Khronos Sample Specification

Jon Leech & Neil Trevett

Version 1.0.1, 2024-04-19 14:14:15Z: from git branch: main commit:
bf2067dd7d43ff28f314446c676250a96754a486

Table of Contents

1. Preamble . . . 1
2. Introduction . . . 2
   2.1. Document Conventions . . . 2
3. Fundamentals . . . 5
4. Licenses and Contributor License Agreements . . . 6
   4.1. Choice of License . . . 6
   4.2. Including a License and Copyright Statement in a File . . . 7
   4.3. Documenting Licenses . . . 8
   4.4. Contributor License Agreements . . . 9
   4.5. Licenses for Ratified and Non-Ratified Specification Artifacts . . . 10
   4.6. Confidential Repository Notice . . . 11
5. Repository Structure . . . 13
   5.1. Boilerplate Files . . . 13
   5.2. Specification Files . . . 13
   5.3. Configuration Files . . . 14
6. Repository Management . . . 17
   6.1. Private and Public Repositories . . . 17
   6.2. Repository Creation and Naming . . . 17
   6.3. Synchronizing Repositories . . . 18
   6.4. Managing Community Interaction . . . 18
   6.5. Publication Strategies . . . 19
7. Continuous Integration . . . 21
   7.1. Repository CI Configuration . . . 21
   7.2. Repository CI Scripts and Jobs . . . 21
   7.3. Sample CI Runtime . . . 21
   7.4. Sample CI Outputs . . . 22
8. Asciidoc Markup for Authoring . . . 23
   8.1. Simplest: GitHub-Flavored Asciidoc . . . 23
   8.2. Intermediate: Unextended Asciidoctor . . . 23
   8.3. Most Complex: Asciidoctor with Extensions . . . 24
   8.4. On the Horizon: Antora Site Generation . . . 25
Appendix A: Lexicon . . . 27
   Glossary . . . 27
   Common Abbreviations . . . 27
Appendix B: Credits (Informative) . . . 28

Chapter 1. Preamble

Chapter 2. Introduction

This document, referred to as the *Sample Specification* or just the “Specification” hereafter, is a reference for creating repositories containing new Khronos API or language specifications, referred to as *Khronos specifications* hereafter.

The Sample Specification is not, itself, a ratified Khronos product. Its purpose is to

- Provide a reference repository which can be cloned as the starting point for new Khronos specifications.
  - The reference repository may also be useful when creating other types of repositories, such as tutorials or sample code. The specification markup and build examples are less relevant to those use cases, but the repository structure and related files are relevant to any Khronos repository.
- Provide examples and documentation for using specification repository tooling and infrastructure including:
  - Boilerplate files needed by all Khronos repositories
  - Licenses and Contributor License Agreements
  - Asciidoctor markup and toolchain, of which this Specification is a an example.
  - Continuous Integration tooling for Github and Khronos Gitlab.

The published version of this Specification is available in the Sample Registry.

The repository used to generate this Specification is stored in the Sample-Docs repository, and should be referred to while reading this document.

The Sample-Docs repository also has a public issue tracker, and allows the submission of pull requests that improve the Specification.

2.1. Document Conventions

The Sample Specification is intended for use by Khronos Working Groups creating new API or language specifications. As such, it includes some material that is relevant only when defining a new specification. If you have forked the Sample-Docs repository as a starting point, you will need to examine all files in the repository; edit or remove files as appropriate for your project; and add new files for additional content specific to your specification.

Note

Informative notes, such as this paragraph, are used to provide commentary and context.

Khronos specifications are intended for use by both implementors, and by application developers seeking to make use of implementations. A specification forms a contract between these parties.

Specification text may address either implementors or developers. Typically the intended audience can be inferred from context, though some sections are defined to address only one of these parties.

Any requirements, prohibitions, recommendations or options defined by normative terminology are imposed only on the audience of that text.

2.1.1. Ratification

Ratification of a Khronos specification is a status conferred by vote of the Khronos Board of Promoters, bringing that core version or extension under the umbrella of the Khronos IP Policy.

The Sample Specification is not ratified since it contains no IP, just process documentation. Khronos specifications, including core versions and optional Khronos-defined extensions, should be ratified prior to publication. Ratification status is shown in the document preamble.

Note

Ratification status is primarily of interest to Khronos members developing their own implementations of a specification.

For developers, ratification does not necessarily mean that an extension is "better"; has a more stable API; or is more widely supported than alternative ways of achieving that functionality.

Interactions between ratified and non-ratified extensions are not themselves ratified.

2.1.2. Informative Language

Some language in a Specification is purely informative, intended to give background or suggestions to implementors or developers.

If an entire chapter or section contains only informative language, its title will be suffixed with "(Informative)".

All NOTEs are implicitly informative.

2.1.3. Normative Terminology

Within this specification, the key words **must**, **required**, **should**, **recommended**, **may**, and **optional** are to be interpreted as described in RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels (https://www.ietf.org/rfc/rfc2119.txt). The additional key word **optionally** is an alternate form of **optional**, for use where grammatically appropriate.

These normative words are emphasized in the Specification for clarity. In text addressing application developers, their use expresses requirements that apply to application behavior. In text addressing implementors, their use expresses requirements that apply to implementations.

Note

Please be conscious in use of normative terminology, and ensure that each such use is agreed by the Working Group controlling a specification. These terms place constraints on implementations that may have consequences for IP licensing and

> Ratification.

In text addressing application developers, the additional key words **can** and **cannot** are to be interpreted as describing the capabilities of an application, as follows:

can

This word means that the application is able to perform the action described.

cannot

This word means that the API and/or the execution environment provide no mechanism through which the application can express or accomplish the action described.

These key words are never used in text addressing implementors.

> *Note*
>
> There is an important distinction between **cannot** and **must not**, as used in the Specification. **Cannot** means something an application literally is unable to express or accomplish through the API or language. **Must not** means something that the application is capable of expressing, but the consequences of doing so are undefined: and potentially unrecoverable for the implementation (see [fundamentals-validusage]).

Unless otherwise noted in the section heading, all sections and appendices in this document are normative.

2.1.4. Technical Terminology

Specifications often make use of commonly used, domain-specific engineering terminology to identify and describe API constructs and and their attributes, states, and behaviors. For example, a “Graphics Pipeline” is a common term in the field of computer graphics, but has a specific meaning with respect to the Vulkan API. The Glossary defines the basic meanings of such terms in the context of the Specification. The Specification body provides fuller definitions of the terms and may elaborate, extend, or clarify the Glossary definitions. When a term defined in the Glossary is used in normative language within the Specification, the definitions within the Specification govern and supersede any meanings the terms may have in other technical contexts (i.e. outside the Specification).

2.1.5. Normative References

References to external documents are considered normative references if the Specification uses any of the normative terms defined in Normative Terminology to refer to them or their requirements, either as a whole or in part.

The following documents are referenced by normative sections of the Specification:

SPDX License List. https://spdx.dev/learn/handling-license-info/ .

REUSE Software. https://reuse.software/ .

Chapter 3. Fundamentals

This chapter introduces fundamental concepts for the Sample Specification.

Note

Refer to the fundamentals section of the Vulkan Specification for examples of concepts to describe in this chapter, such as ABIs, error codes, numeric representation, object models, threading behavior, and the validation model.

Such descriptions will be specific to the implementation being specified, the environments it is designed to execute in, and the language(s) applications may use to access the implementation.

Chapter 4. Licenses and Contributor License Agreements

All content in a Khronos specification **must** be licensed. This chapter covers:

- Choice of license
- How to include a license in a file
- How to document and verify repository licenses using REUSE
- Contributor License Agreements
- Licenses for ratified and non-ratified specification artifacts
- When to use the Confidential Repository Notice

This chapter incorporates and supersedes the “Khronos Open Source Repository Resources” document in the member Causeway area.

4.1. Choice of License

The Khronos default licenses for specifications are:

- Specification source (such as asciidoc markup, or Markdown) and related documentation files are under the Creative Commons Attribution 4.0 License to enable re-mixing.
- Source code such as build scripts and example code is under the Apache 2.0 License.
- When there is a compelling requirement for source code to be usable in a downstream LGPL project, the Working Group can approve a *dual license* on specific files, so that they can also be used under an LGPL-compatibile MIT License. Please consult with Neil Trevett before taking this step.
- Published specification artifacts - generated files such as HTML and PDF documents - are placed under the Khronos Specification Copyright License so long as they are Ratified Specifications. This is required so that such specifications bring Khronos licensees under the umbrella of the Khronos IP agreements. Ths Khronos Specification Copyright License should never be used on markup or source files, or on specification artifacts that include non-ratified content, such as vendor extensions.

Note

In the past, we have used a variety of other licenses. If your Working Group has a compelling reason to use a different, existing open source license, please consult with Neil Trevett before taking this step.

Please do not modify an existing open source license, or create a new one.

4.2. Including a License and Copyright Statement in a File

Every Khronos-authored file **must** have an associated copyright statement, including both the copyright (author and dates) and the *SPDX License Identifier* of the license applying to that file.

In most cases, the copyright statement is included as a comment at the earliest possible syntactically valid place in the file.

Note

In the past, we recommended including the complete license text in every file. When SPDX license identifiers and REUSE are used, only a single copy of each license text need appear, in the `LICENSES` directory.

Note

In the asciidoc markup of the following examples, the {empty} attribute is included to avoid confusing REUSE.

The actual copyright statement should be included as shown in the rendered output of this chapter, with the {empty} attribute removed.

Examples using the default licenses are:

Makefile or Shell Script with Apache 2.0 License

```
# Copyright 2023-2024 The Khronos Group Inc.
# SPDX-License-Identifier: Apache-2.0

...
```

Asciidoc Markup File with Creative Commons Attribution 4.0 License

```
// Copyright 2024 The Khronos Group Inc.
// SPDX-License-Identifier: CC-BY-4.0

...
```

XML Data File with Dual License

```
<?xml version="1.0" encoding="UTF-8"?>
<registry>
    <comment>
Copyright 2015-2024 The Khronos Group Inc.
SPDX-License-Identifier: Apache-2.0 OR MIT
    </comment>
...
```

4.2.1. Licenses for Unmodifiable or Externally Sourced Files

There are some cases where you may not be able to include an explicit copyright statement in the format described above:

1. *Unmodifiable* files need not have an explicit license statement included if their file format does not support it easily, or if the REUSE tool does not know how to extract the license from the file metadata. Examples include image and raw binary data files.
2. *Externally Sourced* files come from other open source projects. One example is a script that is used in our build toolchain with slight modifications. Such files **must** have an open source license that is compatible with the Khronos licenses for the purposes they are required for. However, the way in which these licenses are specified may be different, such as including the complete license text in their copyright header, or implcitly using the LICENSE file of their source repository to apply to every file in that repository. Even if their license allows it, Khronos should never modify that license other than by adding Khronos as a copyright owner along with the original author, under the same license.

To provide a license for these files, add them to the `.reuse/dep5` file described below for the REUSE tool.

4.3. Documenting Licenses

The full and unmodified license text for each license used in the repository **must** be included in the `LICENSES` directory. Every license uses **must** have a corresponding SPDX license identifier. The sample repository already includes the following licenses, which cover most use cases:

Table 1. Licenses in the Sample Specification Repository

SPDX License Identifier	Filename	Description
`Apache-2.0`	`LICENSES/Apache-2.0.txt`	Apache 2.0
`CC-BY-4.0`	`LICENSES/CC-BY-4.0.txt`	Creative Commons Attribution 4.0
`LicenseRef-KhronosSpecCopyright`	`LICENSES/LicenseRef-KhronosSpecCopyright.adoc`	Khronos Specification Copyright License
`MIT`	`LICENSES/MIT.txt`	MIT License

If you need license text and SPDX license identifiers for other open source licenses, obtain them from another Khronos repository already using those licenses, or from the SPDX License List on the SPDX website.

If you need to use a license not found on the SPDX website, please talk with Jon Leech and Neil Trevett first. This may happen if you are using a file from an open source project with a nonstandard license, which is uncommon. It may also happen if you are using a proprietary license **other** than the Khronos Specification Copyright License, which is already included in this repository. Khronos should make every reasonable effort to avoid using such proprietary licenses.

4.3.1. License Verification With REUSE

The REUSE tool is used in Continuous Integration to guarantee that all files in this repository are appropriately licensed.

REUSE verifies that every file in the repository has a license, and that every license is included in the `LICENSES` directory.

Please make certain that you include the REUSE `license-check` CI job in any new repositories.

Adding Licenses to `dep5`

Files that do not and cannot have explicit license information included, such as the Licenses for Unmodifiable or Externally Sourced Files discussed above, **must** have their licenses documented in the file `.reuse/dep5`.

Some examples of the `dep5` syntax for a file or group of files are:

```
Files: images/*.svg
Copyright: 2015-2024 The Khronos Group Inc.
License: CC-BY-4.0

Files: config/khronos.css
Copyright: 2013 Dan Allen
License: MIT
```

The `Files:` line contains the whitespace-separated names of one or more files in the repository. Wildcards are allowed.

The `Copyright` line contains the copyright statement for those files. If there is already a copyright statement in the file without a corresponding license, it should be copied verbatim.

The `License` line contains the SPDX license identifier of those files.

Multiple sections may be added to `dep5` for different licenses. Be careful not to include filenames repeatedly, or include filenames which already have explicit licenses. When a filename is repeated in `dep5` with incompatible licenses, behavior is unpredictable.

REUSE has other useful functionality, such as generating a license manifest, that Working Groups may find useful.

4.4. Contributor License Agreements

Every contribution to Khronos repositories **must** be appropriately licensed by its author. This is ensured by *Contributor License Agreements* and (for Khronos members who make contributions) by the Khronos member agreements and IP policies.

All Khronos members are strongly encouraged to execute the Khronos Corporate Open Source CLA to cover their contributions made under the Apache 2.0 license. Khronos members may edit their

designated employees in Schedule A by emailing Member Services.

Additionally, CLAs are integrated into our GitHub repositories such that all contributors **must** agree to the CLA before their pull requests can be merged. There are several possible CLAs that can be configured, depending on the nature of the repository:

- For most specification repositories, use the Specification Mixed Repository CLA, which covers contributions under both CC-BY 4.0 and Apache 2.0 licenses.
- For repositories containing *only* specification source documents under CC-BY 4.0 and *no* Apache 2.0 materials, use the Specification CLA.
- For repositories containing *only* source code under Apache 2.0 and *no* CC-BY 4.0 materials, use the Khronos Apache 2.0 CLA.
- For repositories containing *only* source code under the MIT license, use the MIT Click Through CLA.

Note

Repositories which start out using only one license are likely to eventually include material under other licenses, so the Mixed Repository CLA is the most forward-looking option. Choose the repository CLA with this in mind.

To integrate a CLA with a new repository, contact our administrator, James Riordon.

4.5. Licenses for Ratified and Non-Ratified Specification Artifacts

When creating a specification, asciidoc (or other markup) files are converted into HTML and/or PDF form more suitable for viewing in a web browser.

These HTML or PDF files are referred to here as “specification artifacts” or simply “artifacts”.

Note

Historically we have published both HTML and PDF artifacts for our specifications. PDF artifacts can become very large and have limited use cases but more readily support downloading for offline use, which some people appreciate.

Carefully consider which artifacts you wish to publish. It is difficult to withdraw a particular specification format you have published in the past, and easy not to publish that format in the first place.

Specification artifacts can be submitted to the Khronos Board of Promoters for ratification. Artifacts, whether ratified or not, can be published in the Khronos Registry for public consumption.

In either case, artifacts **must** be placed under a form of the Khronos Specification Copyright License. By setting appropriate asciidoc attributes, this license can be used for both **ratified** and **unratified** specifications. See the comments in the license markup for details. The Working Group and its Specification Editor are responsible for ensuring that the appropriate license is used for

artifacts they publish.

The Specification License is *not* an open source license, even though the markup files that went into creating artifacts are under such a license. The purposes of the Specification Copyright include:

- stating that the artifact was generated by Khronos
- describing its ratification status with respect to the Khronos IP Policy
- noting any trademarks that are used in the artifact.

Note

The Khronos Specification Copyright License found in the Sample-Docs repository is derived from the Khronos Ratified Specification Header and Khronos Specification Copyright License Header in the member Causeway area.

The Khronos Specification License has been reformatted for asciidoc markup, and uses asciidoc conditionals to control inclusion of different parts of the license as required. Using it in an actual specification repository requires further editing to include details and trademarks as appropriate for that repository, and to select appropriate parts of the license.

4.5.1. Licenses for Other Published Artifacts

For a document which is not a specification requiring ratification or with IP concerns, including the Sample Specification itself, you may use the CC-BY-4.0 license on the artifacts, as well as the input markup files.

Note

TODO Can we do away with the “Khronos Document Copyright Licenses” entirely in favor of CC-BY?

4.6. Confidential Repository Notice

The following notice **must** be placed in any repository covered by the Khronos NDA:

```
CONFIDENTIALITY NOTICE

Unless and until publicly released by Khronos, all material in this repository is
CONFIDENTIAL INFORMATION of Khronos. Confidentiality obligations supersede any
conflicting rights granted under any license terms associated with the material. Once
publicly released, the material is no longer confidential information, and the
applicable license terms fully govern use of the material.

The confidentiality obligation described in this notice does not limit any party's
right to use and disclose that party's own material at its own discretion.
```

Note

> The source of this Notice is in the member file area.

Chapter 5. Repository Structure

The Sample-Docs repository can be used as a framework for new Khronos specification repositories, by forking and then modifying its contents as needed.

This chapter discusses the content and structure of the repository.

5.1. Boilerplate Files

All repositories should include the following files, which are open source conventions and directed primarily at external contributors who access the repository on GitHub.

- `README.adoc` - the first file seen when viewing a repository on gitlab or GitHub. It should describe the purpose of the repository; its high-level structure and content (similarly to this section); and link to `BUILD.adoc`.
- `BUILD.adoc` - describes in detail how to build the specification in the repository.
- `COPYING.adoc` - describes all the licenses used in the repository, and includes an FAQ discussing how the repository content can be used.
- `CONTRIBUTING.adoc` - describes the sorts of contributions and licenses we can accept, and the Contributor License Agreement.
- `CODE_OF_CONDUCT.adoc` - Khronos Code of Conduct for participating in our open source projects.
- `LICENSES/` - contains the complete and unmodified text of all licenses used in the repository. See Documenting Licenses.
- `LICENSE.adoc` - briefly enumerates all the licenses used in the repository. This is unavoidably duplicative of `COPYING.adoc` and the `LICENSES` directory, since those files serve different purposes.
- `ChangeLog.adoc` - the repository change log. Typically used to summarizes changes between public updates of a specification, and intended as a more readable version of the git commit history.

5.2. Specification Files

Specification markup source in asciidoc format is by convention structured as follows:

- `Makefile` (or other build script) to build the specification artifacts. Normally the Makefile specifies targets (such as `html` or `pdf`) and options to invoke asciidoctor with for each target. The Sample-Docs repository's `Makefile` is considerably more complex than strictly required, since it is intended as an example for use with actual specifications that will themselves be much more complex than the Sample-Docs specification.
- `sample.adoc` (or `apiname.adoc` or other meaningful title) - top-level specification markup file, which specifies the document headers and includes separate files for chapters and appendices.
- `chapters/` - directory containing individual asciidoc files corresponding to individual chapters of the specification. Most specifications will need to use and modify the following files, which can be modified starting from the examples in the Sample-Docs repository:

    - `chapters/preamble.adoc` - includes the Khronos Specification Copyright License applied to the specification artifacts.
    - `chapters/introduction.adoc` - first chapter of the specification, defining what the specification is for and what it contains.
    - `chapters/fundamentals.adoc` - second chapter of the specification, defining fundamental concepts needed by the remainder of the document.
    - Additional chapters as relevant. For example, the Sample-Docs repository contains `chapters/licenses.adoc` and `chapters/repostructure.adoc` (which is the markup file for this chapter).
- `appendices/` - directory containing individual asciidoc files corresponding to individual appendices of the specification. This is purely a notational convenience. Appendices could be placed in the `chapters/` directory instead.
    - `appendices/lexicon.adoc` - used by most specifications to contain a glossary and other lexicon material.
    - `appendices/credits.adoc` - where individual contributors to a specification are acknowledged, including the Khronos staff who support our work.
- `images/` - directory containing SVG images included in the specification. The Sample-Docs repository contains a single image:
    - `images/Khronos_RGB_June18.svg` - Khronos logo included on the title page of the generated PDF artifact.

5.3. Configuration Files

The build toolchain for asciidoc-based specs used in the Sample-Docs repository includes a variety of build tools and configuration files as examples:

5.3.1. Helper Scripts

Over time repositories may accumulate many scripts used for infrastructure or small toolchain tasks. Most such scripts should be kept under a `scripts/` subdirectory in the repository. Sometimes scripts will be located elsewhere due to requirements of tools like CI or ease of use.

Scripts in the Sample-Docs repo include:

- Continuous Integration scripts used for specification builds:
    - `.gitlab-ci.yml` - Gitlab CI script
    - `.github/workflows/CI.yml` - Github CI script
- `runDocker` invokes the Khronos `asciidoctor-spec` Docker image.
- `scripts/makedocinfologo` is a shell script used to generate `config/docinfo-header.html`.

5.3.2. Helper and License Markup Files

- `config/attribs.adoc` defines asciidoc attributes for (mostly) math symbols, corresponding to

LaTeX math operators. For example, you can write `{pi} {plusmn} 1` for π ± 1 and similar simple math markup.

- config/copyright-ccby.adoc - CC-BY-4.0 copyright file for inclusion in published specification artifacts. Used in the Sample-Docs repository as this specification is neither ratified nor contains IP, but actual specifications will normally use:
- config/copyright-spec.adoc - Khronos Specification Copyright License file for inclusion in published specification artifacts. When using this file, please read it carefully to set asciidoc attributes controlling which sections are included, and to modify sections as appropriate for your specification.

5.3.3. Asciidoctor Extensions

Asciidoctor (the Ruby implementation of the asciidoc markup language) can be extended, and many extensions are available. Some of our specifications, such as OpenXR and Vulkan, rely on extensions. Others use only unextended asciidoc markup.

Two simple extension, added to the build in the Makefile, are:

- config/asciidoctor_mathematical.rb - allows using [latexmath] blocks inside asciidoc table cells.
- config/open_listing_block.rb - allows a listing block to masquerade as an open block, allowing open block nesting as in the following example.

Note

An example of open block nesting:

Open Block Nesting Example

This is an asciidoc open block. Normally open blocks cannot contain other open blocks.

The outer open block contains this nested open block, which is enabled by the open_listing_block extension. This syntax is not allowed by unextended asciidoctor.

5.3.4. Asciidoc Style Files

- config/docinfo-header.html is an HTML fragment injected into the header of output artifacts as specified by the document header attributes :docinfo: and :docinfodir: in sample.adoc. It includes a base64-encoded version of the Khronos logo that will appear near the head of the HTML document.
- config/khronos.css is a slightly modified version of the standard asciidoctor “colony” theme included in HTML artifacts. We encourage using this CSS for consistency with other Khronos specifications.

Note

config/docinfo-header.html can be regenerated if a different logo than the Khronos logo used in the Sample-Docs repository is desired, such as the logo of an actual

API being specified.

For example, the version in the Sample-Docs repository was generated by:

```
config/makedocinfologo images/Khronos_RGB_June18.svg "Khronos Logo" >
config/docinfo-header.html
```

The first argument is the name of an SVG file to be encoded, the second argument is alt-text for that image.

Chapter 6. Repository Management

This chapter is not prescriptive, but describes issues actual specification repositories may encounter, and some guidelines for dealing with them.

Exceptions to most of these guidelines exist, and can be discussed with Neil Trevett and the Khronos Board of Promoters.

6.1. Private and Public Repositories

Khronos tries to make as much of our material open source as possible. The constraints of the IP agreements mean that most Khronos-approved specifications are initially developed internally to Khronos, and only published after ratification.

To meet both needs, there are usually two specification repositories: a private version in Khronos' gitlab server, and a public version in the `KhronosGroup` organization on GitHub. This allows internal IP-sensitive development to take place while at the same time, public issues and pull requests can be created for already published versions of specifications.

6.2. Repository Creation and Naming

Specifications are kept in repositories called `Name-Docs` where `Name` is the name of the API or language being specified.

On gitlab, there is a naming hierarchy consisting of a gitlab *group* containing multiple *projects*. A group corresponds to a Khronos Working Group, and a project to a git repository with associated gitlab issue tracker, merge request manager, CI configuration, and so on. For example, the `Sample` group contains the `Sample-Docs` project, which includes the repository defining the `Sample-Docs` specification. It exists at URL

https://gitlab.khronos.org/Sample/Sample-Docs

On GitHub, the naming hierarchy consists of our GitHub *organization* (named `KhronosGroup`), containing multiple *repositories*. There is no deeper nesting possible, so we cannot use exactly the same path to repositories on GitHub and gitlab.

GitHub repository names correspond to gitlab project names, and both contain the actual git repository and associated tools. We avoid name collisions on GitHub by using the gitlab project name as the first part of the repository name, and using the repository name on GitHub as the project name on gitlab.

Thus, the Sample-Docs repository on GitHub is at URL

https://github.com/KhronosGroup/Sample-Docs

Note

There is no hard requirement that corresponding repositories in gitlab and GitHub be named the same. For example, the gitlab group and project name for the Vulkan

specification are both `vulkan`, while the GitHub organization and repository name are `KhronosGroup` and `Vulkan-Docs` respectively.

However, with the benefit of hindsight, it is better to use group, project and repository names that are as similar as possible on both platforms.

When your Working Group eventually creates additional repositories, they follow the same naming convention. For example, if we were to add a `Tutorial` repository, its names on GitHub and gitlab would be:

https://gitlab.khronos.org/Sample/Sample-Tutorial https://github.com/KhronosGroup/Sample-Tutorial

Creating and configuring new groups, projects, and repositories is beyond the scope of this document, and requires administrative privileges most Khronos members do not have. Please email Member Services for assistance. Once created, designated owners and administrators will be given privileges sufficient for day-to-day upkeep of repositories.

6.3. Synchronizing Repositories

It is important to keep GitHub and gitlab repositories synchronized. Most changes, including bugfixes new versions and extensions of specifications, are made on gitlab and propagated to GitHub after ratification. However, there is often feedback and changes proposed on GitHub by external developers using a specification.

It is easiest to keep repositories synchronized if they share the same commit history. Initially, this can be accomplished by creating a repository clone with multiple *upstreams* for gitlab and github, and pushing branch(es) in the local clone to both upstreams if they are to be published.

Note

For example, you might have a default (`main`) branch which corresponds to the published version of your specification and exists on both GitHub and gitlab, and a development (`devel`) branch which corresponds to the in-development version of your specification and exists only on gitlab.

The specification editor's responsibilities include pushing changes from gitlab to github when publishing specification updates developed internally, and pushing from github to gitlab when accepting public feedback. There are many ways of using git to accomplish this, such as cherry-picking git commits, git merges, or even simple-minded `diff` / `git apply`. If unsure how to proceed, you can always ask for advice on the Working Group Chairs mailing list.

It is easy to accidentally leak internal material when synchronizing with GitHub, so it's wise to develop strategies and automatic mechanisms to reduce this risk. For example, pushing a git tag from one repository to another can carry the entire git history of that tag with it.

6.4. Managing Community Interaction

Feedback and pull requests from external developers have been very valuable, both for the

improvements they represent and for the sense of engagement with and responsiveness by Khronos developers they can result in. Working Groups should have a process for monitoring and managing GitHub activity from external developers.

External contributions can also pose IP risks and consume considerable Working Group time on relatively minor matters. Working Group chairs should be aware of these risks and manage time spent on GitHub issues accordingly.

We do not want our public specification repositories to turn into technical support forums for programming problems, driver issues, and so on. We encourage external developers to use Khronos Discord or vendor-specific forums, where they are also more likely to get good and timely answers, for such questions.

6.5. Publication Strategies

Working Groups should agree on when to *publish* changes in their repositories. In one sense every commit pushed to a public GitHub repository is "published", but the sense we mean this is of generating and releasing specification artifacts corresponding to a particular point in the commit history. This can be done for a new core API version, a new language extension, or just for a collection of bugfixes and clarifications.

Working Groups will normally go through a process like:

- Agree on what set of changes to release and publish; why; and what to call the resulting artifacts.
- Synchronize gitlab and GitHub repositories, then merge all relevant internal changes to the default branch.
- Tag the resulting "release" appropriately. Semantic versioning names are often appropriate as git tag names.
- Generate specification artifacts corresponding to the release.
- Synchronize the default branch at the tag point back to GitHub.
- Publish the artifacts in the Khronos Registry.
- Make announcements if appropriate, on the Khronos website, Discord, or other channels.

6.5.1. The Khronos Registry

Most Khronos specifications are published in the Khronos Registry. This is a Khronos website containing multiple sections, one for each Working Group using it. Each section is backed by a corresponding GitHub repository which contains artifacts such as HTML and PDF files.

Note

For example, the OpenCL Working Group maintains their pnublished specification source in the OpenCL-Docs repository on GitHub.

Artifacts generated from the specification repository are published in the corresponding OpenCL-Registry repository.

The registry repositories are only a means of publication. GitHub cannot directly host HTML files - they can be downloaded, but not viewed at a www.github.com repository URL. The registry website automatically pulls updates pushed to the various registry repositories.

Note

We do not keep specification artifacts in the specification repositories as such generated files can very rapidly build up, consuming large amounts of space.

Note

There are other ways of publishing artifacts, including:

- GitHub Pages, which can publish artifacts generated in GitHub Actions CI.
- In some cases, specifications are so simple that they can be written entirely in GitHub-flavored asciidoc markup, which is a subset of asciidoc markup, and viewed directly on the specification repository. This is usually only suitable for short, single-file documents.

Other approaches have pros and cons compared to the registry. We think the registry is the best publishing approach for most specifications.

Chapter 7. Continuous Integration

The Sample-Docs repository is configured to perform CI on both Khronos' internal gitlab server and on GitHub using GitHub Actions.

GitHub supports other CI services such as Azure and Travis, but Actions is most closely integrated with GitHub and requires less configuration than other options.

Note

GitHub Actions CI costs are becoming significant for Khronos, in addition to some performance issues we've been experiencing. We are considering options and it is possible we will change the recommendation away from Actions, or start self-hosting Actions runners.

7.1. Repository CI Configuration

To execute CI jobs on GitHub, GitHub Actions must be enabled in the repository settings.

On gitlab, "shared runners" must be enabled in the repository CI settings.

If the repository administrator has trouble doing this, contact Member Services

7.2. Repository CI Scripts and Jobs

Gitlab and GitHub Actions CI scripts are both YAML format files, but the syntax of the YAML differs substantially. Both the Gitlab `.gitlab-ci.yml` and GitHub Actions `.github/workflows/CI.yml` define two jobs:

- The `license-check` job runs REUSE to validate repository licensing.
- The `spec-generate` job builds the specification artifacts in HTML and PDF forms.

These are just a starting point for CI in a specification repository.

7.3. Sample CI Runtime

Khronos publishes a Docker image containing a Debian Linux distribution with the entire toolchain preinstalled. This image is used to run the sample CI jobs, since it has all the necessary specification toolchain components preinstalled. We encourage using this image in new specification repositories as well.

For more discussion of the Docker image, see the Khronos-Provided Docker Image section of `BUILD.adoc`.

Note

We have had problems with Docker caching images and not updating to the latest versions pushed to Dockerhub. Trying to diagnose these problems is very difficult

> since CI executes in cloud resources we do not control and have little access to. To work around these problems, we specify the Docker image by its SHA rather than its name. When the image is updated on Dockerhub, the SHA encoded in the CI scripts and the `runDocker` script must also be updated.

7.4. Sample CI Outputs

On GitHub, click on `Actions` in the repository menu bar and look at the latest workflow corresponding to a branch of interest, or follow this link to workflows for the Sample-Docs repository.

On gitlab, click on `Build` and then `Jobs` in the repository left side panel menu and look at the latest job corresponding to a branch of interest, or follow this link to workflows for the Sample-Docs repository.

On both GitHub and gitlab, if you have a pull request / merge requests for your branch, a link to the latest job run for that PR / MR can be found on the corresponding PR / MR page.

Chapter 8. Asciidoc Markup for Authoring

Khronos has moved to asciidoc markup format for new specifications, and migrated many older specifications as well. Asciidoc is a powerful and expressive markup language with a large ecosystem of tooling and extensions built around it. We strongly recommend all new specifications be written in asciidoc.

There are many paths when authoring in asciidoc, from simple to more complex use cases. These are briefly discussed below.

Note

This document does not discuss asciidoc markup itself, only how it is used in the Khronos context. For new authors, see the asciidoc documentation for user guides and other material. For the most part, following the example of this specification will suffice.

Note

TODO: Need discussion of math markup / rendering and the KaTeX toolchain.

8.1. Simplest: GitHub-Flavored Asciidoc

GitHub supports a subset of asciidoc features, referred to as “GitHub-Flavored Asciidoc” or GFA. Gitlab supports similar functionality.

GFA is suitable for writing short, single-file markup such as the README.adoc in the Sample-Docs repository. We do not recommend it for longer documents, such as complete specifications.

Advantages of GFA are:

- Renders directly in the brower’s view of the file on github, without need for offline processing or hosting of artifacts.

Disadvantages of GFA are:

- Does not support important core asciidoc functionality, particularly include directives needed to construct multi-file projects.
- Does not support extensions and plugins, which limits math rendering, images, and any project-specific functionality your specification might want to use.
- Does not support user-defined CSS and styles.
- Cannot generate PDF, only live HTML views.

8.2. Intermediate: Unextended Asciidoctor

There are several implementations of processors for the asciidoc *markup language*. The most commonly used processor in Khronos repositories is Asciidoctor, which is a command-line tool written in Ruby. There are also JavaScript (Asciidoctor.js) and Java (AsciidoctorJ) implementations,

which are versions of Asciidoctor translated or virtualized to run with those languages.

Out of the box Asciidoctor is flexible enough for many specifications, with no need for custom extensions. This is the simplest way to use Asciidoctor, although your specification will still need to do configuration work. The Sample Specification is largely written in unextended Asciidoctor, although it does include a few small extensions as examples.

Advantages of unextended Asciidoctor are:

- Simplest offline processing tool.
- Supports the full asciidoc markup language.
- Can generate HTML, PDF (with the asciidoctor-pdf gem), EPUB, and other formats.
- Supports high-quality math rendering using KaTeX (recommended for performance reasons) or MathJax.
- Enables project-specific document styles (such as the `config/khronos.css` used in the Sample-Docs specification).
- Supports plugin syntax highlighters for markup of included code samples.
- Can always be extended later, if needed.

Disadvantages of unextended Asciidoctor are:

- Using extensions opens up additional input file format support, markup syntax, semantic tagging, and anything else you can imagine writing a markup extension for.

8.3. Most Complex: Asciidoctor with Extensions

Many features such as PDF generation are in fact Asciidoctor extensions, but this is largely irrelevant to their users. However, as a specification grows, it may be desirable to write *custom extensions* to modify the behavior of the underlying asciidoc processor. For example, the Vulkan and OpenXR specifications use a large number of custom extensions for many purposes, including but not limited to:

- Extended markup language features for convenience (such as the `config/open_listing_block.rb` example found in the Sample-Docs repository).
- Semantic markup of API names, which can vary their rendering depending on the context. For example, Vulkan and OpenXR both extract reference pages from their specifications, and rendering the same API name markup in a reference page and result in a different outbound link (to another reference page, instead of within the single-document specification).
- Special processing of markup. For example, Vulkan includes *valid usage (VU) statements* which are specially tagged and are extracted from the specification at built time, resulting in a JSON database of VUs which is then passed to downstream tools such as the Vulkan Validation Layers to assist in writing tests for the VUs.

Advantages of extended Asciidoctor are:

- Greatest flexibility.

- Can work around issues with Asciidoctor that are important to our documents, but the developers have not prioritized yet.
- Allows compact and semantically meaningful markup of important bits of a document that need special handling for the API or language being specified.

Disadvantages of extended Asciidoctor are:

- Requires learning a small amount Ruby, and about Asciidoctor internals. Their internal documentation has been improving, but you may still in some cases need to ask questions on the Asciidoctor support forum, or read the Asciidoctor code.
- Requires defining new markup constructs, and educating contributors to your project to use them.

8.4. On the Horizon: Antora Site Generation

Antora is a site generation tool for content written in asciidoctor. It is written in Javascript and uses the Asciidoctor.js implementation of Asciidoctor. Antora is written and supported by the same developers who are responsible for Asciidoc and Asciidoctor, and reasonably widely used already.

Antora produces a static HTML website from one or more asciidoctor “components” (which roughly correspond to specifications or other standalone document groups). It supports sitewide indexing, linking, presentation, and search of those components.

The Vulkan Documentation Website is Khronos' first step into Antora. Over time we expect more Khronos specifications will move into Antora format, but are still learning, and are not ready to recommend Antora for general use yet.

Advantages of Antora:

- Enables unified documentation websites combining specifications, tutorials, user guides, examples, and other material relevant to an ecosystem.
- Decent site-wide search using the Lunr extension.
- Detailed control of the site UI and styles is possible.

Disadvantages of Antora:

- Must generate and publish a complete static HTML website, not just a single HTML or PDF file.
- More complicated document setup, with significant restrictions around the structure of files.
- A different syntax for intra-document links that makes it difficult to write markup that will enable versions of the same specification to be built with either Antora or Asciidoctor.
- Using custom asciidoctor extensions written in Ruby requires either translating them to Javascript using a complex toolchain; or learning both Javascript and Ruby, and the somewhat different ways the Asciidoctor internal APIs are used from them.

The different link markup conventions used by Antora were a significant barrier to Vulkan, which cannot easily, or soon abandon the Asciidoctor-generated specifications we publish in the Registry. In an attempt to alleviate this issue, we wrote a series of scripts which transform Vulkan

specification markup into an Antora-friendly form. When combined with Asciidoctor.js extensions which interpret the semantic markup of the Vulkan specification appropriately for Antora, this has enabled us to support both formats.

Unfortunately, the Vulkan approach is specific to the rigid style guidelines and the extended semantic markup conventions of that specification, and the scripts would require considerable adaptation to be used in other specifications.

We hope to have better answers and strategies for Antora migration in the relatively near future.

Appendix A: Lexicon

This appendix defines terms and abbreviations used in the Specification.

Glossary

The terms defined in this section are used consistently throughout the Specification and may be used with or without capitalization, although proper nouns such as “Khronos” should always be capitalized.

Khronos Specification

A specification document authored and published by the Khronos Group, defining an API, language, or other standard.

Sample Specification

This document. A reference and guide to writing an actual Khronos specification, but not itself a Khronos specification.

SPDX

The Software Package Data Exchange, which defines conventions and terminology used when applying licenses to this repository.

Common Abbreviations

The abbreviations and acronyms defined in this section are sometimes used in the Specification and the API where they are considered clear and commonplace.

License ID: A SPDX license identifier, used to refer to one of the licenses used in this repository.

Appendix B: Credits (Informative)

The Sample Specification and corresponding Sample-Docs repository were created by Jon Leech based on the “Khronos Open Source Repository Resources” document maintained by Neil Trevett, and on Khronos' collective experience with open source licenses, repository creation, and specification authoring.

Substantial contributions were made by:

- Jon Leech, Vulkan and OpenGL Spec Editor
- James Riordon, Khronos Webmaster
- Emily Stearns, Khronos Director
- Neil Trevett, Khronos President