmark), 290
- (struct in readers/rst), 303
- Directive_Handler.arg (field in readers/rst), 303

- Directive_Handler.content (field in readers/rst), 303
 - Directive_Handler.final_ws (field in readers/rst), 303
 - Directive_Handler.name (field in readers/commonmark), 290
(field in readers/rst), 303
 - Directive_Handler.options (field in readers/rst), 303
 - Directive_Handler.run (field in readers/commonmark), 290
(field in readers/rst), 303
 - directive_node (procedure in readers/rst), 313
 - Directive_Proc (type in readers/commonmark), 290
(type in readers/rst), 303
 - Directive_Result (enum in readers/commonmark), 290
 - Directive_Result.Failed (enumerator in readers/commonmark), 290
 - Directive_Result.Ran (enumerator in readers/commonmark), 290
 - Directive_Result.Unknown (enumerator in readers/commonmark), 290
 - directive_shape (procedure in readers/rst), 313
 - Directive_Shape (struct in readers/commonmark), 291
(struct in readers/rst), 303
 - Directive_Shape.arg (field in readers/rst), 304
 - Directive_Shape.argument (field in readers/commonmark), 291
 - Directive_Shape.content (field in readers/rst), 304
 - Directive_Shape.final_ws (field in readers/rst), 304
 - Directive_Shape.options (field in readers/commonmark), 291
(field in readers/rst), 304
 - DIRECTIVE_UNKNOWN (constant in readers/rst), 318
 - discharge (procedure in host), 205
 - Disk (struct in host), 193
 - Disk.chunks (field in host), 194
 - Disk.dir (field in host), 193
 - Disk.failure (field in host), 194
 - Disk.limit (field in host), 193
 - Disk.mutex (field in host), 194
 - Disk.peak (field in host), 193
 - Disk.refused (field in host), 194
 - Disk.used (field in host), 193
 - disk_allocator (procedure in host), 206
 - DISK_CHUNK_BYTES (constant in host), 215
 - disk_destroy (procedure in host), 206
 - Disk_Failure (enum in host), 194
 - Disk_Failure.Limit (enumerator in host), 194
 - Disk_Failure.None (enumerator in host), 194
 - Disk_Failure.Space (enumerator in host), 194
 - Disk_Failure.System (enumerator in host), 194
 - DISK_MARGIN (constant in host), 215
 - disk_owns (procedure in host), 206
 - display_path (procedure in host), 206
 - Doc (struct in pyscan), 278
 - Doc.ok (field in pyscan), 278
 - Doc.text (field in pyscan), 278
 - document (procedure in autodoc), 80
(procedure in odindoc), 249
 - Document (struct in core), 89
 - Document.bank (field in core), 89
 - Document.generation (field in core), 89
 - Document.revision (field in core), 89
 - Document.ws (field in core), 89
 - Document_Reading (enum in core), 90
 - Document_Reading.Docutils (enumerator in core), 90
 - Document_Reading.Sphinx (enumerator in core), 90
 - document_valid (procedure in core), 123
 - domain_for (procedure in i18n), 222
 - domain_plural (procedure in gettext), 155
 - domain_plural_with_context (procedure in gettext), 155
 - domain_text (procedure in gettext), 156
 - domain_with_context (procedure in gettext), 156
 - dump (procedure in host), 206
 - dump_tree (procedure in core), 123
- ## E
- each_chunk (procedure in host), 206
 - edit_plan (procedure in gds), 143
 - embed_begin (procedure in readers/rst), 313
 - embed_end (procedure in readers/rst), 314
 - EMBEDDED (constant in typst), 343
 - embedded_catalog (procedure in i18n), 222
 - emit (procedure in core), 123
 - emit_byte (procedure in core), 123
 - emit_int (procedure in core), 123
 - emit_mark (procedure in core), 124
 - Emit_Mode (enum in core), 90
 - Emit_Mode.Measure (enumerator in core), 90
 - Emit_Mode.Write (enumerator in core), 90
 - emit_preamble (procedure in render/typst), 333
 - emit_rune (procedure in core), 124
 - Emitter (struct in core), 90
 - Emitter.mapped (field in core), 90
 - Emitter.mode (field in core), 90
 - Emitter.out (field in core), 90
 - Emitter.source_map (field in core), 90
 - Emitter.written (field in core), 90
 - encode_rune (procedure in core), 124
 - end_byte (procedure in treesitter), 336
 - end_node (procedure in core), 124
 - end_point (procedure in treesitter), 336
 - Engine (enum in graphdog), 162
 - Engine.Circo (enumerator in graphdog), 162
 - Engine.Dot (enumerator in graphdog), 162
 - Engine.Fdp (enumerator in graphdog), 162
 - Engine.Neato (enumerator in graphdog), 162

- Engine.Osage (enumerator in graphdog), 162
- Engine.Patchwork (enumerator in graphdog), 162
- Engine.Sfdp (enumerator in graphdog), 162
- Engine.Twopi (enumerator in graphdog), 162
- engine_name (procedure in graphdog), 165
- ENGLISH_LABELS (variable in core), 136
- enter_source (procedure in readers/rst), 314
- Environment (struct in jinja), 230
- Environment allocator (field in jinja), 230
- Environment.arena (field in jinja), 230
- Environment.backing (field in jinja), 230
- Environment.cache (field in jinja), 230
- Environment.filters (field in jinja), 230
- Environment.globals (field in jinja), 230
- Environment.options (field in jinja), 230
- Environment.out_of_memory (field in jinja), 230
- Environment.stack (field in jinja), 230
- Environment.tests (field in jinja), 230
- equal (procedure in treesitter), 336
- Error (enum in fetch), 137
 - (enum in inventory), 225
 - (struct in gettext), 146
 - (struct in graphdog), 163
 - (struct in jinja), 230
- Error.Bad_Compression (enumerator in inventory), 226
- Error.column (field in gettext), 146
 - (field in graphdog), 163
- Error.excerpt (field in gettext), 147
 - (field in jinja), 231
- Error.file (field in gettext), 146
- Error.hint (field in gettext), 147
 - (field in graphdog), 163
 - (field in jinja), 231
- Error.kind (field in gettext), 146
 - (field in graphdog), 163
 - (field in jinja), 230
- Error.line (field in gettext), 146
 - (field in graphdog), 163
 - (field in jinja), 230
- Error.message (field in gettext), 147
 - (field in graphdog), 163
 - (field in jinja), 230
- Error.Missing (enumerator in fetch), 137
- Error.near (field in graphdog), 163
- Error.Network (enumerator in fetch), 137
- Error.None (enumerator in fetch), 137
 - (enumerator in inventory), 225
- Error.Out_Of_Memory (enumerator in inventory), 226
- Error.Status (enumerator in fetch), 138
- Error.template (field in jinja), 230
- Error.Timeout (enumerator in fetch), 138
- Error.Too_Large (enumerator in fetch), 138
- Error.Truncated (enumerator in inventory), 226
- Error.Unknown_Version (enumerator in inventory), 226
- Error.Unsupported (enumerator in fetch), 137
- Error_Kind (enum in gettext), 147
 - (enum in graphdog), 163
 - (enum in jinja), 231
- Error_Kind.Charset (enumerator in gettext), 147
- Error_Kind.Duplicate (enumerator in gettext), 147
- Error_Kind.Empty_Input (enumerator in graphdog), 163
- Error_Kind.Encoding (enumerator in gettext), 147
- Error_Kind.File_Refused (enumerator in graphdog), 164
- Error_Kind.Html_Label (enumerator in graphdog), 163
- Error_Kind.Io (enumerator in gettext), 147
- Error_Kind.Layout (enumerator in graphdog), 164
- Error_Kind.Limit (enumerator in jinja), 231
- Error_Kind.Memory (enumerator in gettext), 147
 - (enumerator in jinja), 231
- Error_Kind.Missing_Engine (enumerator in graphdog), 163
- Error_Kind.Missing_Format (enumerator in graphdog), 163
- Error_Kind.Mo_Format (enumerator in gettext), 147
- Error_Kind.None (enumerator in gettext), 147
 - (enumerator in graphdog), 163
 - (enumerator in jinja), 231
- Error_Kind.Not_Found (enumerator in jinja), 231
- Error_Kind.Not_Initialized (enumerator in graphdog), 164
- Error_Kind.Out_Of_Memory (enumerator in graphdog), 164
- Error_Kind.Plural (enumerator in gettext), 147
- Error_Kind.Render (enumerator in graphdog), 164
- Error_Kind.Runtime (enumerator in jinja), 231
- Error_Kind.Syntax (enumerator in gettext), 147
 - (enumerator in graphdog), 163
 - (enumerator in jinja), 231
- Error_Kind.Type (enumerator in jinja), 231
- Error_Kind.Undefined (enumerator in jinja), 231
- error_title (procedure in graphdog), 166
- ERROR_TITLES (variable in gettext), 161
- escape (procedure in jinja), 238
- escape_string (procedure in gettext), 156
 - (procedure in jinja), 238
- evaluate (procedure in jinja), 238
- excerpt_at (procedure in host), 206
- exchange (procedure in host), 206
- exit_code (procedure in host), 207
- exported (procedure in pycan), 286
- Extension (struct in readers/
 - commonmark), 291
 - (struct in readers/rst), 304
- Extension.begin (field in readers/
 - commonmark), 291

(field in readers/rst), 304
 Extension.directive (field in readers/
 commonmark), 291
 Extension.directives (field in readers/
 commonmark), 291
 (field in readers/rst), 304
 Extension.end (field in readers/
 commonmark), 291
 (field in readers/rst), 304
 Extension.lookup (field in readers/rst), 304
 Extension.prologue (field in readers/rst),
 304
 Extension.resolve (field in readers/
 commonmark), 292
 Extension.role (field in readers/
 commonmark), 291
 (field in readers/rst), 304
 Extension.roles (field in readers/
 commonmark), 291
 (field in readers/rst), 304
 Extension.shape (field in readers/
 commonmark), 291
 Extension.substitution (field in readers/
 rst), 304
 Extension.table (field in readers/
 commonmark), 291
 Extension.user (field in readers/
 commonmark), 292
 (field in readers/rst), 305
 extension_as (procedure in core), 124
 Extlink (struct in readers/sphinx), 328
 Extlink.caption (field in readers/sphinx),
 328
 Extlink.name (field in readers/sphinx), 328
 Extlink.url (field in readers/sphinx), 328
 extra_anchor (procedure in core), 124
 extra_text (procedure in core), 124
 extract (procedure in i18n), 223
 Extract_Options (struct in i18n), 217
 Extract_Options.base (field in i18n), 217
 Extract_Options.docname (field in i18n),
 217
 Extract_Options.location (field in i18n),
 217
 Extract_Options.path (field in i18n), 217
 Extract_Options.targets (field in i18n), 217
 Extract_Options.uuid (field in i18n), 217

F

fail (procedure in jinja), 239
 fetch (package), 137
 field (procedure in treesitter), 336
 Field (struct in gds), 140
 Field.literal (field in gds), 140
 Field.name (field in gds), 140
 Field_Id (type in treesitter), 334
 field_name_for_child (procedure in
 treesitter), 336
 figure_filename (procedure in i18n), 223
 figure_for_language (procedure in i18n),
 223
 File_Access (struct in graphdog), 164
 File_Access.procedure (field in graphdog),
 164
 File_Access.user (field in graphdog), 164

FILE_ATTRIBUTES (constant in
 graphdog), 167
 File_Read (struct in odindoc), 244
 (struct in pyscan), 278
 File_Read.path (field in odindoc), 244
 (field in pyscan), 278
 File_Read.text (field in odindoc), 244
 (field in pyscan), 278
 File_Reader (struct in odindoc), 244
 (struct in pyscan), 278
 File_Reader.list (field in odindoc), 244
 File_Reader.procedure (field in odindoc),
 244
 (field in pyscan), 278
 File_Reader.reserve (field in odindoc), 244
 File_Reader.user (field in odindoc), 244
 (field in pyscan), 278
 Files (struct in host), 194
 Files.canonical (field in host), 195
 Files.ledger (field in host), 195
 Files.max_bytes (field in host), 194
 Files.project (field in host), 195
 Files.roots (field in host), 194
 files_provider (procedure in host), 207
 filesystem_loader (procedure in jinja), 239
 fill_labels (procedure in i18n), 223
 find (procedure in gettext), 156
 (procedure in inventory), 227
 find_catalog_file (procedure in gettext),
 156
 find_def (procedure in pyscan), 286
 find_folded (procedure in inventory), 227
 find_lexer (procedure in highlight), 184
 find_lexer_for_filename (procedure in
 highlight), 184
 find_program (procedure in typst), 342
 find_reader (procedure in core), 124
 find_renderer (procedure in core), 124
 find_role (procedure in readers/rst), 314
 find_style (procedure in highlight), 184
 first_child (procedure in core), 124
 first_problem (procedure in core), 125
 fitting_budget (procedure in host), 207
 Flag_Or_Name (struct in project), 264
 Flag_Or_Name.name (field in project), 264
 Flag_Or_Name.on (field in project), 264
 float (procedure in jinja), 239
 Folder_Entry (struct in host), 195
 Folder_Entry.inode (field in host), 195
 Folder_Entry.name (field in host), 195
 Folder_Entry.type (field in host), 195
 folders_of (procedure in host), 207
 footnote_symbol (procedure in readers/rst/
 inline), 323
 Format (enum in graphdog), 164
 (enum in host), 195
 Format.Commonmark (enumerator in
 host), 195
 Format.Dot (enumerator in graphdog), 164
 Format.Png (enumerator in graphdog), 164
 Format.Rst (enumerator in host), 195
 Format.Svg (enumerator in graphdog), 164
 Format.Typst (enumerator in host), 195
 format_date (procedure in i18n), 223
 format_error (procedure in gettext), 156
 (procedure in graphdog), 166
 (procedure in jinja), 239

format_message (procedure in graphdog),
166
format_stats (procedure in gettext), 156
Frame (struct in readers/rst), 305
Frame.src (field in readers/rst), 305
Frame.tab (field in readers/rst), 305
Frame.text (field in readers/rst), 305
FULL_VERSION (constant in cli), 81
function (procedure in jinja), 239
Function (struct in pyscan), 278
Function.class (field in pyscan), 279
Function.decorators (field in pyscan), 279
Function.doc (field in pyscan), 279
Function.is_async (field in pyscan), 279
Function.is_lambda (field in pyscan), 279
Function.module (field in pyscan), 278
Function.name (field in pyscan), 278
Function.node (field in pyscan), 278
Function.params (field in pyscan), 279
Function.qualname (field in pyscan), 278
Function.returns (field in pyscan), 279
Function.type_args (field in pyscan), 279
Function.type_comment (field in pyscan),
279
Function.type_returns (field in pyscan),
279
FUZZY_MIN_WORK (constant in gettext),
160
FUZZY_THRESHOLD (constant in
gettext), 160
FUZZY_WORK_PER_BYTE (constant in
gettext), 161

G

gds (package), 139
get (procedure in fetch), 139
(procedure in jinja), 239
get_attr (procedure in pyscan), 286
get_opt (procedure in readers/rst), 314
gettext (package), 145
graphdog (package), 161
graphviz (foreign import in graphdog), 168
grow_session (procedure in host), 207
Guard (struct in host), 195
Guard.backing (field in host), 196
Guard.refused (field in host), 196
Guard.tripped (field in host), 196
guard_allocator (procedure in host), 207
guard_problem (procedure in host), 207
guard_tripped (procedure in host), 208
guess_lexer (procedure in highlight), 185

H

has_arg (procedure in jinja), 239
has_children (procedure in core), 125
has_class (procedure in core), 125
has_error (procedure in treesitter), 336
has_errors (procedure in highlight), 185
has_option (procedure in readers/rst), 314
head (procedure in readers/rst), 314
header_field (procedure in gettext), 157
Header_Field (struct in gettext), 147
Header_Field.name (field in gettext), 148
Header_Field.value (field in gettext), 148
highlight (package), 168
highlight/guidedoc (package), 188
highlight_html (procedure in highlight),
185

highlight_html_lines (procedure in
highlight), 185
highlight_typst (procedure in highlight),
185
highlighter (procedure in highlight/
guidedoc), 188
Highlighter (struct in core), 90
Highlighter.procedure (field in core), 91
Highlighter.user (field in core), 91
Hook_Proc (type in readers/rst), 305
host (package), 189
Host (struct in readers/commonmark), 292
(struct in readers/rst), 305
Host.b (field in readers/commonmark), 292
Host.body (field in readers/rst), 305
Host.ctx (field in readers/commonmark),
292
Host.disabled (field in readers/
commonmark), 292
Host.ext (field in readers/commonmark),
292
Host.html_format (field in readers/
commonmark), 293
Host.inline (field in readers/rst), 305
Host.items (field in readers/rst), 305
Host.label_tmp (field in readers/
commonmark), 293
Host.labels (field in readers/
commonmark), 292
Host.myst (field in readers/commonmark),
292
Host.names (field in readers/
commonmark), 292
Host.nrefs (field in readers/commonmark),
292
Host.refs (field in readers/commonmark),
292
Host.rel_docs (field in readers/
commonmark), 293
Host.rel_images (field in readers/
commonmark), 293
Host.settings (field in readers/
commonmark), 292
Host.syntax (field in readers/
commonmark), 292
Host.user (field in readers/rst), 305
Host.using (field in readers/commonmark),
293
Host_Directive (struct in readers/rst), 305
Host_Directive.arg (field in readers/rst),
306
Host_Directive.content (field in readers/
rst), 306
Host_Directive.name (field in readers/rst),
306
Host_Directive.name_span (field in
readers/rst), 306
Host_Directive.options (field in readers/
rst), 306
Host_Directive.sections (field in readers/
rst), 306
Host_Directive.span (field in readers/rst),
306

host_lines (procedure in readers/rst), 314
 Host_Option (struct in readers/rst), 306
 Host_Option.name (field in readers/rst), 306
 Host_Option.span (field in readers/rst), 306
 Host_Option.value (field in readers/rst), 306
 host_views (procedure in readers/rst), 314
 Hosted_Result (enum in readers/rst), 306
 Hosted_Result.Failed (enumerator in readers/rst), 306
 Hosted_Result.Ran (enumerator in readers/rst), 306
 Hosted_Result.Unknown (enumerator in readers/rst), 306
 Html_Lines (struct in highlight), 169
 Html_Lines.emphasized (field in highlight), 169
 Html_Lines.first (field in highlight), 169
 Html_Lines.user (field in highlight), 169

I

i18n (package), 216
 id_fold (procedure in core), 125
 identity (procedure in host), 208
 Identity (struct in host), 196
 Identity.device (field in host), 196
 Identity.inode (field in host), 196
 Image_Exists (struct in i18n), 217
 Image_Exists.procedure (field in i18n), 217
 Image_Exists.user (field in i18n), 218
 Image_Localization (struct in i18n), 218
 Image_Localization.docname (field in i18n), 218
 Image_Localization.exists (field in i18n), 218
 Image_Localization.format (field in i18n), 218
 Image_Localization.language (field in i18n), 218
 Import (struct in odindoc), 245
 Import.alias (field in odindoc), 245
 Import.id (field in odindoc), 245
 INCLUDE_MISSING (constant in core), 136
 include_paths (procedure in readers/commonmark), 299
 include_source (procedure in core), 125
 indent_of (procedure in readers/rst), 314
 Index (struct in gettext), 148
 Index.built (field in gettext), 148
 Index.entries (field in gettext), 148
 Index.forms (field in gettext), 148
 Index.fuzzy (field in gettext), 148
 Index.pool (field in gettext), 148
 Index.slots (field in gettext), 148
 Index_Entry (struct in gettext), 148
 Index_Entry.count (field in gettext), 148
 Index_Entry.first (field in gettext), 148
 Index_Entry.message (field in gettext), 148
 Index_Part (struct in i18n), 218
 Index_Part.first (field in i18n), 218
 Index_Part.length (field in i18n), 218
 Index_Part.text (field in i18n), 218
 index_parts (procedure in i18n), 223
 index_plan (procedure in gds), 143
 init (procedure in graphdog), 166
 (procedure in odindoc), 249
 (procedure in pyscan), 286
 init_workspace (procedure in core), 125
 init_workspace_from_bytes (procedure in core), 125
 init_workspace_with_plan (procedure in core), 125
 inline (procedure in readers/rst), 314
 inline_node (procedure in readers/rst), 315
 Inline_State (struct in readers/rst/inline), 319
 Inline_State.ctx (field in readers/rst/inline), 319
 Inline_State.default_role (field in readers/rst/inline), 319
 Inline_State.hook (field in readers/rst/inline), 319
 Inline_State.options (field in readers/rst/inline), 319
 Inline_State.roles (field in readers/rst/inline), 319
 Inline_State.roles_used (field in readers/rst/inline), 319
 Inline_State.source (field in readers/rst/inline), 319
 install_crash_reporter (procedure in host), 208
 integer (procedure in jinja), 240
 interrupted_problem (procedure in host), 208
 Intersphinx_Project (struct in project), 265
 Intersphinx_Project.locations (field in project), 265
 Intersphinx_Project.name (field in project), 265
 Intersphinx_Project.uri (field in project), 265
 Intl_Action (enum in project), 265
 Intl_Action.Create (enumerator in project), 265
 Intl_Action.Not_Changed (enumerator in project), 265
 Intl_Action.Update (enumerator in project), 265
 Intl_File (struct in project), 265
 Intl_File.action (field in project), 265
 Intl_File.path (field in project), 265
 Intl_File.stats (field in project), 265
 Intl_Options (struct in project), 265
 Intl_Options.languages (field in project), 266
 Intl_Options.locale_dir (field in project), 265
 Intl_Options.obsolete (field in project), 266
 Intl_Options.pot_dir (field in project), 266
 Intl_Options.width (field in project), 266
 intl_stat (procedure in project), 272
 intl_update (procedure in project), 272
 INVALID_NODE (type in core), 91
 inventory (package), 225
 Inventory (struct in inventory), 226
 Inventory.by_type (field in inventory), 226
 Inventory.folded (field in inventory), 226

Inventory.items (field in inventory), 226
 Inventory.project (field in inventory), 226
 Inventory.version (field in inventory), 226
 io_problem (procedure in host), 208
 is_ascii_alnum (procedure in core), 126
 is_ascii_digit (procedure in core), 126
 is_ascii_punct (procedure in core), 126
 is_ascii_space (procedure in core), 126
 is_builtin (procedure in pyscan), 287
 is_builtin_exception (procedure in
 pyscan), 287
 is_error (procedure in treesitter), 336
 is_exception (procedure in pyscan), 287
 is_extra (procedure in treesitter), 336
 is_indented (procedure in readers/rst), 315
 is_markup (procedure in jinja), 240
 is_missing (procedure in treesitter), 337
 is_named (procedure in treesitter), 337
 is_none (procedure in jinja), 240
 is_null (procedure in treesitter), 337
 is_stage_name (procedure in host), 208
 is_undefined (procedure in jinja), 240
 is_unicode_punctuation (procedure in
 core), 126
 is_unicode_whitespace (procedure in
 core), 126
 ISSUE_TRACKER (constant in host), 215
 Item (struct in inventory), 226
 Item.display (field in inventory), 227
 Item.name (field in inventory), 226
 Item.priority (field in inventory), 227
 Item.project (field in inventory), 226
 Item.type (field in inventory), 226
 Item.uri (field in inventory), 227
 Item.version (field in inventory), 226
 Iterator (struct in treesitter), 334
 Iterator.count (field in treesitter), 334
 Iterator.index (field in treesitter), 334
 Iterator.named (field in treesitter), 334
 Iterator.node (field in treesitter), 334

J

jinja (package), 228
 join (procedure in project), 272
 join_lines (procedure in readers/rst), 315

K

kept_message (procedure in core), 126
 kernel32 (foreign import in host), 216
 keys (procedure in jinja), 240
 Keyword (struct in jinja), 231
 (struct in pyscan), 279
 Keyword.name (field in jinja), 231
 (field in pyscan), 279
 Keyword.value (field in jinja), 231
 (field in pyscan), 279
 Kind (enum in odindoc), 245
 kind (procedure in core), 126
 (procedure in treesitter), 337
 Kind.Bit_Field (enumerator in odindoc),
 245
 Kind.Bit_Set (enumerator in odindoc), 245
 Kind.Const (enumerator in odindoc), 245
 Kind.Enum (enumerator in odindoc), 245
 Kind.Foreign_Import (enumerator in
 odindoc), 245
 Kind.Proc (enumerator in odindoc), 245

Kind.Proc_Group (enumerator in odindoc),
 245
 Kind.Struct (enumerator in odindoc), 245
 Kind.Type (enumerator in odindoc), 245
 Kind.Union (enumerator in odindoc), 245
 Kind.Var (enumerator in odindoc), 245
 kind_class (procedure in highlight), 185
 kind_is (procedure in highlight), 185
 kind_name (procedure in core), 126
 (procedure in highlight), 185
 kind_parent (procedure in highlight), 185

L

Label (enum in core), 91
 label (procedure in core), 126
 Label.Arguments (enumerator in core), 93
 Label.Attention (enumerator in core), 91
 Label.Availability (enumerator in core), 93
 Label.Caution (enumerator in core), 91
 Label.Code_Author (enumerator in core),
 92
 Label.Contents (enumerator in core), 91
 Label.Danger (enumerator in core), 91
 Label.Error (enumerator in core), 91
 Label.Hint (enumerator in core), 91
 Label.Important (enumerator in core), 91
 Label.Link_Code (enumerator in core), 92
 Label.Link_Definition (enumerator in
 core), 92
 Label.Link_Equation (enumerator in core),
 92
 Label.Link_Heading (enumerator in core),
 92
 Label.Link_Image (enumerator in core), 92
 Label.Link_Table (enumerator in core), 92
 Label.Link_Term (enumerator in core), 92
 Label.Module_Author (enumerator in
 core), 92
 Label.Note (enumerator in core), 91
 Label.Parameters (enumerator in core), 92
 Label.Raisees (enumerator in core), 92
 Label.Return_Type (enumerator in core),
 92
 Label.Return_Values (enumerator in core),
 92
 Label>Returns (enumerator in core), 92
 Label.Section_Author (enumerator in
 core), 92
 Label.See_Also (enumerator in core), 91
 Label.Template_Parameters (enumerator in
 core), 93
 Label.Throws (enumerator in core), 93
 Label.Tip (enumerator in core), 91
 Label.TODO (enumerator in core), 91
 Label.Variables (enumerator in core), 92
 Label.Version_Added (enumerator in core),
 91
 Label.Version_Changed (enumerator in
 core), 92
 Label.Version_Deprecated (enumerator in
 core), 92
 Label.Version_Removed (enumerator in
 core), 92
 Label.Warning (enumerator in core), 91
 LABELS (constant in gds), 145
 Labels (type in core), 93

- Language (struct in treesitter), 334
- language_candidates (procedure in gettext), 157
- leave_source (procedure in readers/rst), 315
- Ledger (struct in host), 196
- Ledger.budget (field in host), 196
- Ledger.held (field in host), 196
- Ledger.refused (field in host), 196
- length (procedure in jinja), 240
- Lexer (enum in highlight), 169
- Lexer.Bash (enumerator in highlight), 170
- Lexer.C (enumerator in highlight), 169
- Lexer.Console (enumerator in highlight), 170
- Lexer.Cpp (enumerator in highlight), 169
- Lexer.Csharp (enumerator in highlight), 170
- Lexer.Css (enumerator in highlight), 170
- Lexer.Default (enumerator in highlight), 171
- Lexer.Diff (enumerator in highlight), 171
- Lexer.Dockerfile (enumerator in highlight), 171
- Lexer.Dot (enumerator in highlight), 171
- Lexer.Go (enumerator in highlight), 170
- Lexer.Guess (enumerator in highlight), 171
- Lexer.Html (enumerator in highlight), 170
- Lexer.Ini (enumerator in highlight), 170
- Lexer.Java (enumerator in highlight), 170
- Lexer.Javascript (enumerator in highlight), 170
- Lexer.Json (enumerator in highlight), 170
- Lexer.Kotlin (enumerator in highlight), 170
- Lexer.Latex (enumerator in highlight), 171
- Lexer.Makefile (enumerator in highlight), 171
- Lexer.Markdown (enumerator in highlight), 171
- Lexer.Odin (enumerator in highlight), 170
- Lexer.Powershell (enumerator in highlight), 171
- Lexer.Pycon (enumerator in highlight), 169
- Lexer.Pytb (enumerator in highlight), 169
- Lexer.Python (enumerator in highlight), 169
- Lexer.Rst (enumerator in highlight), 171
- Lexer.Rust (enumerator in highlight), 170
- Lexer.Sql (enumerator in highlight), 171
- Lexer.Swift (enumerator in highlight), 170
- Lexer.Text (enumerator in highlight), 169
- Lexer.Toml (enumerator in highlight), 170
- Lexer.Typescript (enumerator in highlight), 170
- Lexer.Typst (enumerator in highlight), 171
- Lexer.Xml (enumerator in highlight), 170
- Lexer.Yaml (enumerator in highlight), 170
- Lexer.Zig (enumerator in highlight), 170
- lexer_alias (procedure in highlight), 185
- lexer_info (procedure in highlight), 186
- Lexer_Info (struct in highlight), 171
- Lexer_Info.aliases (field in highlight), 171
- Lexer_Info.fileNames (field in highlight), 171
- Lexer_Info.mimetypes (field in highlight), 171
- Lexer_Info.name (field in highlight), 171
- Lexer_Info.spec (field in highlight), 172
- Lexer_Info.step (field in highlight), 171
- lexer_name (procedure in highlight), 186
- LICENSE_NOTICE (constant in cli), 81
- Limits (struct in core), 93
- Limits.max_depth (field in core), 93
- Limits.max_expansion (field in core), 93
- Limits.max_include_depth (field in core), 93
- Limits.max_included_bytes (field in core), 93
- Limits.max_nodes (field in core), 93
- Limits.max_work (field in core), 93
- Limits.stack_bytes (field in core), 93
- Line (struct in readers/rst), 306
 - (type in readers/commonmark), 293
 - (type in readers/rst/inline), 319
- Line.col (field in readers/rst), 307
- Line.offset (field in readers/rst), 307
- Line.raw (field in readers/rst), 307
- Line.text (field in readers/rst), 307
- line_bounds (procedure in core), 127
- line_column (procedure in core), 127
- line_span (procedure in readers/rst), 315
- Line_View (struct in core), 93
- Line_View.offset (field in core), 94
- Line_View.text (field in core), 93
- lines_span (procedure in readers/rst), 315
- link_file (procedure in host), 208
- link_info (procedure in core), 127
- Link_Info (struct in core), 94
- Link_Info.destination (field in core), 94
- Link_Info.kind (field in core), 94
- Link_Info.name (field in core), 94
- Link_Info.resolved (field in core), 94
- Link_Info.title (field in core), 94
- Link_Kind (enum in core), 94
- Link_Kind.Email (enumerator in core), 94
- Link_Kind.External (enumerator in core), 94
- Link_Kind.Internal (enumerator in core), 94
- Link_Kind.Project (enumerator in core), 94
- Link_Kind.Unresolved (enumerator in core), 94
- list (procedure in jinja), 240
- List (struct in jinja), 232
- List allocator (field in jinja), 232
- List.fields (field in jinja), 232
- List.items (field in jinja), 232
- List.tuple (field in jinja), 232
- list_folder (procedure in host), 209
- list_info (procedure in core), 127
- List_Info (struct in core), 94
- List_Info.delimiter (field in core), 95
- List_Info.start (field in core), 95
- List_Info.style (field in core), 94
- list_items (procedure in readers/commonmark), 299
- list_packages (procedure in odindoc), 249
- List_Style (enum in core), 95
- List_Style.Arabic (enumerator in core), 95
- List_Style.Bullet (enumerator in core), 95
- List_Style.Lower_Alpha (enumerator in core), 95
- List_Style.Lower_Roman (enumerator in core), 95

List_Style.Upper_Alpha (enumerator in core), 95
 List_Style.Upper_Roman (enumerator in core), 95
 LIST_STYLE_NAMES (variable in core), 136
 load (procedure in gds), 143
 (procedure in inventory), 227
 (procedure in odindoc), 250
 (procedure in project), 273
 load_mo (procedure in gettext), 157
 load_object (procedure in core), 127
 Load_Options (struct in project), 266
 Load_Options.conf_dir (field in project), 266
 Load_Options.no_config (field in project), 266
 Load_Options.output (field in project), 266
 Load_Options.overrides (field in project), 266
 Load_Options.source (field in project), 266
 load_po (procedure in gettext), 157
 load_template (procedure in jinja), 240
 Loader (struct in jinja), 232
 Loader.destroy (field in jinja), 232
 Loader.procedure (field in jinja), 232
 Loader.user (field in jinja), 232
 Loader_Proc (type in jinja), 232
 locate (procedure in typst), 342
 lock (procedure in gds), 143
 Look (struct in highlight), 172
 Look.background (field in highlight), 172
 Look.bold (field in highlight), 172
 Look.border (field in highlight), 172
 Look.color (field in highlight), 172
 Look.italic (field in highlight), 172
 Look.underline (field in highlight), 172
 lookup_name (procedure in pyscan), 287
 Loop (struct in jinja), 232
 Loop.changed (field in jinja), 233
 Loop.depth0 (field in jinja), 232
 Loop.index0 (field in jinja), 232
 Loop.items (field in jinja), 232
 Loop.last_seen (field in jinja), 232
 Loop.recurse (field in jinja), 233

M

main_stack_bytes (procedure in host), 209
 make_catalog (procedure in gettext), 157
 make_id (procedure in core), 127
 make_storage (procedure in host), 209
 make_template (procedure in i18n), 224
 make_translator (procedure in gettext), 157
 Map_Entry (struct in core), 95
 Map_Entry.offset (field in core), 95
 Map_Entry.span (field in core), 95
 mark (procedure in readers/rst), 315
 Markdown (struct in readers/sphinx), 328
 Markdown.config (field in readers/sphinx), 329
 Markdown.ext (field in readers/sphinx), 329
 Markdown.host (field in readers/sphinx), 329
 Markdown.parser (field in readers/sphinx), 329

Markdown.reader (field in readers/sphinx), 329
 Markdown.settings (field in readers/sphinx), 329
 Markdown.state (field in readers/sphinx), 329
 markdown_settings (procedure in readers/sphinx), 330
 markup (procedure in jinja), 240
 Markup (type in jinja), 233
 Math_Style (enum in core), 95
 Math_Style.Plain (enumerator in core), 95
 Math_Style.Sphinx (enumerator in core), 95
 MAX_OBJECT_TYPES (constant in readers/sphinx), 331
 MAX_RETRIES (constant in host), 216
 MAX_SOURCE_BYTES (constant in host), 216
 Media_Kind (enum in core), 96
 media_kind (procedure in core), 127
 Media_Kind.Audio (enumerator in core), 96
 Media_Kind.Image (enumerator in core), 96
 Media_Kind.Video (enumerator in core), 96
 Member (struct in odindoc), 246
 Member.docs (field in odindoc), 246
 Member.kind (field in odindoc), 246
 Member.name (field in odindoc), 246
 Member.signature (field in odindoc), 246
 Member_Kind (enum in odindoc), 246
 Member_Kind.Bit_Field (enumerator in odindoc), 246
 Member_Kind.Enumerator (enumerator in odindoc), 246
 Member_Kind.Field (enumerator in odindoc), 246
 MEMORY_HINT (constant in host), 216
 memory_loader (procedure in jinja), 240
 merge (procedure in gettext), 157
 Merge_Options (struct in gettext), 148
 Merge_Options.fuzzy_matching (field in gettext), 148
 Merge_Options.keep_obsolete (field in gettext), 149
 Merge_Options.stats (field in gettext), 149
 Merge_Stats (struct in gettext), 149
 Merge_Stats.bounds (field in gettext), 149
 Merge_Stats.comparisons (field in gettext), 149
 Merge_Stats.cut_short (field in gettext), 149
 Merge_Stats.searches (field in gettext), 149
 Merge_Stats.work (field in gettext), 149
 Message (struct in autodoc), 78
 (struct in core), 96
 (struct in gettext), 149
 (struct in odindoc), 246
 Message.code (field in autodoc), 78
 (field in core), 96
 (field in odindoc), 246
 Message.comments (field in gettext), 150
 Message.ctx (field in gettext), 149
 Message.flags (field in gettext), 150
 Message.fuzzy (field in gettext), 150

- Message.has_ctx (field in gettext), 149
 - Message.has_plural (field in gettext), 150
 - Message.has_previous (field in gettext), 150
 - Message.hint (field in autodoc), 78
 - (field in core), 96
 - (field in odindoc), 246
 - Message.id (field in gettext), 149
 - Message.id_plural (field in gettext), 149
 - Message.line (field in gettext), 150
 - Message.message (field in core), 96
 - Message.notes (field in gettext), 150
 - Message.obsolete (field in gettext), 150
 - Message.previous (field in gettext), 150
 - Message.references (field in gettext), 150
 - Message.severity (field in autodoc), 78
 - (field in core), 96
 - (field in odindoc), 246
 - Message.status (field in core), 96
 - Message.strs (field in gettext), 150
 - Message.text (field in autodoc), 78
 - (field in odindoc), 246
 - Message.title (field in autodoc), 78
 - (field in core), 96
 - (field in odindoc), 246
 - Meta (struct in readers/commonmark), 293
 - Meta.content (field in readers/commonmark), 293
 - Meta.key (field in readers/commonmark), 293
 - Meter (struct in host), 197
 - Meter.backing (field in host), 197
 - Meter.ledger (field in host), 197
 - Meter.prepaid (field in host), 197
 - meter_allocator (procedure in host), 209
 - meter_prepay (procedure in host), 209
 - meter_refund (procedure in host), 209
 - metered_refusal (procedure in host), 209
 - migrate (procedure in project), 273
 - Migrate_Options (struct in project), 266
 - Migrate_Options.force (field in project), 266
 - Migrate_Options.input (field in project), 266
 - Migrate_Options.output (field in project), 266
 - Migrate_Options.year (field in project), 267
 - Migrate_Result (struct in project), 267
 - Migrate_Result.attention (field in project), 267
 - Migrate_Result.converted (field in project), 267
 - Migrate_Result.input (field in project), 267
 - Migrate_Result.notes (field in project), 267
 - Migrate_Result.output (field in project), 267
 - Migrate_Result.toml (field in project), 267
 - module (procedure in pyscan), 287
 - Module (struct in pyscan), 279
 - Module.all (field in pyscan), 280
 - Module.broken (field in pyscan), 280
 - Module.doc (field in pyscan), 280
 - Module.info (field in pyscan), 280
 - Module.kind (field in pyscan), 279
 - Module.name (field in pyscan), 279
 - Module.origin (field in pyscan), 280
 - Module.path (field in pyscan), 280
 - Module.project (field in pyscan), 279
 - Module.root (field in pyscan), 280
 - Module.scope (field in pyscan), 280
 - Module.source (field in pyscan), 280
 - Module.too_deep (field in pyscan), 280
 - Module.tree (field in pyscan), 280
 - module_attr (procedure in pyscan), 287
 - Module_Kind (enum in pyscan), 280
 - Module_Kind.Namespace (enumerator in pyscan), 281
 - Module_Kind.Native (enumerator in pyscan), 281
 - Module_Kind.Package (enumerator in pyscan), 280
 - Module_Kind.Source (enumerator in pyscan), 280
 - module_object (procedure in pyscan), 287
 - mro (procedure in pyscan), 287
 - Mro_Entry (struct in pyscan), 281
 - Mro_Entry.class (field in pyscan), 281
 - Mro_Entry.path (field in pyscan), 281
 - MYST_CONFORMANCE (constant in readers/commonmark), 301
 - myst_options (procedure in defaults), 137
 - myst_reader (procedure in readers/commonmark), 299
 - MYST_VERSION (constant in readers/commonmark), 301
- ## N
- named_child (procedure in treesitter), 337
 - named_child_count (procedure in treesitter), 337
 - Native (type in jinja), 233
 - new_message (procedure in gettext), 158
 - new_plan (procedure in gds), 143
 - next_anchor_id (procedure in core), 127
 - next_attribute (procedure in core), 128
 - next_child (procedure in core), 128
 - (procedure in treesitter), 337
 - next_named_sibling (procedure in treesitter), 337
 - next_number (procedure in gds), 144
 - next_sibling (procedure in treesitter), 337
 - next_token (procedure in highlight), 186
 - nfd_first (procedure in core), 128
 - no_reader_settings (procedure in core), 128
 - node (procedure in core), 128
 - Node (struct in core), 96
 - (struct in treesitter), 334
 - Node.attributes (field in core), 97
 - Node.ctx (field in treesitter), 334
 - Node.flags (field in core), 96
 - Node.id (field in treesitter), 334
 - Node.kind (field in core), 96
 - Node.payload (field in core), 97
 - Node.span (field in core), 97
 - Node.subtree_end (field in core), 96
 - Node.text (field in core), 96
 - Node.tree (field in treesitter), 334
 - Node.value (field in core), 96
 - node_count (procedure in core), 128
 - Node_Flag (enum in core), 97
 - Node_Flag.Anonymous (enumerator in core), 97
 - Node_Flag.Auto (enumerator in core), 97

- Node_Flag.Generated (enumerator in core), 97
- Node_Flag.Header (enumerator in core), 97
- Node_Flag.Local (enumerator in core), 97
- Node_Flag.Referenced (enumerator in core), 97
- Node_Flag.Symbol (enumerator in core), 97
- Node_Flag.Tight (enumerator in core), 97
- Node_Flags (bit set in core), 97
- Node_Id (type in core), 97
- Node_Kind (enum in core), 98
- Node_Kind.Abbreviation (enumerator in core), 102
- Node_Kind.Admonition (enumerator in core), 100
- Node_Kind.Attribution (enumerator in core), 98
- Node_Kind.Block_Quote (enumerator in core), 98
- Node_Kind.Caption (enumerator in core), 100
- Node_Kind.Cell (enumerator in core), 100
- Node_Kind.Citation (enumerator in core), 100
- Node_Kind.Citation_Reference (enumerator in core), 101
- Node_Kind.Classifier (enumerator in core), 98
- Node_Kind.Code (enumerator in core), 101
- Node_Kind.Code_Block (enumerator in core), 99
- Node_Kind.Comment (enumerator in core), 100
- Node_Kind.Compound (enumerator in core), 100
- Node_Kind.Container (enumerator in core), 100
- Node_Kind.Contents (enumerator in core), 100
- Node_Kind.Definition (enumerator in core), 98
- Node_Kind.Definition_Item (enumerator in core), 98
- Node_Kind.Definition_List (enumerator in core), 98
- Node_Kind.Desc (enumerator in core), 101
- Node_Kind.Desc_Addname (enumerator in core), 102
- Node_Kind.Desc_Annotation (enumerator in core), 102
- Node_Kind.Desc_Content (enumerator in core), 101
- Node_Kind.Desc_Name (enumerator in core), 102
- Node_Kind.Desc_Optional (enumerator in core), 102
- Node_Kind.Desc_Parameter (enumerator in core), 102
- Node_Kind.Desc_Parameterlist (enumerator in core), 102
- Node_Kind.Desc_Returns (enumerator in core), 102
- Node_Kind.Desc_Signature (enumerator in core), 101
- Node_Kind.Directive (enumerator in core), 100
- Node_Kind.Doctest_Block (enumerator in core), 99
- Node_Kind.Document (enumerator in core), 98
- Node_Kind.Emphasis (enumerator in core), 101
- Node_Kind.Field (enumerator in core), 98
- Node_Kind.Field_Body (enumerator in core), 99
- Node_Kind.Field_List (enumerator in core), 98
- Node_Kind.Field_Name (enumerator in core), 98
- Node_Kind.Figure (enumerator in core), 100
- Node_Kind.Footnote (enumerator in core), 100
- Node_Kind.Footnote_Reference (enumerator in core), 101
- Node_Kind.Glossary (enumerator in core), 101
- Node_Kind.Hard_Break (enumerator in core), 102
- Node_Kind.Heading (enumerator in core), 98
- Node_Kind.Hlist (enumerator in core), 101
- Node_Kind.Image (enumerator in core), 101
- Node_Kind.Index (enumerator in core), 101
- Node_Kind.Inline_Target (enumerator in core), 102
- Node_Kind.Legend (enumerator in core), 100
- Node_Kind.Line (enumerator in core), 99
- Node_Kind.Line_Block (enumerator in core), 99
- Node_Kind.Link (enumerator in core), 101
- Node_Kind.List (enumerator in core), 98
- Node_Kind.List_Item (enumerator in core), 98
- Node_Kind.Math (enumerator in core), 102
- Node_Kind.Math_Block (enumerator in core), 99
- Node_Kind.Only (enumerator in core), 101
- Node_Kind.Option (enumerator in core), 99
- Node_Kind.Option_Argument (enumerator in core), 99
- Node_Kind.Option_Description (enumerator in core), 99
- Node_Kind.Option_Group (enumerator in core), 99
- Node_Kind.Option_Item (enumerator in core), 99

Node_Kind.Option_List (enumerator in core), 99
 Node_Kind.Paragraph (enumerator in core), 98
 Node_Kind.Preformatted (enumerator in core), 99
 Node_Kind.Production (enumerator in core), 101
 Node_Kind.Production_List (enumerator in core), 101
 Node_Kind.Raw_Block (enumerator in core), 99
 Node_Kind.Raw_Inline (enumerator in core), 102
 Node_Kind.Reference (enumerator in core), 101
 Node_Kind.Role (enumerator in core), 102
 Node_Kind.Row (enumerator in core), 99
 Node_Kind.Rubric (enumerator in core), 100
 Node_Kind.Section (enumerator in core), 98
 Node_Kind.Sidebar (enumerator in core), 100
 Node_Kind.Soft_Break (enumerator in core), 102
 Node_Kind.Strong (enumerator in core), 101
 Node_Kind.Subscript (enumerator in core), 102
 Node_Kind.Substitution_Definition (enumerator in core), 100
 Node_Kind.Substitution_Reference (enumerator in core), 102
 Node_Kind.Subtitle (enumerator in core), 98
 Node_Kind.Superscript (enumerator in core), 102
 Node_Kind.Table (enumerator in core), 99
 Node_Kind.Table_Body (enumerator in core), 99
 Node_Kind.Table_Head (enumerator in core), 99
 Node_Kind.Target (enumerator in core), 100
 Node_Kind.Term (enumerator in core), 98
 Node_Kind.Text (enumerator in core), 101
 Node_Kind.Thematic_Break (enumerator in core), 99
 Node_Kind.Title (enumerator in core), 98
 Node_Kind.Title_Reference (enumerator in core), 102
 Node_Kind.Toctree (enumerator in core), 100
 Node_Kind.Toctree_Entry (enumerator in core), 100
 Node_Kind.Topic (enumerator in core), 100
 Node_Kind.Version_Modified (enumerator in core), 101
 Node_Kind.Xref (enumerator in core), 102
 node_text (procedure in core), 128
 NONE (constant in core), 136
 none (procedure in jinja), 241

None (struct in jinja), 233
 normalize_label (procedure in core), 128
 normalize_nfd_into (procedure in core), 128
 NOTICES (constant in treesitter), 339
 now (procedure in gds), 144
 number_repr (procedure in pyscan), 287

O

Object (struct in pyscan), 281
 Object.abstract (field in pyscan), 282
 Object.class (field in pyscan), 281
 Object.class_property (field in pyscan), 282
 Object.decorator (field in pyscan), 282
 Object.def (field in pyscan), 282
 Object.doc (field in pyscan), 282
 Object.function (field in pyscan), 281
 Object.kind (field in pyscan), 281
 Object.module (field in pyscan), 281
 Object.owner (field in pyscan), 282
 Object.partial (field in pyscan), 282
 Object.partial_args (field in pyscan), 282
 Object.partial_method (field in pyscan), 282
 Object.path (field in pyscan), 281
 Object.tuple (field in pyscan), 281
 Object.value (field in pyscan), 281
 Object.wrapper (field in pyscan), 281
 Object_Kind (enum in pyscan), 282
 Object_Kind.Builtin (enumerator in pyscan), 283
 Object_Kind.Class (enumerator in pyscan), 282
 Object_Kind.External (enumerator in pyscan), 283
 Object_Kind.Function (enumerator in pyscan), 282
 Object_Kind.Module (enumerator in pyscan), 282
 Object_Kind.None (enumerator in pyscan), 282
 Object_Kind.Value (enumerator in pyscan), 283
 Object_Type (struct in readers/sphinx), 329
 Object_Type.directive (field in readers/sphinx), 329
 Object_Type.display (field in readers/sphinx), 329
 Object_Type.first_word (field in readers/sphinx), 329
 Object_Type.index (field in readers/sphinx), 329
 Object_Type.program (field in readers/sphinx), 330
 Object_Type.role (field in readers/sphinx), 329
 Object_Type.target (field in readers/sphinx), 329
 odindoc (package), 242
 offset_in (procedure in core), 129
 offset_of (procedure in typst), 343
 On_Refusal (type in host), 197
 open (procedure in readers/rst), 315
 Opt_Spec (struct in readers/rst), 307
 Opt_Spec.choices (field in readers/rst), 307

- Opt_Spec.name (field in readers/rst), 307
- Opt_Spec.type (field in readers/rst), 307
- Opt_Type (enum in readers/rst), 307
- Opt_Type.Char (enumerator in readers/rst), 308
- Opt_Type.Choice (enumerator in readers/rst), 308
- Opt_Type.Class (enumerator in readers/rst), 308
- Opt_Type.Encoding (enumerator in readers/rst), 308
- Opt_Type.Figwidth (enumerator in readers/rst), 308
- Opt_Type.Flag (enumerator in readers/rst), 307
- Opt_Type.Int (enumerator in readers/rst), 307
- Opt_Type.Length (enumerator in readers/rst), 308
- Opt_Type.Length_Percent (enumerator in readers/rst), 308
- Opt_Type.Name (enumerator in readers/rst), 308
- Opt_Type.Nonneg_Int (enumerator in readers/rst), 307
- Opt_Type.Opt_Int (enumerator in readers/rst), 307
- Opt_Type.Path (enumerator in readers/rst), 308
- Opt_Type.Percentage (enumerator in readers/rst), 308
- Opt_Type.Pos_Int (enumerator in readers/rst), 307
- Opt_Type.Text (enumerator in readers/rst), 307
- Opt_Type.Text_Required (enumerator in readers/rst), 307
- Opt_Type.Widths (enumerator in readers/rst), 308
- opt_value (procedure in readers/rst), 315
- Option (struct in autodoc), 79
 - (struct in odindoc), 246
 - (struct in readers/commonmark), 293
- Option.name (field in autodoc), 79
 - (field in odindoc), 246
 - (field in readers/commonmark), 293
- Option.span (field in readers/commonmark), 293
- Option.value (field in autodoc), 79
 - (field in odindoc), 246
 - (field in readers/commonmark), 293
- option_lines (procedure in readers/rst), 315
- Options (struct in fetch), 138
 - (struct in gds), 140
 - (struct in graphdog), 165
 - (struct in jinja), 233
 - (struct in readers/rst/inline), 319
 - (struct in typst), 341
- Options.attributes (field in graphdog), 165
- Options.author (field in gds), 140
- Options.autoescape (field in jinja), 233
- Options.autoescape_extensions (field in jinja), 233
- Options.character_level (field in readers/rst/inline), 320
- Options.check_only (field in gds), 141
- Options.command (field in gds), 140
- Options.default_substitution (field in readers/rst/inline), 320
- Options.dir (field in gds), 140
- Options.discussion (field in gds), 140
- Options.docinfo (field in readers/rst/inline), 320
- Options.dry_run (field in gds), 141
- Options.embedded_fonts_only (field in typst), 341
- Options.engine (field in graphdog), 165
- Options.evidence (field in gds), 140
- Options.files (field in graphdog), 165
- Options.font_paths (field in typst), 341
- Options.format (field in graphdog), 165
- Options.historical (field in gds), 141
- Options.identifier (field in gds), 140
- Options.keep_trailing_newline (field in jinja), 233
- Options.label (field in gds), 140
- Options.labels (field in gds), 140
- Options.language (field in readers/rst/inline), 320
- Options.loader (field in jinja), 233
- Options.lstrip_blocks (field in jinja), 233
- Options.max_bytes (field in fetch), 138
- Options.max_iterations (field in jinja), 234
- Options.max_nesting (field in jinja), 234
- Options.max_output (field in jinja), 234
- Options.max_recursion (field in jinja), 233
- Options.number (field in gds), 141
- Options.offline (field in typst), 341
- Options.package_cache (field in typst), 341
- Options.pep_base_url (field in readers/rst/inline), 320
- Options.pep_references (field in readers/rst/inline), 319
- Options.random_seed (field in jinja), 234
- Options.reason (field in gds), 140
- Options.record (field in gds), 140
- Options.render (field in gds), 141
- Options.reserve (field in graphdog), 165
 - (field in typst), 341
- Options.rfc_base_url (field in readers/rst/inline), 320
- Options.rfc_references (field in readers/rst/inline), 320
- Options.rollback (field in gds), 141
- Options.sectsubtitle (field in readers/rst/inline), 320
- Options.slug (field in gds), 140
- Options.smart_quotes (field in readers/rst/inline), 320
- Options.smart_quotes_locales (field in readers/rst/inline), 320
- Options.stack_bytes (field in jinja), 234
- Options.state (field in gds), 140
- Options.tab_width (field in readers/rst/inline), 320
- Options.target (field in gds), 141
- Options.timeout_ms (field in fetch), 138
- Options.to (field in gds), 140
- Options.trim_blocks (field in jinja), 233

Options.trim_footnote_reference_space
 (field in readers/rst/inline), 320
 Options.undefined (field in jinja), 233
 Options.user_agent (field in fetch), 138
 Options.warnings (field in graphdog), 165
 OS (type in odindoc), 246
 out_of_memory (procedure in host), 210
 Outcome (struct in typst), 341
 Outcome.detail (field in typst), 341
 Outcome.diagnostics (field in typst), 341
 Outcome.exit_code (field in typst), 341
 Outcome.launch_error (field in typst), 341
 Outcome.ok (field in typst), 341
 Output (struct in highlight), 172
 Output.buf (field in highlight), 172
 Output.written (field in highlight), 172
 output_fits (procedure in highlight), 186
 output_string (procedure in highlight), 186
 output_text (procedure in host), 210

P

Package (struct in odindoc), 246
 Package.decls (field in odindoc), 247
 Package.dir (field in odindoc), 247
 Package.doc (field in odindoc), 247
 Package.errors (field in odindoc), 247
 Package.files (field in odindoc), 247
 Package.id (field in odindoc), 247
 Package.imports (field in odindoc), 247
 Package.name (field in odindoc), 247
 Package.platform (field in odindoc), 247
 Package.synopsis (field in odindoc), 247
 Package.targets (field in odindoc), 247
 Package.too_deep (field in odindoc), 247
 Package.too_large (field in odindoc), 247
 Package_Ref (struct in odindoc), 247
 Package_Ref.dir (field in odindoc), 247
 Package_Ref.id (field in odindoc), 247
 Param (struct in pyscan), 283
 Param.annotation (field in pyscan), 283
 Param.default (field in pyscan), 283
 Param.kind (field in pyscan), 283
 Param.name (field in pyscan), 283
 Param.type_comment (field in pyscan), 283
 Param_Kind (enum in pyscan), 283
 Param_Kind.Keyword_Only (enumerator in pyscan), 283
 Param_Kind.Positional_Only (enumerator in pyscan), 283
 Param_Kind.Positional_Or_Keyword (enumerator in pyscan), 283
 Param_Kind.Var_Keyword (enumerator in pyscan), 283
 Param_Kind.Var_Positional (enumerator in pyscan), 283
 parent (procedure in treesitter), 337
 parent_of (procedure in core), 129
 parse (procedure in treesitter), 337
 PARSE_BASE_BYTES (constant in odindoc), 250
 parse_blocks (procedure in readers/rst), 316
 parse_body (procedure in readers/commonmark), 300
 (procedure in readers/rst), 316
 parse_bound (procedure in odindoc), 250

PARSE_BYTES_PER_BYTE (constant in odindoc), 250
 parse_engine (procedure in graphdog), 166
 parse_expression (procedure in pyscan), 288
 parse_format (procedure in graphdog), 166
 parse_inline (procedure in readers/commonmark), 300
 (procedure in readers/rst/inline), 323
 parse_mo (procedure in gettext), 158
 parse_plural_forms (procedure in gettext), 158
 parse_po (procedure in gettext), 158
 parse_record (procedure in gds), 144
 parse_source (procedure in readers/rst), 316
 parse_target (procedure in i18n), 224
 parse_template (procedure in jinja), 241
 parse_titled (procedure in readers/rst), 316
 Parser (struct in readers/rst), 308
 (struct in treesitter), 334
 Parser.b (field in readers/rst), 308
 Parser.ctx (field in readers/rst), 308
 Parser.depth (field in readers/rst), 308
 Parser.ext (field in readers/rst), 309
 Parser.has_pending (field in readers/rst), 309
 Parser.host (field in readers/rst), 310
 Parser.hosted (field in readers/rst), 310
 Parser.in_sidebar (field in readers/rst), 309
 Parser.inl (field in readers/rst), 308
 Parser.level_base (field in readers/rst), 309
 Parser.pending (field in readers/rst), 309
 Parser.pending_span (field in readers/rst), 309
 Parser.section_base (field in readers/rst), 309
 Parser.sections (field in readers/rst), 309
 Parser.sections_used (field in readers/rst), 309
 Parser.src (field in readers/rst), 308
 Parser.stack (field in readers/rst), 309
 Parser.styles (field in readers/rst), 309
 Parser.styles_used (field in readers/rst), 309
 Parser.tab (field in readers/rst), 308
 Parser.table_extra (field in readers/rst), 310
 Parser.text (field in readers/rst), 308
 Parser.top_count (field in readers/rst), 309
 Parser.top_transition (field in readers/rst), 309
 Parser.transition_span (field in readers/rst), 309
 parser_delete (procedure in treesitter), 338
 parser_new (procedure in treesitter), 338
 Pass (struct in core), 103
 Pass.name (field in core), 103
 Pass.procedure (field in core), 103
 Pass.user (field in core), 103
 Pass_Outcome (enum in core), 103
 Pass_Outcome.Failed (enumerator in core), 103
 Pass_Outcome.Rebuilt (enumerator in core), 103
 Pass_Outcome.Unchanged (enumerator in core), 103

- PATH_BYTES (constant in host), 216
- path_of (procedure in readers/rst), 316
- path_type (procedure in host), 210
- Pdf_Document (struct in project), 267
- Pdf_Document.author (field in project), 267
- Pdf_Document.file (field in project), 267
- Pdf_Document.root (field in project), 267
- Pdf_Document.title (field in project), 267
- pinned (procedure in typst), 343
- PINNED_VERSION (constant in typst), 343
- place_titled (procedure in readers/rst), 316
- plain_text (procedure in core), 129
- Plan (struct in gds), 141
- Plan.changes (field in gds), 141
- Plan.summary (field in gds), 141
- plan_bytes (procedure in core), 129
- plan_session (procedure in host), 210
- plural (procedure group in gettext), 160
- plural_forms_for (procedure in gettext), 158
- plural_index (procedure in gettext), 158
- Plural_Node (struct in gettext), 150
- Plural_Node.a (field in gettext), 150
- Plural_Node.b (field in gettext), 150
- Plural_Node.c (field in gettext), 150
- Plural_Node.op (field in gettext), 150
- Plural_Node.value (field in gettext), 151
- Plural_Op (enum in gettext), 151
- Plural_Op.Add (enumerator in gettext), 151
- Plural_Op.And (enumerator in gettext), 151
- Plural_Op.Choose (enumerator in gettext), 151
- Plural_Op.Div (enumerator in gettext), 151
- Plural_Op.Equal (enumerator in gettext), 151
- Plural_Op.Greater (enumerator in gettext), 151
- Plural_Op.Greater_Equal (enumerator in gettext), 151
- Plural_Op.Less (enumerator in gettext), 151
- Plural_Op.Less_Equal (enumerator in gettext), 151
- Plural_Op.Mod (enumerator in gettext), 151
- Plural_Op.Mul (enumerator in gettext), 151
- Plural_Op.N (enumerator in gettext), 151
- Plural_Op.Not (enumerator in gettext), 151
- Plural_Op.Not_Equal (enumerator in gettext), 151
- Plural_Op.Number (enumerator in gettext), 151
- Plural_Op.Or (enumerator in gettext), 151
- Plural_Op.Sub (enumerator in gettext), 151
- Plural_Problem (struct in gettext), 152
- Plural_Problem.column (field in gettext), 152
- Plural_Problem.message (field in gettext), 152
- Plural_Rule (struct in gettext), 152
- Plural_Rule.count (field in gettext), 152
- Plural_Rule.nodes (field in gettext), 152
- Plural_Rule.nplurals (field in gettext), 152
- Plural_Rule.root (field in gettext), 152
- plural_with_context (procedure group in gettext), 160
- Point (struct in treesitter), 334
- Point.column (field in treesitter), 334
- Point.row (field in treesitter), 334
- Policy (struct in core), 103
- Policy.includes (field in core), 103
- Policy.keep_documents_with_errors (field in core), 103
- Policy.math (field in core), 103
- Policy.raw (field in core), 103
- Policy.strict (field in core), 103
- Policy.unsafe_urls (field in core), 103
- pool_classes (procedure in readers/rst), 316
- pool_lines (procedure in readers/rst), 316
- pool_lower (procedure in readers/rst), 317
- pool_of (procedure in gettext), 159
- pool_string (procedure in core), 129
- pop (procedure in core), 129
- Pot_Header (struct in i18n), 218
- Pot_Header.copyright (field in i18n), 218
- Pot_Header.creation_date (field in i18n), 219
- Pot_Header.language_team (field in i18n), 218
- Pot_Header.last_translator (field in i18n), 218
- Pot_Header.project (field in i18n), 218
- Pot_Header.version (field in i18n), 218
- prepare_docstring (procedure in pycan), 288
- prepare_session (procedure in host), 210
- prev_named_sibling (procedure in treesitter), 338
- prev_sibling (procedure in treesitter), 338
- Previous (struct in gettext), 152
- Previous.ctx (field in gettext), 152
- Previous.has_ctx (field in gettext), 152
- Previous.has_plural (field in gettext), 152
- Previous.id (field in gettext), 152
- Previous.id_plural (field in gettext), 152
- Private (enum in odindoc), 248
- Private.File (enumerator in odindoc), 248
- Private.Package (enumerator in odindoc), 248
- Private.Public (enumerator in odindoc), 248
- problem (procedure in gds), 144
- Problem (struct in gds), 141
 - (struct in highlight), 172
 - (struct in host), 197
- Problem.code (field in gds), 141
 - (field in host), 197
- Problem.column (field in highlight), 173
- Problem.excerpt (field in highlight), 173
- Problem.exit (field in gds), 141
 - (field in host), 198
- Problem.hint (field in gds), 141
 - (field in host), 197
- Problem.kind (field in highlight), 172
- Problem.lexer (field in highlight), 172
- Problem.line (field in highlight), 173
- Problem.message (field in gds), 141
 - (field in host), 197
- Problem.name (field in highlight), 172
- Problem.near (field in highlight), 173
- Problem.path (field in gds), 141

(field in host), 197
 Problem.suggestion (field in highlight), 172
 Problem.title (field in gds), 141
 (field in host), 197
 Problem_Kind (enum in highlight), 173
 Problem_Kind.Lexing_Error (enumerator in highlight), 173
 Problem_Kind.None (enumerator in highlight), 173
 Problem_Kind.Unknown_Language (enumerator in highlight), 173
 Problem_Kind.Unknown_Style (enumerator in highlight), 173
 Progress (enum in i18n), 219
 Progress.Translated (enumerator in i18n), 219
 Progress.Untranslated (enumerator in i18n), 219
 Progress_Classes (bit set in i18n), 219
 project (package), 250
 Project (struct in odindoc), 248
 (struct in project), 268
 (struct in pyscan), 283
 Project.config (field in project), 268
 Project.files (field in odindoc), 248
 (field in pyscan), 284
 Project.memory (field in project), 268
 Project.modules (field in pyscan), 284
 Project.mro_depth (field in pyscan), 284
 Project.owner (field in project), 268
 Project.packages (field in odindoc), 248
 Project.parser (field in pyscan), 284
 Project.reader (field in odindoc), 248
 (field in pyscan), 284
 Project.registry (field in project), 268
 Project.resolving (field in pyscan), 284
 Project.roots (field in odindoc), 248
 (field in pyscan), 284
 Project.settings (field in project), 268
 Project.stack_base (field in pyscan), 284
 Project.steps (field in pyscan), 284
 Project.trees (field in pyscan), 284
 Project.warnings (field in project), 268
 publish (procedure in host), 210
 push (procedure in core), 129
 push_lines (procedure in readers/rst), 317
 put (procedure in highlight), 186
 put_byte (procedure in highlight), 186
 put_html (procedure in highlight), 186
 pyscan (package), 274
 python (procedure in treesitter), 338
 python_space (procedure in readers/rst/inline), 323
 PYTHON_VERSION (constant in treesitter), 339

Q

quickstart (procedure in project), 273
 Quickstart_Options (struct in project), 269
 Quickstart_Options.author (field in project), 269
 Quickstart_Options.date (field in project), 269
 Quickstart_Options.dir (field in project), 269

Quickstart_Options.language (field in project), 269
 Quickstart_Options.project (field in project), 269
 Quickstart_Options.release (field in project), 269
 Quickstart_Options.root_doc (field in project), 269
 Quickstart_Options.separate (field in project), 269
 Quickstart_Options.suffix (field in project), 269
 Quickstart_Options.version (field in project), 269
 Quickstart_Options.year (field in project), 269

R

read (procedure in host), 210
 (procedure in readers/commonmark), 300
 (procedure in readers/rst), 317
 (procedure in readers/sphinx), 330
 Read_Context (struct in core), 104
 Read_Context.b (field in core), 104
 Read_Context.included (field in core), 104
 Read_Context.limits (field in core), 104
 Read_Context.options (field in core), 104
 Read_Context.policy (field in core), 104
 Read_Context.reports (field in core), 104
 Read_Context.resources (field in core), 104
 Read_Context.scratch (field in core), 104
 Read_Context.source (field in core), 104
 Read_Context.ws (field in core), 104
 read_document (procedure in core), 129
 read_extension (procedure in core), 129
 Read_Extension (struct in core), 104
 Read_Extension.data (field in core), 104
 Read_Extension.id (field in core), 105
 read_file (procedure in host), 211
 read_inline (procedure in readers/commonmark), 300
 (procedure in readers/rst), 317
 (procedure in readers/sphinx), 330
 Read_Inline_Proc (type in core), 105
 read_kept (procedure in host), 211
 read_myst (procedure in readers/commonmark), 300
 read_myst_inline (procedure in readers/commonmark), 300
 Read_Options (struct in core), 105
 Read_Options.extension (field in core), 105
 Read_Options.promote_title (field in core), 105
 Read_Options.tab_width (field in core), 105
 Read_Options.today (field in core), 105
 read_owned (procedure in host), 211
 Read_Proc (type in core), 105
 Read_Result (struct in core), 105
 Read_Result.document (field in core), 105
 Read_Result.using (field in core), 105
 read_source (procedure in host), 211
 reader (procedure in readers/commonmark), 300
 (procedure in readers/rst), 317

- (procedure in readers/sphinx), 331
- Reader (struct in core), 105
- Reader.conformance (field in core), 106
- Reader.id (field in core), 105
- Reader.name (field in core), 105
- Reader.read (field in core), 106
- Reader.read_inline (field in core), 106
- Reader.version (field in core), 105
- reader_settings (procedure in core), 130
- readers/commonmark (package), 289
- readers/rst (package), 301
- readers/rst/inline (package), 319
- readers/sphinx (package), 324
- Record (struct in gds), 141
- Record.authors (field in gds), 142
- Record.created (field in gds), 142
- Record.discussion (field in gds), 142
- Record.fields (field in gds), 142
- Record.historical (field in gds), 142
- Record.kind (field in gds), 142
- Record.labels (field in gds), 142
- Record.number (field in gds), 141
- Record.path (field in gds), 142
- Record.slug (field in gds), 141
- Record.state (field in gds), 142
- Record.text (field in gds), 142
- Record.title (field in gds), 142
- Record.updated (field in gds), 142
- recover (procedure in gds), 144
- reference (procedure in pyscan), 288
- Reference_Style (enum in gettext), 152
- Reference_Style.None (enumerator in gettext), 153
- Reference_Style.One_Per_Line (enumerator in gettext), 153
- Reference_Style.Wrapped (enumerator in gettext), 152
- Refusal (struct in host), 198
- Refusal.disk (field in host), 198
- Refusal.limit (field in host), 198
- Refusal.need (field in host), 198
- Refusal.place (field in host), 198
- Refusal.used (field in host), 198
- Refusal_Choice (enum in host), 198
- Refusal_Choice.Disk (enumerator in host), 198
- Refusal_Choice.Raise (enumerator in host), 198
- Refusal_Choice.Stop (enumerator in host), 198
- Refusal_Decision (struct in host), 198
- Refusal_Decision.choice (field in host), 198
- Refusal_Decision.limit (field in host), 198
- Region (enum in core), 106
- Region.Attributes (enumerator in core), 106
- Region.Cells (enumerator in core), 106
- Region.Links (enumerator in core), 106
- Region.Lists (enumerator in core), 106
- Region.Nodes (enumerator in core), 106
- Region.None (enumerator in core), 106
- Region.Output (enumerator in core), 106
- Region.Reports (enumerator in core), 106
- Region.Scratch (enumerator in core), 106
- Region.Second_Bank (enumerator in core), 106
- Region.Source_Map (enumerator in core), 106
- Region.Sources (enumerator in core), 106
- Region.Text (enumerator in core), 106
- registry (procedure in defaults), 137
- Registry (struct in core), 106
- Registry.readers (field in core), 106
- Registry.renderers (field in core), 107
- release (procedure in project), 273
 - (procedure in readers/rst), 317
- remove_path (procedure in host), 211
- remove_stages (procedure in host), 211
- rename_path (procedure in host), 212
- render (procedure in graphdog), 166
 - (procedure in jinja), 241
 - (procedure in render/html), 331
 - (procedure in render/text), 332
 - (procedure in render/typst), 333
- render/html (package), 331
- render/text (package), 332
- render/typst (package), 332
- Render_Context (struct in core), 107
- Render_Context.doc (field in core), 107
- Render_Context.limits (field in core), 107
- Render_Context.options (field in core), 107
- Render_Context.out (field in core), 107
- Render_Context.policy (field in core), 107
- Render_Context.reports (field in core), 107
- Render_Context.scratch (field in core), 107
- Render_Context.view (field in core), 107
- render_document (procedure in core), 130
- render_json (procedure in host), 212
- Render_Options (struct in core), 107
- Render_Options.anchor_prefix (field in core), 108
- Render_Options.highlighter (field in core), 108
- Render_Options.labels (field in core), 108
- Render_Options.language (field in core), 107
- Render_Options.math (field in core), 108
- Render_Options.math_display (field in core), 108
- Render_Options.math_inline (field in core), 108
- Render_Options.permalink_icon (field in core), 108
- Render_Options.permalinks (field in core), 108
- Render_Options.reference_profile (field in core), 107
- Render_Options.secnumber_suffix (field in core), 108
- Render_Options.section_numbers (field in core), 107
- Render_Options.standalone (field in core), 107
- Render_Options.stylesheet (field in core), 107
- Render_Options.title (field in core), 107
- render_position (procedure in jinja), 241
- render_problem_json (procedure in host), 212
- render_problem_text (procedure in host), 212
- Render_Proc (type in core), 108
- render_source (procedure in jinja), 241
- render_text (procedure in host), 212

- renderer (procedure in render/html), 331
 - (procedure in render/text), 332
 - (procedure in render/typst), 333
- Renderer (struct in core), 108
- Renderer.conformance (field in core), 109
- Renderer.id (field in core), 108
- Renderer.media_type (field in core), 108
- Renderer.name (field in core), 108
- Renderer.render (field in core), 109
- Renderer.version (field in core), 108
- replace_field (procedure in gds), 144
- report (procedure in core), 130
 - (procedure in readers/rst), 317
- Report (struct in host), 199
- Report.category (field in host), 199
- Report.code (field in host), 199
- Report.column (field in host), 199
- Report.end_column (field in host), 199
- Report.end_line (field in host), 199
- Report.excerpt (field in host), 199
- Report.hint (field in host), 199
- Report.line (field in host), 199
- Report.marker (field in host), 199
- Report.message (field in host), 199
- Report.path (field in host), 199
- Report.schema (field in host), 199
- Report.severity (field in host), 199
- Report.source (field in host), 199
- Report.title (field in host), 199
- report_message (procedure in core), 130
- Reports (struct in core), 109
- Reports.errors (field in core), 109
- Reports.fatal (field in core), 109
- Reports.has_primary (field in core), 109
- Reports.items (field in core), 109
- Reports.muted (field in core), 109
- Reports.primary (field in core), 109
- Reports.suppressed (field in core), 109
- Reports.truncated (field in core), 109
- Reports.used (field in core), 109
- Reports.warnings (field in core), 109
- repr (procedure in jinja), 241
- repr_bytes (procedure in pyscan), 288
- repr_str (procedure in pyscan), 288
- Request (struct in autodoc), 79
 - (struct in core), 109
 - (struct in odindoc), 248
- Request.argument (field in autodoc), 79
 - (field in odindoc), 248
- Request.cancel (field in core), 110
- Request.class (field in autodoc), 79
- Request.content (field in autodoc), 79
- Request.current (field in odindoc), 248
- Request.directive (field in autodoc), 79
 - (field in odindoc), 248
- Request.limits (field in core), 110
- Request.module (field in autodoc), 79
- Request.options (field in autodoc), 79
 - (field in odindoc), 248
- Request.passes (field in core), 110
- Request.policy (field in core), 110
- Request.read (field in core), 109
- Request.reader (field in core), 109
- Request.render (field in core), 109
- Request.renderer (field in core), 109
- Request.resources (field in core), 110
- Request.source (field in core), 109
- Request.source_map (field in core), 110
- Reserve (struct in graphdog), 165
 - (struct in typst), 342
- Reserve.procedure (field in graphdog), 165
 - (field in typst), 342
- Reserve.user (field in graphdog), 165
 - (field in typst), 342
- reset_workspace (procedure in core), 130
- resize_output (procedure in host), 212
- resize_source_map (procedure in host), 212
- resolve (procedure in gds), 144
 - (procedure in highlight), 186
 - (procedure in readers/rst/inline), 324
- resolve_link (procedure in core), 130
- resolve_style (procedure in highlight), 186
- Resource_Provider (struct in core), 110
- Resource_Provider.procedure (field in core), 110
- Resource_Provider.user (field in core), 110
- Resource_Status (enum in core), 110
- Resource_Status.Limit (enumerator in core), 110
- Resource_Status.Missing (enumerator in core), 110
- Resource_Status.Ok (enumerator in core), 110
- Resource_Status.Refused (enumerator in core), 110
- Result (struct in autodoc), 79
 - (struct in core), 110
 - (struct in fetch), 138
 - (struct in odindoc), 248
- Result.data (field in fetch), 138
- Result.error (field in fetch), 138
- Result.lines (field in autodoc), 79
 - (field in odindoc), 248
- Result.mapped (field in core), 111
- Result.message (field in fetch), 138
- Result.messages (field in autodoc), 79
 - (field in odindoc), 248
- Result.need (field in core), 111
- Result.primary (field in core), 111
- Result.reports (field in core), 111
- Result.status (field in core), 111
 - (field in fetch), 138
- Result.suppressed (field in core), 111
- Result.truncated (field in core), 111
- Result.url (field in fetch), 138
- Result.written (field in core), 111
- Role (struct in readers/commonmark), 293
 - (struct in readers/rst/inline), 320
- Role.base (field in readers/rst/inline), 320
- Role.classes (field in readers/rst/inline), 320
- Role.content (field in readers/commonmark), 294
- Role.format (field in readers/rst/inline), 320
- Role.language (field in readers/rst/inline), 321
- Role.lines (field in readers/commonmark), 294
- Role.name (field in readers/commonmark), 294
 - (field in readers/rst/inline), 320
- Role.source (field in readers/commonmark), 294
- Role.span (field in readers/commonmark), 294
- Role_Base (enum in readers/rst/inline), 321

- Role_Base.Abbreviation (enumerator in readers/rst/inline), 321
- Role_Base.Acronym (enumerator in readers/rst/inline), 321
- Role_Base.Code (enumerator in readers/rst/inline), 321
- Role_Base.Custom (enumerator in readers/rst/inline), 321
- Role_Base.Emphasis (enumerator in readers/rst/inline), 321
- Role_Base.Literal (enumerator in readers/rst/inline), 321
- Role_Base.Math (enumerator in readers/rst/inline), 321
- Role_Base.Pep_Reference (enumerator in readers/rst/inline), 321
- Role_Base.Raw (enumerator in readers/rst/inline), 321
- Role_Base.Rfc_Reference (enumerator in readers/rst/inline), 321
- Role_Base.Strong (enumerator in readers/rst/inline), 321
- Role_Base.Subscript (enumerator in readers/rst/inline), 321
- Role_Base.Superscript (enumerator in readers/rst/inline), 321
- Role_Base.Title_Reference (enumerator in readers/rst/inline), 321
- Role_Call (struct in readers/rst/inline), 321
- Role_Call.b (field in readers/rst/inline), 322
- Role_Call.content (field in readers/rst/inline), 322
- Role_Call.marked (field in readers/rst/inline), 322
- Role_Call.name (field in readers/rst/inline), 322
- Role_Call.span (field in readers/rst/inline), 322
- Role_Call.state (field in readers/rst/inline), 321
- Role_Call.text (field in readers/rst/inline), 322
- Role_Handler (struct in readers/commonmark), 294
(struct in readers/rst), 310
- Role_Handler.name (field in readers/commonmark), 294
(field in readers/rst), 310
- Role_Handler.run (field in readers/commonmark), 294
(field in readers/rst), 310
- Role_Hook (struct in readers/rst/inline), 322
- Role_Hook.procedure (field in readers/rst/inline), 322
- Role_Hook.user (field in readers/rst/inline), 322
- Role_Proc (type in readers/commonmark), 294
(type in readers/rst), 310
- root (procedure in treesitter), 338
- Root (struct in odindoc), 249
- ROOT (type in core), 104
- Root.collection (field in odindoc), 249
- Root.dir (field in odindoc), 249
- rst_options (procedure in defaults), 137
- Rule (enum in readers/commonmark), 294
- Rule.Autolink (enumerator in readers/commonmark), 295
- Rule.Backticks (enumerator in readers/commonmark), 295
- Rule.Blockquote (enumerator in readers/commonmark), 294
- Rule.Code (enumerator in readers/commonmark), 294
- Rule.Emphasis (enumerator in readers/commonmark), 295
- Rule.Entity (enumerator in readers/commonmark), 295
- Rule.Escape (enumerator in readers/commonmark), 295
- Rule.Fence (enumerator in readers/commonmark), 294
- Rule.Footnote_Def (enumerator in readers/commonmark), 296
- Rule.Footnote_Ref (enumerator in readers/commonmark), 296
- Rule.Front_Matter (enumerator in readers/commonmark), 295
- Rule.Heading (enumerator in readers/commonmark), 295
- Rule.Hr (enumerator in readers/commonmark), 294
- Rule.Html_Block (enumerator in readers/commonmark), 295
- Rule.Html_Inline (enumerator in readers/commonmark), 295
- Rule.Image (enumerator in readers/commonmark), 295
- Rule.Lheading (enumerator in readers/commonmark), 295
- Rule.Link (enumerator in readers/commonmark), 295
- Rule.List (enumerator in readers/commonmark), 294
- Rule.Math_Block (enumerator in readers/commonmark), 295
- Rule.Math_Inline (enumerator in readers/commonmark), 295
- Rule.Myst_Block_Break (enumerator in readers/commonmark), 295
- Rule.Myst_Directive (enumerator in readers/commonmark), 296
- Rule.Myst_Line_Comment (enumerator in readers/commonmark), 295
- Rule.Myst_Role (enumerator in readers/commonmark), 295
- Rule.Myst_Target (enumerator in readers/commonmark), 295
- Rule.Newline (enumerator in readers/commonmark), 295
- Rule.Reference (enumerator in readers/commonmark), 294

Rule.Table (enumerator in readers/
commonmark), 294
run (procedure in cli), 81
 (procedure in host), 212
run_hosted (procedure in readers/rst), 318
run_workers (procedure in host), 213
RUNTIME_VERSION (constant in
treesitter), 339

S

same_contents (procedure in host), 213
save_po (procedure in gettext), 159
Scanner (struct in highlight), 173
Scanner.code (field in highlight), 173
Scanner.count (field in highlight), 174
Scanner.head (field in highlight), 174
Scanner.inner (field in highlight), 174
Scanner.limit (field in highlight), 173
Scanner.nested (field in highlight), 174
Scanner.pos (field in highlight), 173
Scanner.queue (field in highlight), 174
Scanner.region (field in highlight), 174
Scanner.region_start (field in highlight),
174
Scanner.state (field in highlight), 173
scanner_init (procedure in highlight), 187
SCHEMA_VERSION (constant in core), 136
Scope (struct in pyscan), 284
Scope.defs (field in pyscan), 284
Scratch (struct in core), 111
Scratch.bytes (field in core), 111
Scratch.high (field in core), 111
Scratch.used (field in core), 111
scratch_buffer (procedure in core), 130
scratch_mark (procedure in core), 130
Scratch_Mark (type in core), 111
scratch_release (procedure in core), 131
scratch_take (procedure in core), 131
serve (procedure in project), 273
Server (struct in project), 269
Server.project (field in project), 269
Server.rebuild (field in project), 270
Server.stamp (field in project), 270
Server.user (field in project), 270
Session (struct in autodoc), 79
 (struct in host), 200
Session.allocated (field in host), 201
Session allocator (field in host), 200
Session.denied (field in host), 201
Session.disk (field in host), 201
Session.held (field in host), 201
Session.inputs (field in host), 201
Session.kept (field in host), 201
Session.limit (field in host), 200
Session.map_size (field in host), 200
Session.merged (field in autodoc), 80
Session.output (field in host), 200
Session.own (field in host), 200
Session.project (field in autodoc), 80
Session.ready (field in host), 200
Session.refused (field in host), 201
Session.result (field in host), 200
Session.retries (field in host), 200
Session.shared (field in host), 200
Session.sizes (field in host), 200
Session.source_map (field in host), 200
Session.storage (field in host), 200
Session.ws (field in host), 200
session_destroy (procedure in host), 213
session_init (procedure in autodoc), 80
set (procedure in jinja), 241
set_cell (procedure in core), 131
set_header (procedure in i18n), 224
set_header_field (procedure in gettext),
159
set_key (procedure in jinja), 241
set_kind (procedure in core), 131
set_link (procedure in core), 131
set_list (procedure in core), 131
set_span_end (procedure in core), 131
set_text (procedure in core), 131
set_value (procedure in core), 131
Settings (struct in readers/commonmark),
296
 (struct in readers/rst), 310
Settings.character_level_inline_markup
 (field in readers/rst), 311
Settings.commonmark_only (field in
readers/commonmark), 296
Settings.disable_syntax (field in readers/
commonmark), 296
Settings.dmath_allow_digits (field in
readers/commonmark), 297
Settings.dmath_allow_labels (field in
readers/commonmark), 297
Settings.dmath_allow_space (field in
readers/commonmark), 297
Settings.dmath_double_inline (field in
readers/commonmark), 297
Settings.docinfo (field in readers/rst), 311
Settings.docname (field in readers/
commonmark), 298
Settings.enable_extensions (field in
readers/commonmark), 296
Settings.env_config (field in readers/
commonmark), 298
Settings.extension (field in readers/
commonmark), 296
 (field in readers/rst), 310
Settings.footnote_transition (field in
readers/commonmark), 297
Settings.heading_anchors (field in readers/
commonmark), 297
Settings.heading_slug_func (field in
readers/commonmark), 297
Settings.host (field in readers/rst), 310
Settings.html_meta (field in readers/
commonmark), 297
Settings.language (field in readers/rst), 311
Settings.linkify_fuzzy_links (field in
readers/commonmark), 296
Settings.pep_base_url (field in readers/rst),
311
Settings.pep_references (field in readers/
rst), 310
Settings.rfc_base_url (field in readers/rst),
311
Settings.rfc_references (field in readers/
rst), 311
Settings.sectsubtitle (field in readers/rst),
311

- Settings.smart_quotes (field in readers/rst), 311
- Settings.smart_quotes_locales (field in readers/rst), 311
- Settings.sub_delimiters (field in readers/commonmark), 297
- Settings.substitutions (field in readers/commonmark), 297
- Settings.tab_width (field in readers/rst), 311
- Settings.today (field in readers/commonmark), 298
- Settings.trim_footnote_reference_space (field in readers/rst), 311
- Settings.url_schemes (field in readers/commonmark), 296
- Settings.words_per_minute (field in readers/commonmark), 298
- settle (procedure in host), 213
- Severity (enum in autodoc), 80
 - (enum in core), 111
 - (enum in odindoc), 249
- Severity.Error (enumerator in autodoc), 80
 - (enumerator in core), 111
 - (enumerator in odindoc), 249
- Severity.Note (enumerator in autodoc), 80
 - (enumerator in core), 111
 - (enumerator in odindoc), 249
- Severity.Warning (enumerator in autodoc), 80
 - (enumerator in core), 111
 - (enumerator in odindoc), 249
- similarity (procedure in gettext), 159
- Sizes (struct in host), 201
- Sizes.attributes (field in host), 201
- Sizes.cells (field in host), 201
- Sizes.links (field in host), 201
- Sizes.lists (field in host), 201
- Sizes.nodes (field in host), 201
- Sizes.output (field in host), 202
- Sizes.reports (field in host), 202
- Sizes.scratch (field in host), 201
- Sizes.second_bank (field in host), 202
- Sizes.sources (field in host), 202
- Sizes.text (field in host), 201
- sizes_for (procedure in host), 213
- slab_plan (procedure in core), 132
- Slab_Plan (struct in core), 112
- Slab_Plan.attributes (field in core), 112
- Slab_Plan.cells (field in core), 112
- Slab_Plan.links (field in core), 112
- Slab_Plan.lists (field in core), 112
- Slab_Plan.nodes (field in core), 112
- Slab_Plan.output (field in core), 112
- Slab_Plan.reports (field in core), 112
- Slab_Plan.scratch (field in core), 112
- Slab_Plan.second_bank (field in core), 112
- Slab_Plan.sources (field in core), 112
- Slab_Plan.text (field in core), 112
- Slot (struct in pyscan), 284
- Slot.doc (field in pyscan), 284
- Slot.name (field in pyscan), 284
- slots (procedure in pyscan), 288
- Smart_Quotes (enum in readers/rst/inline), 322
 - (type in readers/rst), 311
- Smart_Quotes.Alt (enumerator in readers/rst/inline), 322
- Smart_Quotes.Off (enumerator in readers/rst/inline), 322
- Smart_Quotes.On (enumerator in readers/rst/inline), 322
- Snapshot (struct in core), 112
- Snapshot.attributes (field in core), 112
- Snapshot.cells (field in core), 112
- Snapshot.links (field in core), 112
- Snapshot.lists (field in core), 112
- Snapshot.nodes (field in core), 112
- Snapshot.text (field in core), 112
- Snapshot.text_used (field in core), 112
- Snapshot_Storage (struct in core), 113
- Snapshot_Storage.attributes (field in core), 113
- Snapshot_Storage.cells (field in core), 113
- Snapshot_Storage.links (field in core), 113
- Snapshot_Storage.lists (field in core), 113
- Snapshot_Storage.nodes (field in core), 113
- Snapshot_Storage.text (field in core), 113
- sort_by_reference (procedure in gettext), 159
- sort_items (procedure in inventory), 227
- Source (struct in core), 113
- Source.name (field in core), 113
- Source.text (field in core), 113
- source_entry (procedure in core), 132
- Source_Entry (struct in core), 113
- Source_Entry.included_at (field in core), 113
- Source_Entry.parent (field in core), 113
- Source_Entry.use (field in core), 113
- Source_Entry.using (field in core), 113
- Source_Id (type in core), 113
- source_of (procedure in core), 132
- Source_Span (struct in core), 113
- Source_Span.end (field in core), 114
- Source_Span.source (field in core), 114
- Source_Span.start (field in core), 114
- source_text (procedure in core), 132
- Source_Use (enum in core), 114
- Source_Use.Literal (enumerator in core), 114
- Source_Use.Parsed (enumerator in core), 114
- span_of (procedure in readers/rst), 318
- spend (procedure in core), 132
- sphinx_options (procedure in defaults), 137
- spill_disk (procedure in host), 213
- split_label (procedure in core), 132
- split_lines (procedure in pyscan), 288
- Stack (struct in core), 114
- Stack.items (field in core), 114
- Stack.used (field in core), 114
- stack_budget_limit (procedure in host), 214
- stack_diagnostic (procedure in core), 132
- stack_exhausted (procedure in core), 132
- stack_guard (procedure in core), 133
- Stack_Guard (struct in core), 114
- Stack_Guard.base (field in core), 114
- Stack_Guard.limit (field in core), 114
- STACK_RESERVE (constant in core), 136
- stack_slice (procedure in core), 133

- STAGE_PREFIX (constant in host), 216
- standard_role (procedure in readers/rst), 318
- start_byte (procedure in treesitter), 338
- start_point (procedure in treesitter), 338
- State (struct in jinja), 234
- State.autoescape (field in jinja), 234
- State.caller (field in jinja), 235
- State.ctx (field in jinja), 234
- State.current (field in jinja), 234
- State.depth (field in jinja), 234
- State.env (field in jinja), 234
- State.err (field in jinja), 234
- State.failed (field in jinja), 234
- State.guard (field in jinja), 235
- State.iterations (field in jinja), 234
- State.line (field in jinja), 234
- State.out (field in jinja), 234
- State.rng (field in jinja), 235
- State.scratch (field in jinja), 235
- State.stored (field in jinja), 235
- STATES (constant in gds), 145
- STATIC (constant in graphdog), 167
 - (constant in treesitter), 339
- stats (procedure in gettext), 159
- Stats (struct in gettext), 153
- Stats.fuzzy (field in gettext), 153
- Stats.obsolete (field in gettext), 153
- Stats.translated (field in gettext), 153
- Stats.untranslated (field in gettext), 153
- Status (enum in core), 114
- Status.Cancelled (enumerator in core), 115
- Status.Capacity (enumerator in core), 114
- Status.Internal (enumerator in core), 115
- Status.Invalid_Input (enumerator in core), 114
- Status.Limit (enumerator in core), 115
- Status.Ok (enumerator in core), 114
- Status.Policy (enumerator in core), 114
- Status.Unsupported (enumerator in core), 114
- Storage (struct in core), 115
- Storage.banks (field in core), 115
- Storage.reports (field in core), 115
- Storage.scratch (field in core), 115
- Storage.sources (field in core), 115
- str (procedure in jinja), 242
- str_value (procedure in pyscan), 288
- String_Kind (enum in pyscan), 285
- String_Kind.Bytes (enumerator in pyscan), 285
- String_Kind.Formatted (enumerator in pyscan), 285
- String_Kind.Str (enumerator in pyscan), 285
- string_value (procedure in pyscan), 289
- Style (enum in highlight), 174
- Style.Abap (enumerator in highlight), 174
- Style.Algol (enumerator in highlight), 174
- Style.Algol_Nu (enumerator in highlight), 174
- Style.Arduino (enumerator in highlight), 174
- Style.Autumn (enumerator in highlight), 174
- Style.Borland (enumerator in highlight), 174
- Style.Bw (enumerator in highlight), 174
- Style.Coffee (enumerator in highlight), 174
- Style.Colorful (enumerator in highlight), 174
- Style.Default (enumerator in highlight), 175
- Style.Dracula (enumerator in highlight), 175
- Style.Emacs (enumerator in highlight), 175
- Style.Friendly (enumerator in highlight), 175
- Style.Friendly_Grayscale (enumerator in highlight), 175
- Style.Fruity (enumerator in highlight), 175
- Style.Github_Dark (enumerator in highlight), 175
- Style.Gruvbox_Dark (enumerator in highlight), 175
- Style.Gruvbox_Light (enumerator in highlight), 175
- Style.Igor (enumerator in highlight), 175
- Style.Inkpot (enumerator in highlight), 175
- Style.Lightbulb (enumerator in highlight), 175
- Style.Lilypond (enumerator in highlight), 175
- Style.Lovelace (enumerator in highlight), 175
- Style.Manni (enumerator in highlight), 175
- Style.Material (enumerator in highlight), 175
- Style.Monokai (enumerator in highlight), 175
- Style.Murphy (enumerator in highlight), 175
- Style.Native (enumerator in highlight), 175
- Style.None (enumerator in highlight), 176
- Style.Nord (enumerator in highlight), 176
- Style.Nord_Darker (enumerator in highlight), 176
- Style.One_Dark (enumerator in highlight), 176
- Style.Paraíso_Dark (enumerator in highlight), 176
- Style.Paraíso_Light (enumerator in highlight), 176
- Style.Pastie (enumerator in highlight), 176
- Style.Perldoc (enumerator in highlight), 176
- Style.Rainbow_Dash (enumerator in highlight), 176
- Style.Rrt (enumerator in highlight), 176
- Style.Sas (enumerator in highlight), 176
- Style.Solarized_Dark (enumerator in highlight), 176
- Style.Solarized_Light (enumerator in highlight), 176
- Style.Sphinx (enumerator in highlight), 176
- Style.Staroffice (enumerator in highlight), 176
- Style.Stata_Dark (enumerator in highlight), 176

Style.Stata_Light (enumerator in highlight), 176
 Style.Tango (enumerator in highlight), 176
 Style.Trac (enumerator in highlight), 176
 Style.Vim (enumerator in highlight), 177
 Style.Vs (enumerator in highlight), 177
 Style.Xcode (enumerator in highlight), 177
 Style.Zenburn (enumerator in highlight), 177
 Style_Info (struct in highlight), 177
 Style_Info.background (field in highlight), 177
 Style_Info.base (field in highlight), 177
 Style_Info.dark (field in highlight), 177
 Style_Info.highlight (field in highlight), 177
 Style_Info.line_number (field in highlight), 177
 Style_Info.line_number_background (field in highlight), 177
 Style_Info.name (field in highlight), 177
 Style_Info.rules (field in highlight), 177
 Style_Info.special_background (field in highlight), 177
 Style_Info.special_line_number (field in highlight), 177
 style_name (procedure in highlight), 187
 Style_Rule (struct in highlight), 177
 Style_Rule.definition (field in highlight), 178
 Style_Rule.kind (field in highlight), 177
 STYLES (variable in highlight), 188
 Substitution (struct in readers/commonmark), 298
 Substitution.name (field in readers/commonmark), 298
 Substitution.value (field in readers/commonmark), 298
 Substitution_Hook (struct in readers/rst/inline), 323
 Substitution_Hook.procedure (field in readers/rst/inline), 323
 Substitution_Hook.user (field in readers/rst/inline), 323
 subtree_end (procedure in core), 133
 suggest_budget (procedure in host), 214
 suggest_lexer (procedure in highlight), 187
 symbol (procedure in treesitter), 338
 Symbol (struct in core), 115
 (type in treesitter), 334
 Symbol.hash (field in core), 115
 Symbol.key (field in core), 115
 Symbol.used (field in core), 115
 Symbol.value (field in core), 115
 symbol_get (procedure in core), 133
 symbol_put (procedure in core), 133
 symbol_set (procedure in core), 133
 Symbol_Status (enum in core), 115
 Symbol_Status.Duplicate (enumerator in core), 115
 Symbol_Status.Full (enumerator in core), 115
 Symbol_Status.Ok (enumerator in core), 115

Symbol_Status.Probe_Limit (enumerator in core), 116
 Symbol_Table (struct in core), 116
 Symbol_Table.count (field in core), 116
 Symbol_Table.max_probe (field in core), 116
 Symbol_Table.slots (field in core), 116
 symbol_table_init (procedure in core), 133
 sync_all (procedure in host), 214
 sync_each (procedure in host), 214
 Syntax_Extension (enum in readers/commonmark), 298
 Syntax_Extension.Amsmath (enumerator in readers/commonmark), 298
 Syntax_Extension.Colon_Fence (enumerator in readers/commonmark), 298
 Syntax_Extension.Deflist (enumerator in readers/commonmark), 298
 Syntax_Extension.Dollarmath (enumerator in readers/commonmark), 298
 Syntax_Extension.Fieldlist (enumerator in readers/commonmark), 298
 Syntax_Extension.Html_Admonition (enumerator in readers/commonmark), 298
 Syntax_Extension.Html_Image (enumerator in readers/commonmark), 298
 Syntax_Extension.Linkify (enumerator in readers/commonmark), 299
 Syntax_Extension.Replacements (enumerator in readers/commonmark), 299
 Syntax_Extension.Smartquotes (enumerator in readers/commonmark), 299
 Syntax_Extension.Substitution (enumerator in readers/commonmark), 299
 Syntax_Extension.Tasklist (enumerator in readers/commonmark), 299
 syntax_extension_named (procedure in readers/commonmark), 301
 Syntax_Extensions (bit set in readers/commonmark), 299
 syntax_rule (procedure in readers/commonmark), 301

T

Table (struct in core), 116
 Table.rows (field in core), 116
 Table.used (field in core), 116
 table_decorate (procedure in readers/rst), 318
 Table_Extra (struct in readers/rst), 312
 Table_Extra.align (field in readers/rst), 312
 Table_Extra.auto_widths (field in readers/rst), 312
 Table_Extra.classes (field in readers/rst), 312

- Table_Extra.found (field in readers/rst), 312
- Table_Extra.name (field in readers/rst), 312
- Table_Extra.title (field in readers/rst), 312
- Table_Extra.width (field in readers/rst), 312
- Table_Extra.width_class (field in readers/rst), 312
- Table_Extra.widths (field in readers/rst), 312
- take (procedure in readers/rst), 318
- Target (enum in host), 202
 - (enum in i18n), 219
 - (enum in typst), 342
- Target.Doctest_Block (enumerator in i18n), 219
- Target.Html (enumerator in host), 202
 - (enumerator in typst), 342
- Target.Image (enumerator in i18n), 219
- Target.Index (enumerator in i18n), 219
- Target.Literal_Block (enumerator in i18n), 219
- Target.Pdf (enumerator in host), 202
 - (enumerator in typst), 342
- Target.Raw (enumerator in i18n), 219
- Targets (bit set in i18n), 219
 - (type in odindoc), 249
- Template (struct in i18n), 219
 - (struct in jinja), 235
- Template.catalog (field in i18n), 220
- Template.compiled (field in jinja), 235
- Template.env (field in jinja), 235
- Template.name (field in jinja), 235
- Template.seen (field in i18n), 220
- Template.uuids (field in i18n), 220
- text (procedure group in gettext), 160
 - (procedure in core), 133
 - (procedure in pyscan), 289
 - (procedure in treesitter), 338
- Text (struct in core), 116
- Text.length (field in core), 116
- Text.start (field in core), 116
- text_close (procedure in core), 133
- text_domain (procedure in gettext), 159
- text_open (procedure in core), 134
- text_push (procedure in core), 134
- text_push_byte (procedure in core), 134
- text_push_rune (procedure in core), 134
- text_rewind (procedure in core), 134
- thread_stack_bytes (procedure in host), 214
- to_string (procedure in jinja), 242
- Toc_Parents (enum in readers/sphinx), 330
- Toc_Parents.All (enumerator in readers/sphinx), 330
- Toc_Parents.Domain (enumerator in readers/sphinx), 330
- Toc_Parents.Hide (enumerator in readers/sphinx), 330
- today (procedure in gds), 144
- Token (struct in highlight), 178
- Token.end (field in highlight), 178
- Token.kind (field in highlight), 178
- Token.start (field in highlight), 178
- Token_Kind (enum in highlight), 178
- Token_Kind.Chord_Modifier (enumerator in highlight), 184
- Token_Kind.Comment (enumerator in highlight), 181
- Token_Kind.Comment_Hashbang (enumerator in highlight), 181
- Token_Kind.Comment_Multiline (enumerator in highlight), 181
- Token_Kind.Comment_PreProc (enumerator in highlight), 182
- Token_Kind.Comment_Preproc (enumerator in highlight), 181
- Token_Kind.Comment_Preproc_File (enumerator in highlight), 181
- Token_Kind.Comment_Single (enumerator in highlight), 181
- Token_Kind.Comment_Special (enumerator in highlight), 181
- Token_Kind.Error (enumerator in highlight), 178
- Token_Kind.Escape (enumerator in highlight), 178
- Token_Kind.Generic (enumerator in highlight), 182
- Token_Kind.Generic_Deleted (enumerator in highlight), 182
- Token_Kind.Generic_Emph (enumerator in highlight), 182
- Token_Kind.Generic_Emph_Strong (enumerator in highlight), 182
- Token_Kind.Generic_Error (enumerator in highlight), 182
- Token_Kind.Generic_Heading (enumerator in highlight), 182
- Token_Kind.Generic_Inserted (enumerator in highlight), 182
- Token_Kind.Generic_Output (enumerator in highlight), 182
- Token_Kind.Generic_Prompt (enumerator in highlight), 182
- Token_Kind.Generic_Strong (enumerator in highlight), 182
- Token_Kind.Generic_Subheading (enumerator in highlight), 182
- Token_Kind.Generic_Traceback (enumerator in highlight), 182
- Token_Kind.Generic_Underline (enumerator in highlight), 182
- Token_Kind.Keyword (enumerator in highlight), 178
- Token_Kind.Keyword_Constant (enumerator in highlight), 178
- Token_Kind.Keyword_Declaration (enumerator in highlight), 178
- Token_Kind.Keyword_Namespace (enumerator in highlight), 178
- Token_Kind.Keyword_Pseudo (enumerator in highlight), 178
- Token_Kind.Keyword_Reserved (enumerator in highlight), 179
- Token_Kind.Keyword_Type (enumerator in highlight), 179

- Token_Kind.Literal (enumerator in highlight), 180
- Token_Kind.Literal_Date (enumerator in highlight), 180
- Token_Kind.Literal_Scalar_Plain (enumerator in highlight), 182
- Token_Kind.Name (enumerator in highlight), 179
- Token_Kind.Name_Attribute (enumerator in highlight), 179
- Token_Kind.Name_Backslash_Reference (enumerator in highlight), 183
- Token_Kind.Name_Builtin (enumerator in highlight), 179
- Token_Kind.Name_Builtin_Articulation (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Clef (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Context (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Context_Property (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Dynamic (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Grob (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Grob_Property (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Header_Variable (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Markup_Command (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Music_Command (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Music_Function (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Paper_Variable (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Pseudo (enumerator in highlight), 179
- Token_Kind.Name_Builtin_Repeat_Type (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Scale (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Scheme_Builtin (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Scheme_Function (enumerator in highlight), 183
- Token_Kind.Name_Builtin_Translator (enumerator in highlight), 183
- Token_Kind.Name_Class (enumerator in highlight), 179
- Token_Kind.Name_Constant (enumerator in highlight), 179
- Token_Kind.Name_Decorator (enumerator in highlight), 179
- Token_Kind.Name_Entity (enumerator in highlight), 179
- Token_Kind.Name_Exception (enumerator in highlight), 179
- Token_Kind.Name_Function (enumerator in highlight), 179
- Token_Kind.Name_Function_Magic (enumerator in highlight), 179
- Token_Kind.Name_Label (enumerator in highlight), 179
- Token_Kind.Name_Lvalue (enumerator in highlight), 182
- Token_Kind.Name_Namespace (enumerator in highlight), 179
- Token_Kind.Name_Other (enumerator in highlight), 179
- Token_Kind.Name_Property (enumerator in highlight), 179
- Token_Kind.Name_Tag (enumerator in highlight), 179
- Token_Kind.Name_Variable (enumerator in highlight), 179
- Token_Kind.Name_Variable_Class (enumerator in highlight), 180
- Token_Kind.Name_Variable_Global (enumerator in highlight), 180
- Token_Kind.Name_Variable_Instance (enumerator in highlight), 180
- Token_Kind.Name_Variable_Magic (enumerator in highlight), 180
- Token_Kind.Number (enumerator in highlight), 181
- Token_Kind.Number_Bin (enumerator in highlight), 181
- Token_Kind.Number_Float (enumerator in highlight), 181
- Token_Kind.Number_Hex (enumerator in highlight), 181
- Token_Kind.Number_Integer (enumerator in highlight), 181
- Token_Kind.Number_Integer_Long (enumerator in highlight), 181
- Token_Kind.Number_Oct (enumerator in highlight), 181
- Token_Kind.Operator (enumerator in highlight), 181
- Token_Kind.Operator_Word (enumerator in highlight), 181
- Token_Kind.Other (enumerator in highlight), 178
- Token_Kind.Pitch (enumerator in highlight), 184
- Token_Kind.Punctuation (enumerator in highlight), 181
- Token_Kind.Punctuation_Indicator (enumerator in highlight), 182
- Token_Kind.Punctuation_Marker (enumerator in highlight), 181
- Token_Kind.String (enumerator in highlight), 180
- Token_Kind.String_Affix (enumerator in highlight), 180
- Token_Kind.String_Backtick (enumerator in highlight), 180

Token_Kind.String_Char (enumerator in highlight), 180
 Token_Kind.String_Delimiter (enumerator in highlight), 180
 Token_Kind.String_Doc (enumerator in highlight), 180
 Token_Kind.String_Double (enumerator in highlight), 180
 Token_Kind.String_Escape (enumerator in highlight), 180
 Token_Kind.String_Heredoc (enumerator in highlight), 180
 Token_Kind.String_Interpol (enumerator in highlight), 180
 Token_Kind.String_Other (enumerator in highlight), 180
 Token_Kind.String_Regex (enumerator in highlight), 180
 Token_Kind.String_Single (enumerator in highlight), 180
 Token_Kind.String_Symbol (enumerator in highlight), 181
 Token_Kind.Text (enumerator in highlight), 178
 Token_Kind.Token_Number (enumerator in highlight), 184
 Token_Kind.Token_String (enumerator in highlight), 183
 Token_Kind.Token_String_Escape (enumerator in highlight), 184
 Token_Kind.Token_String_Symbol (enumerator in highlight), 184
 Token_Kind.Whitespace (enumerator in highlight), 178
 token_text (procedure in highlight), 187
 tokenize (procedure in highlight), 187
 Toml_Kind (enum in project), 270
 Toml_Kind.Array (enumerator in project), 270
 Toml_Kind.Boolean (enumerator in project), 270
 Toml_Kind.Datetime (enumerator in project), 270
 Toml_Kind.Float (enumerator in project), 270
 Toml_Kind.Integer (enumerator in project), 270
 Toml_Kind.String (enumerator in project), 270
 Toml_Kind.Table (enumerator in project), 270
 Toml_Value (struct in project), 270
 Toml_Value.boolean (field in project), 271
 Toml_Value.defined (field in project), 271
 Toml_Value.float (field in project), 271
 Toml_Value.inline (field in project), 271
 Toml_Value.integer (field in project), 270
 Toml_Value.items (field in project), 271
 Toml_Value.keys (field in project), 271
 Toml_Value.kind (field in project), 270
 Toml_Value.line (field in project), 271
 Toml_Value.text (field in project), 270
 Toml_Value.values (field in project), 271

top (procedure in core), 134
 transform_document (procedure in core), 134
 translated (procedure in gettext), 159
 translation (procedure in gettext), 159
 Translation (struct in i18n), 220
 Translation.catalog (field in i18n), 220
 Translation.images (field in i18n), 221
 Translation.kept (field in i18n), 221
 Translation.limits (field in i18n), 220
 Translation.policy (field in i18n), 220
 Translation.progress (field in i18n), 220
 Translation.read (field in i18n), 220
 Translation.read_inline (field in i18n), 220
 Translation.source_name (field in i18n), 220
 Translation.targets (field in i18n), 220
 Translation.translated (field in i18n), 221
 translation_pass (procedure in i18n), 224
 Translator (struct in gettext), 153
 Translator allocator (field in gettext), 153
 Translator.bound (field in gettext), 153
 Translator.domain (field in gettext), 153
 Translator.language (field in gettext), 153
 Tree (struct in treesitter), 334
 tree_delete (procedure in treesitter), 339
 treesitter (package), 333
 trim_blank_end (procedure in readers/rst), 318
 trusted_policy (procedure in core), 134
 truthy (procedure in jinja), 242
 try_unparse (procedure in pyscan), 289
 ts (foreign import in treesitter), 339
 tuple (procedure in jinja), 242
 Type_Alias (struct in autodoc), 80
 Type_Alias.name (field in autodoc), 80
 Type_Alias.target (field in autodoc), 80
 type_name (procedure in jinja), 242
 typst (package), 339
 typst_highlighter (procedure in highlight/guidedoc), 188

U

UI_DOMAIN (constant in i18n), 225
 UI_Problem (struct in i18n), 221
 UI_Problem.error (field in i18n), 221
 UI_Problem.path (field in i18n), 221
 ui_text (procedure in i18n), 224
 ui_translator (procedure in i18n), 224
 undefined (procedure in jinja), 242
 Undefined (struct in jinja), 235
 Undefined.hint (field in jinja), 235
 Undefined.name (field in jinja), 235
 Undefined.owner (field in jinja), 235
 Undefined_Mode (enum in jinja), 236
 Undefined_Mode.Chainable (enumerator in jinja), 236
 Undefined_Mode.Debug (enumerator in jinja), 236
 Undefined_Mode.Default (enumerator in jinja), 236
 Undefined_Mode.Strict (enumerator in jinja), 236
 unescape (procedure in readers/rst/inline), 324
 Unit (enum in i18n), 221
 Unit.Alt (enumerator in i18n), 221

Unit.Caption (enumerator in i18n), 222
 Unit.Contents (enumerator in i18n), 222
 Unit.Entry (enumerator in i18n), 222
 Unit.Graphviz (enumerator in i18n), 221
 Unit.Index (enumerator in i18n), 222
 Unit.Inline (enumerator in i18n), 221
 Unit.Literal (enumerator in i18n), 221
 Unit.Meta (enumerator in i18n), 222
 Unit.None (enumerator in i18n), 221
 Unit.Parsed (enumerator in i18n), 221
 unit_of (procedure in i18n), 224
 unload (procedure in project), 273
 unparse (procedure in pyscan), 289
 Unparse_Mode (enum in pyscan), 285
 Unparse_Mode.Python (enumerator in
 pyscan), 285
 Unparse_Mode.Sphinx (enumerator in
 pyscan), 285
 Unparse_Mode.Sphinx_Repr (enumerator
 in pyscan), 285
 unwrap_node (procedure in core), 134

V

validate (procedure in core), 135
 validate_registry (procedure in core), 135
 Value (union in jinja), 236
 values_equal (procedure in jinja), 242
 VERSION (constant in cli), 81
 (constant in core), 136
 (constant in readers/commonmark), 301
 (constant in readers/rst), 318
 (constant in readers/sphinx), 331
 (constant in render/html), 331
 (constant in render/text), 332
 (constant in render/typst), 333
 version (procedure in graphdog), 167
 VERSION_SUFFIX (constant in cli), 81
 version_text (procedure in cli), 81
 view (procedure in core), 135
 View (struct in core), 116
 View.snap (field in core), 116
 View.sources (field in core), 116

W

was_interrupted (procedure in host), 214
 with_context (procedure group in gettext),
 160
 Workspace (struct in core), 116
 Workspace.active (field in core), 117
 Workspace.banks (field in core), 117
 Workspace.generation (field in core), 117
 Workspace.keep_errors (field in core), 117
 Workspace.output (field in core), 117
 Workspace.ready (field in core), 117
 Workspace.reports (field in core), 117
 Workspace.revisions (field in core), 117
 Workspace.scratch (field in core), 117
 Workspace.sources (field in core), 117
 Workspace.sources_used (field in core),
 117
 workspace_size (procedure in core), 135
 Wrapper (enum in pyscan), 285
 Wrapper.Cached_Property (enumerator in
 pyscan), 285
 Wrapper.Classmethod (enumerator in
 pyscan), 285
 Wrapper.None (enumerator in pyscan), 285

Wrapper.Property (enumerator in pyscan),
 285
 Wrapper.Staticmethod (enumerator in
 pyscan), 285
 Wrapper.Unknown (enumerator in
 pyscan), 286
 write (procedure in inventory), 228
 write_css (procedure in highlight), 187
 write_figure_filename (procedure in i18n),
 225
 write_html (procedure in highlight), 187
 write_html_lines (procedure in highlight),
 187
 write_id (procedure in core), 135
 write_message (procedure in i18n), 225
 write_mo (procedure in gettext), 160
 write_object (procedure in core), 135
 Write_Options (struct in gettext), 153
 Write_Options.obsolete (field in gettext),
 154
 Write_Options.previous (field in gettext),
 154
 Write_Options.references (field in gettext),
 153
 Write_Options.width (field in gettext), 153
 write_po (procedure in gettext), 160
 write_problem (procedure in highlight),
 188
 write_template (procedure in i18n), 225
 write_typst (procedure in highlight), 188
 write_typst_theme (procedure in
 highlight), 188
 write_unsynced (procedure in host), 214