Guidedog Discussions GDS 0011

Sphinx's built-in extensions

STATE
committed
TYPE
Standards Track
AUTHORS
Vikrant Rathore
CREATED
2026-10-15
UPDATED
2026-10-31
DISCUSSION
No external discussion URL assigned
LABELS
compatibility, architecture, build

Record status

The catalog defines the review process and the meaning of each state. This numbered record forms part of the catalog.

Contents

1. Abstract 1
2. Motivation and scope 1
3. Specification 2
4. Rationale and alternatives 7
5. Security and operational considerations 8
6. Backwards compatibility 8
7. Acceptance criteria and implementation status 8
8. Open questions 8
9. Review history 8
10. References 8

1. Abstract

Guidedog supports all of reStructuredText and every extension Sphinx ships in `sphinx.ext`. A project lists an extension by its Sphinx name in `extensions`, and the extension works. It works in Guidedog's own way. Guidedog never runs Python, and its output is not byte-identical with Sphinx's.

This record designs the nine extensions Guidedog did not yet have: `napoleon`, `autosummary`, `viewcode`, `linkcode`, `inheritance_diagram`, `apidoc`, `coverage`, `duration` and `imgconverter`. For each one it states what Sphinx does, what Guidedog does instead, the settings, the output in HTML, PDF and text, the diagnostics, the limits, and what `guidedog migrate` does with a `conf.py` that uses it.

2. Motivation and scope

A Sphinx project names extensions in `conf.py`. Before this record, Guidedog built eleven of the built-in ones and refused the others with `config.invalid`. `guidedog migrate` left them out, so a migrated project lost their directives silently until a build reported them as unknown. A project that used `napoleon` for Google-style docstrings, or `autosummary` for its API tables, could not move.

The rule we adopt is simple to state and to check: every reStructuredText construct and every built-in Sphinx extension is supported. “Supported” means the extension’s directives, roles, settings and builders exist, and they produce the result an author expects. It does not mean the same bytes. Where Sphinx needs Python at build time, Guidedog uses the source scan that autodoc already uses, the linked Graphviz, and the existing highlighter.
Third-party extensions (`sphinxcontrib.*`, `sphinx_design`, …) are outside this record. They remain the business of declared object types (GDS 0006) and build plugins (GDS 0010).

3. Specification

3.1. Common ground

Every extension below is turned on by its Sphinx name in `extensions`. Its settings keep Sphinx’s names, types and defaults, so a migrated `conf.toml` reads like the `conf.py` it came from. A setting of an extension that is not listed is accepted and has no effect, as in Sphinx.
Four shared parts carry the work:

- **The source scan.** `gd-pyscan` reads Python source with tree-sitter and resolves names across the folders of `autodoc_source_paths`. `gd-autodoc` turns what it finds into the `py:` directives autodoc writes. Nothing is imported. The limits of the scan are the limits of autodoc: an object made at run time is not seen.
- **Generated documents.** A generated page is a reStructuredText document held in memory, with the source files it was made from as its dependencies. The `rust_autoapi_*` pages already work this way. Generated pages are never written into the source folder. Editing a Python file reads its pages again.
- **Graphviz.** Drawings use the linked Graphviz, as `graphviz` directives do. They are SVG in HTML and in the PDF.
- **The highlighter.** Source pages use `gd-highlight`, the highlighter of every code block.

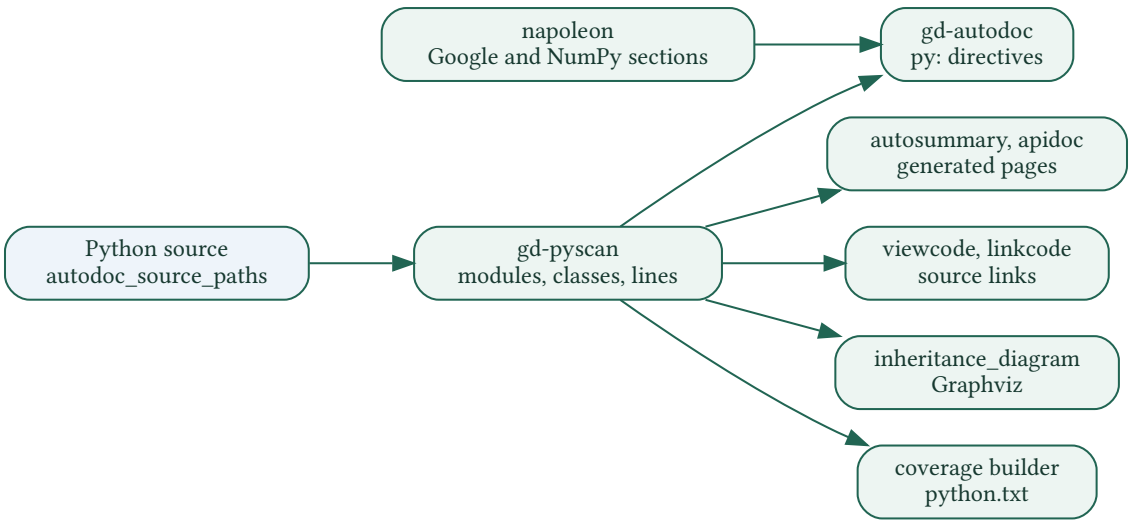

Fig. 13 Python source is scanned once; the extensions read the scan.

Diagnostics follow the house style: a code, a title, the location with an excerpt, a plain explanation and a hint. Each extension has its own code prefix, listed below. `suppress_warnings` silences them by that prefix (`napoleon`, `napoleon.section`).

3.2. sphinx.ext.napoleon

Sphinx. Napoleon rewrites Google-style and NumPy-style docstrings into reStructuredText fields before autodoc reads them. `Args:` becomes `:param x:` and `:type x:` fields, `Returns:` becomes `:returns:` and `:rtype:`, `Example:` becomes a rubric or an admonition, and so on. It hooks `autodoc-process-docstring` and `autodoc-skip-member`.
Guidedog. `gd-autodoc` prepares each docstring as Sphinx prepares it. Napoleon adds one step to that preparation: when the extension is listed, a docstring is converted from the Google and NumPy section

syntax into the same fields autodoc already renders. The converter is a pure function from docstring text and settings to reStructuredText lines. It works line by line and keeps a map from output lines to input lines, so a later markup error points at the docstring line that caused it. The member filter of `napoleon_include_init_with_doc`, `napoleon_include_private_with_doc` and `napoleon_include_special_with_doc` joins autodoc's own member selection.
For example, this docstring:

```
def area(width, height):
    """Compute an area.

    Args:
        width (float): The width.
        height (float): The height.

    Returns:
        float: The product.
    """
```

reads as if it were written:

```
Compute an area.

:param width: The width.
:type width: float
:param height: The height.
:type height: float
:returns: The product.
:rtype: float
```

The settings are Sphinx's sixteen: `napoleon_google_docstring` (true), `napoleon_numpy_docstring` (true), `napoleon_include_init_with_doc`, `napoleon_include_private_with_doc`, `napoleon_include_special_with_doc`, `napoleon_use_admonition_for_examples`,
`napoleon_use_admonition_for_notes`, `napoleon_use_admonition_for_references`, `napoleon_use_ivar` (all false), `napoleon_use_param`, `napoleon_use_rtype`, `napoleon_use_keyword` (true), `napoleon_preprocess_types` (false), `napoleon_type_aliases` (a table of strings, empty), `napoleon_custom_sections` and `napoleon_attr_annotations` (true). `napoleon_custom_sections` is a list whose items are a section name, or a pair of a name and the section it behaves like (another section's name, `params_style` or `returns_style`):

```
napoleon_custom_sections = ["Side Effects", ["Arguments", "params_style"]]
```

Output. The fields are those of autodoc, so HTML, PDF and text show what an autodoc field list shows. Admonitions are ordinary admonitions.
Diagnostics. `napoleon.section` reports a section whose body is not indented or a custom section that names an unknown section. `napoleon.field` reports a parameter line it cannot split into a name, a type and a description. `napoleon.type` reports a malformed type specification when `napoleon_preprocess_types` is on (an unclosed brace or quote). Each is a warning; the docstring is kept as written from that line on.
Limits. A docstring is converted only when autodoc reads it from source. A docstring written by hand inside a `py:function` directive is not converted, as in Sphinx. Conversion is linear in the docstring's length, and a docstring is bounded by autodoc's limits.
Migration. The extension name and all sixteen settings convert as written. `None` for `napoleon_type_aliases` or `napoleon_custom_sections` is the default and is left out.

3.3. sphinx.ext.autosummary

Sphinx. The `autosummary` directive makes a table of objects, each with its signature and the first sentence of its docstring. With `:toctree:` it also generates a stub page for each object, from Jinja templates, and lists the pages in a hidden toctree. `:recursive:` walks a package's submodules. The `autolink` role links a name when an object of that name exists.
Guidedog. The table is made from the source scan: the summary is the first sentence of the docstring, with Sphinx's rule for where a sentence ends, and the signature is shortened as the `:signatures:` option says (`long`, `short` or `none`; `:nosignatures:` is `none`). Stub pages are generated documents under the `:toctree:` folder,

named as Sphinx names them (`<folder>/<full.name>`, or by `autosummary_filename_map`). Their text comes from the templates `autosummary/base.rst`, `module.rst` and `class.rst`, rendered with Guidedog's Jinja and the variables Sphinx passes (`fullname`, `objname`, `members`, `functions`, `classes`, `exceptions`, `attributes`, `modules`, …). A project overrides a template by placing a file of the same name under `templates_path`, as in Sphinx. `autosummary_context` adds variables.
A page is generated, never written. Therefore `autosummary_generate_overwrite` has nothing to protect and is accepted without effect. A stub that the project also wrote by hand at the same document name wins over the generated one, with the note `autosummary.exists`.
The settings are `autosummary_context` (a table, empty), `autosummary_filename_map` (a table of strings, empty), `autosummary_generate` (true, false, or a list of source files whose directives generate pages; default true), `autosummary_generate_overwrite` (true), `autosummary_mock_imports` (a list, empty: follow `autodoc_mock_imports`), `autosummary_imported_members` (false) and `autosummary_ignore_module_all` (true). The directive's options are Sphinx's: `:toctree:`, `:caption:`, `:template:`, `:recursive:`, `:signatures:`, `:nosignatures:` and `:class:`.
Output. In HTML, the table is a two-column table with the `autosummary longtable` classes and links to the objects. In the PDF, it is a Typst table with internal links. In text, it is a plain table. The stub pages are ordinary pages in every builder.
Diagnostics. `autosummary.import` reports an entry the source scan cannot find (the object name and the folders searched). `autosummary.template` reports a missing template or a template error with its line. `autosummary.toctree` reports a `:toctree:` folder outside the source folder. `autosummary.exists` is the note above. `autosummary.limit` reports a recursive walk cut short.
Limits. A recursive walk visits at most 10,000 modules and 64 levels of packages. A stub page is bounded by the engine limits like any document.
Migration. The extension name and the seven settings convert. A list for `autosummary_generate` converts as a list of strings.

3.4. sphinx.ext.apidoc

Sphinx. Since Sphinx 8.2, `apidoc` runs `sphinx-apidoc` during the build: for each entry of `apidoc_modules` it writes `.rst` files with `automodule` directives into a destination folder.
Guidedog. The same pages are generated documents, made from the source scan. Each entry of `apidoc_modules` gives a package folder (`path`, relative to `conf.toml`) and a document folder (`destination`, relative to the source folder). Guidedog walks the package, skips what `exclude_patterns` matches, and generates a page per package (and per module with `separate_modules`) with the `automodule` directives and options `sphinx-apidoc` writes, plus `<destination>/modules` as the table of contents. The pages are never written to disk, so a source tree stays clean.

```
[[apidoc_modules]]
path = "../src/shapes"
destination = "api"
separate_modules = true
```

An entry may override any of the general settings by its short name: `exclude_patterns`, `max_depth`, `follow_links`, `separate_modules`, `include_private`, `no_headings`, `module_first`, `implicit_namespaces` and `automodule_options`. The general settings are `apidoc_exclude_patterns` (empty), `apidoc_max_depth` (4), `apidoc_follow_links`, `apidoc_separate_modules`, `apidoc_include_private`, `apidoc_no_headings`, `apidoc_module_first`, `apidoc_implicit_namespaces` (all false) and `apidoc_automodule_options` (`members`, `show-inheritance`, `undoc-members`). The package folder is also added to the folders autodoc reads, so the generated directives find their modules without a separate `autodoc_source_paths` entry.
Output. Generated pages are ordinary pages in every builder.
Diagnostics. Loading `conf.toml` refuses an entry without `path` or `destination`, a destination outside the source folder, and an unknown key (with "Did you mean"). At build time, `apidoc.path` reports a package folder that does not exist, `apidoc.exists` notes a hand-written page that wins over a generated one, and `apidoc.limit` reports a walk cut short.
Limits. A walk visits at most 10,000 modules. `follow_links` never leaves the readable folders (GDS 0004).
Migration. The extension name and the settings convert. A Python set, such as Sphinx's default for `apidoc_automodule_options`, becomes a list.

3.5. sphinx.ext.viewcode

Sphinx. Viewcode imports each documented module, finds its source, and writes a highlighted copy under `_modules/`. Each object description gets a `[source]` link to its lines, and each source block gets a `[docs]` link back.

Guidedog. The source scan already knows each object's file and line span. For every module with a documented object, the HTML builders write `_modules/<package>/<module>.html`: the module's source highlighted by `gd-highlight`, with an anchor per top-level class and function, and `_modules/index.html` listing the modules. Each `py:` object description whose module is in the scan gets a `[source]` link, and each anchored block in the source page gets a `[docs]` link back to the description. Both directions use the page's relative URLs, so they work under `dirhtml` and on a local file system.

The settings are `viewcode_follow_imported_members` (true: an object documented through a re-export links to the module that defines it), `viewcode_enable_epub` (false; accepted, since Guidedog has no EPUB builder) and `viewcode_line_numbers` (false: line numbers in the source pages). The old name `viewcode_import` is read as `viewcode_follow_imported_members`.

Output. HTML only, as in Sphinx. `singlehtml`, the PDF and text add no links and no source pages; a book does not need a second copy of the code.

Diagnostics. `viewcode.source` notes a documented module the scan could not find, so its objects have no `[source]` link. `viewcode.limit` reports a module larger than the limit below, shown without a source page.

Limits. A source page is made from a file of at most 8 MiB, the limit of any source read.

Migration. The extension name and settings convert; `viewcode_import` becomes `viewcode_follow_imported_members`.

3.6. sphinx.ext.linkcode

Sphinx. Linkcode calls the project's Python function `linkcode_resolve(domain, info)` for each object description, and adds a `[source]` link to the URL it returns.

Guidedog. A function cannot be data, so Guidedog takes a URL template instead:

```
linkcode_url = "https://github.com/o/r/blob/{branch}/{path}#L{line}-L{end_line}"
```

The template's placeholders are:

Table 24 Placeholders of `linkcode_url`

Placeholder	Value
`{domain}`	The object's domain: `py`, `c`, `cpp`, `js`, `rust`, `odin`.
`{module}`	The Python module, or the Rust or Odin crate or package path.
`{fullname}`	The object's name within its module, such as `Square.area`.
`{path}`	The object's source file, relative to `linkcode_root`, with `/` separators.
`{line}`	The first line of the object's definition, counted from 1.
`{end_line}`	The last line of the definition.
`{branch}`	`linkcode_branch`, or the branch read from the repository.
`{{` and `}}`	A literal brace.

`linkcode_root` names the folder that `{path}` is relative to, relative to the source folder. Empty, it is the nearest folder above the source folder that holds `.git`. `linkcode_branch` is the text of `{branch}`. Empty, it is the branch named in that repository's `.git/HEAD`, or the commit id when `HEAD` is detached. A link is added only when every placeholder the template uses has a value for that object. A `py:` object found by the source scan has them all. A `c:` or `cpp:` object described by hand has no file, so only a template that uses nothing beyond `{domain}`, `{module}`, `{fullname}` and `{branch}` links it.

Output. HTML builders add the `[source]` link after the signature. The PDF and text add nothing.

Diagnostics. Loading `conf.toml` refuses a template with an unknown placeholder or an unclosed brace. Listing the extension without `linkcode_url` is a warning, `config.ext.linkcode`, and adds no links. At build time, `linkcode.branch` reports that `{branch}` is used but no branch is set and no repository was found. `linkcode.outside` notes an object whose file lies outside `linkcode_root`.

Limits. None beyond the source scan's.
Migration. The extension name converts. A `linkcode_resolve` defined in `conf.py` cannot be read as data. `guidedog migrate` reports it as `migrate.linkcode`, with a hint showing a `linkcode_url` template to write in its place.

3.7. sphinx.ext.inheritance_diagram

Sphinx. The `inheritance-diagram` directive imports the named classes or modules, walks their bases, and draws the graph with Graphviz, with clickable nodes.
Guidedog. The classes come from the source scan: their bases as written, resolved across the scanned modules. A base the scan cannot resolve (a class from a package outside the source folders) is drawn as a plain node with its written name and no link. The graph is written as DOT and drawn by the linked Graphviz, so it needs no `dot` program. The directive's options are Sphinx's: `:parts:`, `:private-bases:`, `:caption:`, `:top-classes:` and `:include-subclasses:`. The settings are `inheritance_graph_attrs`, `inheritance_node_attrs`, `inheritance_edge_attrs` (tables of Graphviz attributes, empty) and `inheritance_alias` (a table from full class names to the names shown, empty).
Output. HTML embeds the adaptive SVG with `<object type="image/svg+xml">` when nodes carry links (each node linked to its class description when there is one) or `<img>` when unlinked, with the `graphviz-adaptive` behaviour of other drawings. The PDF embeds a separate unlinked light-palette SVG. Text shows the list of classes and their bases.
Diagnostics. `inheritance.class` reports a name that is neither a class nor a module in the scan. `inheritance.limit` reports a graph that exceeds the limit below. Loading `conf.toml` refuses an attribute name that is not a Graphviz identifier, so attributes cannot inject DOT.
Limits. A diagram has at most 1,000 classes. Graphviz runs under the confinement of every drawing (GDS 0004).
Migration. The extension name and the four settings convert. A `dict(rankdir="LR")` call with literal keyword arguments converts like a dictionary.

3.8. sphinx.ext.coverage

Sphinx. The `coverage` builder imports the documented modules and lists every public function, class and method that has no description. It writes `python.txt` (and `c.txt` for C headers) in its output folder.
Guidedog. `guidedog build coverage` reads the project like any builder, then compares the objects of the source scan with the `py:` descriptions of the read documents. It writes `_build/coverage/python.txt` in Sphinx's layout: the headline (`coverage_write_headline`), each module's undocumented functions, classes and methods, and the statistics table (`coverage_statistics_to_report`). With `coverage_c_path` and `coverage_c_regexes` it reads C headers with those patterns and writes `c.txt`. The modules checked are `coverage_modules` with their submodules, or, when it is empty, the modules the documents describe.
The settings are `coverage_modules`, `coverage_ignore_modules`, `coverage_ignore_functions`, `coverage_ignore_classes`, `coverage_ignore_pyobjects`, `coverage_c_path` (lists, empty), `coverage_c_regexes` (a table of patterns, empty), `coverage_ignore_c_items` (a table of lists, empty), `coverage_write_headline`, `coverage_statistics_to_report`, `coverage_statistics_to_stdout` (true), `coverage_skip_undoc_in_source` and `coverage_show_missing_items` (false). The ignore patterns are regular expressions in the subset `nitpick_ignore_regex` uses. Like Python's `re.match` they match at the start of a name; `coverage_ignore_pyobjects` matches anywhere in it, like `re.search`.
Output. `python.txt` and `c.txt`. With `coverage_statistics_to_stdout` the table is also printed when the build ends. The coverage builder writes no pages.
Diagnostics. `coverage.module` reports a module of `coverage_modules` that the scan cannot find. With `coverage_show_missing_items`, `coverage.undocumented` notes each missing object. Loading `conf.toml` refuses a pattern that does not compile, with the pattern and the reason.
Limits. A C header is read up to the source size limit. Patterns have the step bound of the pattern engine.
Migration. The extension name and the settings convert. `guidedog build coverage` replaces `sphinx-build -b coverage`.

3.9. sphinx.ext.duration

Sphinx. Duration measures how long each document takes to read and prints the slowest ones when the build ends.

Guidedog. The build measures reading per document with a monotonic clock. When the extension is listed, the build summary shows the total reading time (`duration_print_total`) and the `duration_n_slowest` slowest documents (`duration_print_slowest`), and the output folder receives `duration_write_json`, a JSON object from document names to seconds. A document read from the cache took no time to read and is not listed. A document slower than `duration_limit` seconds is a warning.
The settings are `duration_print_total` (true), `duration_print_slowest` (true), `duration_n_slowest` (5; 0 lists every document), `duration_write_json` (`sphinx-reading-durations.json`; empty writes nothing) and `duration_limit` (seconds, a whole or decimal number; unset means no limit).
Output. A summary on the terminal and one JSON file per builder output. Documents are unchanged.
Diagnostics. `duration.limit` names the document, its time and the limit. Loading `conf.toml` refuses a negative `duration_n_slowest`, a `duration_limit` that is not a positive number, and a `duration_write_json` path that leaves the output folder.
Limits. Times differ from run to run. They are evidence about one build on one machine, not a guarantee.
Migration. The extension name and settings convert. `duration_write_json = None` becomes `""`.

3.10. sphinx.ext.imgconverter

Sphinx. Imgconverter runs ImageMagick to convert images a builder cannot use, such as GIF or SVG for LaTeX.
Guidedog. The PDF is typeset by Typst, which reads SVG, PNG, JPEG, GIF, WebP and PDF directly. HTML reads SVG, PNG, JPEG, GIF and WebP directly. There is nothing to convert, so the extension is accepted and runs nothing. Where an image names several candidates (`picture.*`), each builder picks the first one it reads. An image in a format the PDF cannot read (TIFF or BMP, for example) is reported, and its alternative text is shown instead.
The settings `image_converter` (`convert`) and `image_converter_args` (empty) are accepted and have no effect. No program is ever started.
Output. Unchanged images in every builder.
Diagnostics. `imgconverter.format` names the image and its format, with the hint to provide a PNG, JPEG, GIF, WebP or SVG version.
Limits. None.
Migration. The extension name and settings convert.

3.11. sphinx.ext.doctest

`doctest` is already supported, and this record restates its boundary. Its directives (`doctest`, `testcode`, `testoutput`, `testsetup`, `testcleanup`, `doctest-block`) are shown with their code highlighted. Guidedog never runs Python, so it never runs the tests. There is no `doctest` builder; a project runs its doctests with Python's own tools.

4. Rationale and alternatives

Run Python for these extensions. Sphinx's own code would give Sphinx's output exactly. It would also make every build depend on an interpreter, a virtual environment and the import side effects of the documented package. A build that imports a package runs its code, which breaks the trust model of GDS 0004. The source scan already serves autodoc. Reusing it costs less and keeps builds reproducible.
Leave the nine out. That is what the boundary was. A migrated project then lost content silently or failed to build. The extensions are popular, and each one has a source-only design that serves its purpose.
A Python callback for linkcode. `linkcode_resolve` is the one place where Sphinx asks for code in configuration. Almost every published resolver builds a URL from a module, a file and lines. A template with defined placeholders covers that case as data, and its mistakes are found when `conf.toml` loads, not in the middle of a build.
Write generated pages to disk. `sphinx-apidoc` and `autosummary` write `.rst` files that projects then commit or ignore. Generated documents in memory give the same pages without stale files, and they follow the incremental rules every other document follows.

5. Security and operational considerations

No extension runs project code or starts a program. Every file they read lies inside the readable folders (GDS 0004); `--untrusted` narrows those as for any other input. The source scan, Graphviz and the highlighter keep their limits. The `linkcode` template becomes URLs in pages, so it goes through the URL checks of every link; `--untrusted` drops a link whose scheme is not safe.

6. Backwards compatibility

A `conf.toml` that loaded before still loads, with the same meaning. A `conf.toml` that listed one of the nine extensions was refused before and loads now. Settings with the `napoleon_` and `autosummary_` prefixes were ignored with a warning before; they are now read and checked, so a misspelled one is an error with a "Did you mean" hint. The reading-settings digest includes the new settings that change how documents read, so changing one reads the documents again.

7. Acceptance criteria and implementation status

The record is accepted when the settings, the configuration checks and the migration exist, with tests. That part is implemented: `gd-conf` lists the nine extensions as built in and holds every setting above with its default, type and check, and `gd-migrate` converts the names and settings and reports `linkcode_resolve`.
The record becomes `committed` when each extension behaves as specified, with tests per extension in the crates that implement it, an integration test that builds a small Python project with all nine listed, and the manual's pages for each extension.

8. Open questions

- Should `linkcode` accept a template per domain? One template serves the common case. A table keyed by domain could be added later without breaking the string form.
- Should `viewcode` pages exist for Rust and Odin sources? The scan for those languages knows the spans too. This record keeps Sphinx's Python scope.

9. Review history

2026-10-15: Drafted. The maintainer decided that Guidedog supports all of reStructuredText and all of Sphinx's built-in extensions, in Guidedog's own way.

10. References

- Sphinx-style projects and recoverable publication sets the rule of capability over imitation.
- Memory budgets and what a build may read defines the readable folders and the limits.
- Typed build plugins is the route for third-party extensions.
- Sphinx 9.1.0, `sphinx/ext/*.py` and `doc/usage/extensions/*.rst`, is the behaviour reference.
- `crates/gd-conf/src/settings.rs` lists every setting; `crates/gd-migrate` converts them.

Lifecycle event

```
2026-10-15: prediscussion -> discussion. Assigned a permanent number and
opened for discussion.
```

Lifecycle event

```
2026-10-15: discussion -> accepted. Maintainer decided that Guidedog supports
all of Sphinx's built-in extensions; settings and migration implemented
```

Lifecycle event

```
2026-10-15: accepted -> committed. All nine extensions implemented and
verified. evidence/builtin-extensions-20261015.rst
```