Certora Documents Infrastructure

Release 0.1.0

Certora, Inc

Mar 10, 2024

Contents

Easily create Sphinx docs for Certora.

1 Features

1.1 Include CVL code

Easily include CVL code from spec files.

MyST (.md)

```
```{cvlinclude} /../../code/voting/Voting_solution.spec
:cvlobject: onlyLegalVotedChanges sumResultsEqualsTotalVotes numVoted
:spacing: 1
```
```

reStructuredText (.rst)

```
.. cvlinclude:: /../../code/voting/Voting_solution.spec
  :cvlobject: onlyLegalVotedChanges sumResultsEqualsTotalVotes numVoted
  :spacing: 1
```

Rendered as:

```
/// @title No illegal changes to `_hasVoted`
invariant onlyLegalVotedChanges()
    !illegalStore;

/// @title Sum of voter in favor and against equals total number of voted
invariant sumResultsEqualsTotalVotes()
    votesInFavor() + votesAgainst() == to_mathint(totalVotes());

/// @title Count the number of times `_hasVoted` been written to
ghost mathint numVoted {
    init_state axiom numVoted == 0;
}
```

1.2 Link to Github

MyST (.md)

```
For example {clink}`Voting solution spec </voting/Voting_solution.spec>`.
```

reStructuredText (.rst)

```
For example :clink:`Voting solution spec </voting/Voting_solution.spec>`.
```

Rendered as:

For example Voting solution spec.

1.3 CVL syntax highlighting

```
/**
 * # Simple voting contract complete spec
 *
 * To use gambit, run from the tutorials-code root folder the following command:
 * `certoraMutate --prover_conf solutions/lesson4_invariants/simple_voting/Voting_
↪solution.conf --mutation_conf solutions/lesson4_invariants/simple_voting/mutate.
↪json`
 */
```

(continues on next page)

(continued from previous page)

```
methods
{
    function votesInFavor() external returns (uint256) envfree;
    function votesAgainst() external returns (uint256) envfree;
    function totalVotes() external returns (uint256) envfree;
    function hasVoted(address) external returns (bool) envfree;
}
```

1.4 Create pdf versions

For example `Certora documents infrastructure`.

---

2 Contents

2.1 Quickstart

Installation

1. Clone the docsinfra repository
2. Install the Python package by running the following from the cloned repository's folder:

```
pip3 install -e .
```

Warning: It is always recommended to use a Python virtual environment, such as venv, when installing a Python package.

Initialization

Use the `certora-doc-quickstart` script to quickly initialize a document.

```
certora-doc-quickstart <PROJECT_DIR> --project <PROJECT_NAME>
```

This will create two folders inside `PROJECT_DIR`:

1. `source` - for source files (i.e. reStructuredText)
2. `build` - for the resulting html (or latex) files

See *Quickstart script* for more information.

Build html

To build html run:

```
sphinx-build -b html <path to source> <path to build>/html
```

View the resulting web pages in `<path to build>/html/index.html` on your web browser.

To build a pdf, see *Generating PDF output*.

Example

Suppose there is a project folder at `root`, with spec files under `root/code`, as shown below.

Listing 1: Project initial folder structure

```
root (top project dir)
  └── code
       └── ... spec and conf files
```

To quickly start a document with project name "Certora project", and documentation and build files under `root/docs`, run the following command from `root`:

```
certora-doc-quickstart docs -p "Certora project" --code ../../code
```

Note: The reason we give the code path as `../../code` is that it needs to be relative to `root/docs/source/` folder.

Listing 2: Project folder structure after quickstart

```
root
  ├── docs
  │    ├── source
  │    │    ├── index.rst (root documentation file)
  │    │    └── conf.py (configuration file)
  │    └── build
  │         └── html (created by sphinx-build command)
  │              └── index.html (root html file)
  └── code
       └── ... spec and conf files
```

Build the html file by running from `root`:

```
sphinx-build -b html docs/source/ docs/build/html
```

View `root/docs/build/html/html.index` on your browser.

2.2 Generating PDF output

Important: To generate pdf output you will need a LaTex installation with the `pdflatex` engine.

Note: Although it is possible to build Sphinx documents directly into pdf, here we only describe building it via LaTex first, since the output is better.

Basic use

Running the command below will:

1. create a LaTex file inside `<output-dir>/latex`
2. run `pdflatex` on the LaTex file twice (to get correct references)

```
sphinx-build -M latexpdf <source-dir> <output-dir>
```

Options

To modify the resulting LaTex document, one can add various options to the configuration file `conf.py`. These options are detailed in Options for LaTex output, and also in LaTex Customiztion.

Another way to control the options is modifying them in the `sphinx-build` command using the `-D` option. See sphinx-build for more details.

Example

Adding the following lines to `conf.py` will change the paper size and add a logo.

```
latex_elements["papersize"] = "a4paper"
latex_logo = "_static/logo.png"  # Relative to the source dir, must be png
```

Alternatively, the following command will do the same:

```
sphinx-build -M latexpdf docs/source docs/build/ -D latex_elements.papersize=a4paper \
  -D latex_logo=_static/logo.png
```

Note: There are several logo `.png` images you can use in the `_static` folder.

Important options

- `latex_elements.papersize`: `a4paper` or `letterpaper`
- `latex_elements.pointsize`: `10pt`, `11pt` or `12pt`
- `latex_logo`: path to logo `.png` file, relative to source dir
- `latex_toplevel_sectioning`: `part`, `chapter` or `section`
- `latex_theme`: `manual` (larger document) or `howto` (smaller document)

Preferred format

Use the following options to create a smaller document with the Certora logo.

```
# Create full pdf
sphinx-build -M latexpdf docs/source/ docs/build/fullpdf \
  -D latex_elements.papersize=a4paper \
  -D latex_logo=_static/logo.png \
  -D latex_toplevel_sectioning=section \
  -D latex_theme=howto \
  -t is_dev_build
```

Here is the output `Certora documents infrastructure`.

Building partial document

To create a pdf of only a part of the documentation:

1. Change the source dir to the desired folder with `index.rst` file, e.g. `docs/source/showcase`
2. Provide the path to the folder containing the relevant `conf.py` file using the `-c` option, e.g. to use the standard config file: `-c docs/source/`
3. Update the *code_path_override* variable to be relative to the new source directory, e.g. `-D code_path=/.
   ./../../code/`
4. Optionally, modify the title and html title, e.g. `-D project="Sphinx showcase"` and `-D html_title="Sphinx showcase"`

For example, to create a pdf only from the *Sphinx tutorial and showcase* chapter:

```
# Create pdf of one part
sphinx-build -M latexpdf docs/source/showcase docs/build/partpdf \
  -c docs/source/ \
  -D code_path_override=../../../code \
  -D project="Sphinx showcase" \
  -D html_title="Sphinx showcase" \
  -D latex_elements.papersize=a4paper \
  -D latex_logo=_static/logo.png \
  -D latex_toplevel_sectioning=section \
  -D latex_theme=howto \
  -t is_dev_build
```

Here is the output `Sphinx showcase`.

2.3 Sphinx tutorial and showcase

This chapter describes the most useful Sphinx directives and roles.

If you're markdown files (`.md`) – these use the MyST package, and the MyST examples are what you need. Those using reStructuredText filed (`.rst`) should refer to the reStructuredText examples.

For additional information see:

- reStructuredText Primer
- Sphinx Directives
- MyST

Standard markup

Basic inline markup

Fonts

MyST (.md)

```
* We can use _italic_ and *bold*.
* We can even have a `mono font`.
```

reStructuredText (.rst)

```
* We can use *italic* and **bold**.
* We can even have a ``mono font``.
```

Rendered as:

- We can use *italic* and **bold**.
- We can even have a `mono font`.

Headings

MyST (.md)

Listing 3: Headings conventions

```
# Top level

## Second level

### Third level
```

reStructuredText (.rst)

For a full list and explanation, see: reStructuredText Sections. Note there are no levels assigned to particular heading characters. Sphinx deduces the levels in each `.rst` file.

Listing 4: Headings conventions

```
Section heading
===============

Sub-section
-----------

Sub sub-section
^^^^^^^^^^^^^^^

Even lower level
"""""""""""""""""
```

Horizontal rule

Use four dashes `----` (with empty lines above and below) to get a horizontal rule like the one below.

Lists

Bullet lists

```
* Bullet item
* Can contain nested lists

  * Like this
  * And this
```

Rendered as:

- Bullet item
- Can contain nested lists
  - Like this
  - And this

Numbered lists

MyST (.md)

```
1. Numbered list
2. Second item
```

reStructuredText (.rst)

```
#. Numbered list
#. Second item
```

Rendered as:

1. Numbered list
2. Second item

Definition list

MyST (.md)

```
Some term
: Followed by definition of the term, which must be indented.

  The definition can even consist of multiple paragraphs.

Second term
: Description of the second term.
```

reStructuredText (.rst)

```
Some term
   Followed by definition of the term, which must be indented.

   The definition can even consist of multiple paragraphs.

Second term
   Description of the second term.
```

Rendered as:

Some term
: Followed by definition of the term, which must be indented.

  The definition can even consist of multiple paragraphs.

Second term
: Description of the second term.

Links

External links

MyST (.md)

```
* Simple external link: [Certora](https://www.certora.com/).
* Using predefined link: [Rick Astley][RickRolled] (defined below)

[RickRolled]: https://www.youtube.com/watch?v=dQw4w9WgXcQ
```

reStructuredText (.rst)

```
* Simple external link: `Certora <https://www.certora.com/>`_.
* Using predefined link: `Rick Astley`_ (defined below)

.. _Rick Astley: https://www.youtube.com/watch?v=dQw4w9WgXcQ
```

Rendered as:

- Simple external link: Certora.
- Using predefined link: Rick Astley (defined below)

Embedding a Youtube video

MyST (.md)

````
```{youtube} VGSsPIsbb6U
:align: center
```
````

reStructuredText (.rst)

```
.. youtube:: VGSsPIsbb6U
   :align: center
```

Rendered as:

https://youtu.be/VGSsPIsbb6U

Internal links

Link anywhere inside the documentation.

MyST (.md)

See MyST Cross-referencing.

reStructuredText (.rst)

```
.. _my-reference-label:

Cross-reference inside documentation
""""""""""""""""""""""""""""""""""""

Set up a label ``.. _my-reference-label`` as shown above.
Note underscore prefix in the label name.

To reference use the ``:ref:`` directive like so: :ref:`my-reference-label`.
```

Link to code file on Github

Link to a code file using the `:clink:` role. The link will be either to GitHub or to local file, depending on the value of `link_to_github` variable in the `source/conf.py` file.

Absolute paths will be considered as relative to the *absolute code path* – see *code_path_override*. For complete documentation, see *Codelink extension*.

The basic syntax is:

MyST (.md)

Listing 5: Syntax

```
{clink}`Optional name <relative-path-to-code-file>`
{clink}`Optional name <absolute path relative to absolute code path>`
```

reStructuredText (.rst)

Listing 6: Syntax

```
:clink:`Optional name <relative-path-to-code-file>`
:clink:`Optional name <absolute path relative to absolute code path>`
```

For example:

MyST (.md)

```
* Reference to a folder: {clink}`Voting folder </voting>`
* Reference to a file: {clink}`Voting_solution.spec </voting/Voting_solution.spec>`
* Reference without text: {clink}`/voting/Voting_solution.spec`
```

reStructuredText (.rst)

```
* Reference to a folder: :clink:`Voting folder </voting>`
* Reference to a file: :clink:`Voting_solution.spec </voting/Voting_solution.spec>`
* Reference without text: :clink:`/voting/Voting_solution.spec`
```

Rendered as:

- Reference to a folder: Voting folder
- Reference to a file: Voting_solution.spec
- Reference without text: /voting/Voting_solution.spec

Code blocks

Best practice

It is best to include a code-block from a spec or Solidity file that is part of a regtest. This will ensure that you will be alerted if there are any breaking changes. Use the directives described in *From external file*.

Including source code for CVL elements using the `includecvl` directive (see *Including CVL elements* below) has the added benefit that it is protected against changes to the code file itself. Added or removed lines will not affect it.

In-place code

Code-block

You can insert a *CVL* code block in-place, using the `code-block` directive, as shown below. The same directive can be used for other languages, such as Solidity.

MyST (.md)

````
```{code-block} cvl

   methods {
       function balanceOf(address) external returns (uint256) envfree;
   }

   rule testBalance(address user) {
       assert balanceOf(user) > 0;
   }
```
````

reStructuredText (.rst)

```
.. code-block:: cvl

   methods {
       function balanceOf(address) external returns (uint256) envfree;
   }

   rule testBalance(address user) {
       assert balanceOf(user) > 0;
   }
```

Rendered as:

```
methods {
    function balanceOf(address) external returns (uint256) envfree;
}

rule testBalance(address user) {
    assert balanceOf(user) > 0;
}
```

Additional features, such as line numbers and emphasized lines are demonstrated below. You can find all the options available at: code-block directive.

MyST (.md)

````
```{code-block} cvl
:linenos:
:lineno-start: 7
:emphasize-lines: 10,17
:caption: CVL2 code example

methods
{
    function DataWarehouse.getRegisteredSlot(
        bytes32 blockHash,
        address account,
        bytes32 slot
    ) external returns (uint256) => _getRegisteredSlot(blockHash, account, slot);
}
```
````

(continues on next page)

(continued from previous page)

```
ghost mapping(address => uint256) _exchangeRateSlotValue;

function _getRegisteredSlot(
    bytes32 blockHash,
    address account,
    bytes32 slot
) returns uint256 {
    return _exchangeRateSlotValue[account];
}
```

reStructuredText (.rst)

```
.. code-block:: cvl
   :linenos:
   :lineno-start: 7
   :emphasize-lines: 10,17
   :caption: CVL2 code example

   methods
   {
       function DataWarehouse.getRegisteredSlot(
           bytes32 blockHash,
           address account,
           bytes32 slot
       ) external returns (uint256) => _getRegisteredSlot(blockHash, account, slot);
   }

   ghost mapping(address => uint256) _exchangeRateSlotValue;

   function _getRegisteredSlot(
       bytes32 blockHash,
       address account,
       bytes32 slot
   ) returns uint256 {
       return _exchangeRateSlotValue[account];
   }
```

Rendered as:

Listing 7: CVL2 code example

```
7  methods
8  {
9      function DataWarehouse.getRegisteredSlot(
10         bytes32 blockHash,
11         address account,
12         bytes32 slot
13     ) external returns (uint256) => _getRegisteredSlot(blockHash, account, slot);
14 }
15
16 ghost mapping(address => uint256) _exchangeRateSlotValue;
17
18 function _getRegisteredSlot(
19     bytes32 blockHash,
```

(continues on next page)

(continued from previous page)

```
    address account,
    bytes32 slot
) returns uint256 {
    return _exchangeRateSlotValue[account];
}
```

Inline CVL and solidity

MyST (.md)

You can add inline *CVL* code using the `:cvl:` role, and inline Solidity using the `:solidity:` role. To do so you must first define these roles at the top of your `.md` file, like so:

````
```{role} cvl(code)
:language: cvl
```

```{role} solidity(code)
:language: solidity
```
````

Now we can use them, as in the following example:

```
Type casting between integers in *CVL* has two different forms,
{cvl}`assert_uint256` and {cvl}`require_uint256`.
In the {solidity}`constructor(uin256 x)` ...
```

reStructuredText (.rst)

You can add inline *CVL* code using the `:cvl:` role, and inline Solidity using the `:solidity:` role. These roles are defined in the `conf.py` file. For example, the following paragraph:

```
Type casting between integers in *CVL* has two different forms,
:cvl:`assert_uint256` and :cvl:`require_uint256`. In the
:solidity:`constructor(uin256 x)` ...
```

Rendered as:

Type casting between integers in *CVL* has two different forms, **assert_uint256** and **require_uint256**. In the **constructor**(uin256 x) ...

From external file

Including CVL elements

Use the `cvlinclude` directive to include CVL elements *by name*. This is the preferred way to include rules, invariants, ghosts and the methods block. Complete documentation is available at *Include CVL extension*.

Example

MyST (.md)

```
```{cvlinclude} ../../../../code/voting/Voting_solution.spec
:cvlobject: numVoted onlyLegalVotedChanges sumResultsEqualsTotalVotes
:caption: Voting rules
```

reStructuredText (.rst)

```
.. cvlinclude:: ../../../../code/voting/Voting_solution.spec
   :cvlobject: numVoted onlyLegalVotedChanges sumResultsEqualsTotalVotes
   :caption: Voting rules
```

Rendered as:

Listing 8: Voting rules

```
/// @title Count the number of times `_hasVoted` been written to
ghost mathint numVoted {
    init_state axiom numVoted == 0;
}

/// @title No illegal changes to `_hasVoted`
invariant onlyLegalVotedChanges()
    !illegalStore;

/// @title Sum of voter in favor and against equals total number of voted
invariant sumResultsEqualsTotalVotes()
    votesInFavor() + votesAgainst() == to_mathint(totalVotes());
```

- If the path to the spec file is absolute, it is considered as relative to the `/source/` directory.
- The `:cvlobject:` option accepts names of CVL elements (rule, invariant and ghosts). To include the `methods` block, add `methods` to these names. The elements will be shown in the order they are given.

Note: Hooks are not supported (since they are not supported by the `CVLDoc` package). Use `literalinclude` below.

Including any code

Use the `literalinclude` directive to include code from an external file. As above, providing an absolute path is taken as relative to the `/source/` directory. For all possible options of `literalinclude`, see the literalinclude directive.

Important: An alternative to using line numbers when including code are the `:start-after:`, `:start-at:`, `:end-before:`, and `:end-at:` options. These accept string, which they match to find the desired lines.

For example:
```

MyST (.md)

```
```{literalinclude} ../../../../code/voting/Voting_solution.spec
:language: solidity
:lines: 4-
:emphasize-lines: 4-6
```
```

reStructuredText (.rst)

```
.. literalinclude:: ../../../../code/voting/Voting.sol
   :language: solidity
   :lines: 4-
   :emphasize-lines: 4-6
```

Rendered as:

```
contract Voting {

  mapping(address => bool) internal _hasVoted;

  uint256 public votesInFavor;
  uint256 public votesAgainst;
  uint256 public totalVotes;

  function vote(bool isInFavor) public {
    require(!_hasVoted[msg.sender]);
    _hasVoted[msg.sender] = true;

    totalVotes += 1;
    if (isInFavor) {
      votesInFavor += 1;
    } else {
      votesAgainst += 1;
    }
  }

  function hasVoted(address voter) public view returns (bool) {
    return _hasVoted[voter];
  }
}
```

Indexing and glossary

Indexing

To add terms to the genindex, place an appropriate `index` directive before the part you wish to index. See Sphinx - index directive for a comprehensive description of this directive, here are some simple examples.

Simple indexing

The following will create three index entries.

MyST (.md)

Todo: Test this one!

```
```{index} municipality, town, city
```
```

reStructuredText (.rst)

```
.. index:: municipality, town, city
```

Adding single values

MyST (.md)

```
```{eval-rst}
.. index::
   single: propositional logic
   single: logic; propositional
```
```

reStructuredText (.rst)

```
.. index::
   single: propositional logic
   single: logic; propositional
```

This will create two index entries, the first as “propositional logic” and the second will be a sub-index under “logic”.

Adding reference labels to indexes

Use the `:name:` option for adding a label that can be used with `:ref:`. For example:

MyST (.md)

```
```{eval-rst}
.. index::
   :name: intro_to_formal
```

Introduction to formal verification

See {ref}`intro_to_formal` ...
```

reStructuredText (.rst)

```
.. index::
   :name: intro_to_formal

Introduction to formal verification
-----------------------------------

See :ref:`intro_to_formal` ...
```

Inline indexing

You can add index entries inline. Here is an example from Sphinx - index directive:

MyST (.md)

```
This is a normal MyST {index}`paragraph` that contains several
{index}`index entries <pair: index; entry>`.
```

reStructuredText (.rst)

```
This is a normal reST :index:`paragraph` that contains several
:index:`index entries <pair: index; entry>`.
```

Glossary

For complete documentation on the `glossary` directive see Sphinx - Glossary.

Creating a glossary

Create a glossary using the `glossary` directive, followed by a *Definition list* of the desired terms. A term can have several names, as shown in the following example.

MyST (.md)

````
{.glossary}
CVL
: The Certora Verification Language, used for writing specs for Solidity contracts.

```{glossary}
Prover
Certora Prover
: The tool used for verifying specs written in {term}`CVL`.
```
````

reStructuredText (.rst)

```
.. glossary::

   CVL
      The Certora Verification Language, used for writing specs for Solidity␣
↪contracts.

   Prover
   Certora Prover
      The tool used for verifying specs written in :term:`CVL`.
```

Rendered as:

CVL
: The Certora Verification Language, used for writing specs for Solidity contracts.

Prover
Certora Prover
: The tool used for verifying specs written in *CVL*.

Referencing a glossary term

Use the `term` role to refer to a glossary term, for example:

MyST (.md)

```
* Simple reference such as {term}`CVL`
* Showing alternative text like {term}`The Prover <Prover>`
```

reStructuredText (.rst)

```
* Simple reference such as :term:`CVL`
* Showing alternative text like :term:`The Prover <Prover>`
```

Rendered as:

- Simple reference such as *CVL*
- Showing alternative text like *The Prover*

Comments and TODOs

RestructuredText comments

```
.. This is a comment in RestructuredText, the entire paragraph will be ignored
   by sphinx. Just note the indentation.
```

Development build

We can have content that is visible only in *dev-build* mode. To enable dev-build mode, add `-t is_dev_build` to the `sphinx-build` command (see *Build html* and *Generating PDF output*). For example:

```
sphinx-build -b html docs/source/ docs/build/html -t is_dev_build
```

Note: In dev-build the html title (on the side bar) will have "- Development" added to it. This behavior can be modified in the `/source/conf.py` file.

Contents for dev-build only

To produce contents that will appear only in dev-build, use the `.. only` directive, like this:

MyST (.md)

````
```{only} is_dev_build

The following will only be included in dev builds.
```
````

reStructuredText (.rst)

```
.. only:: is_dev_build

   The following will only be included in dev builds.
```

Rendered as:

The following will only be included in dev builds.

TODOs

TODO comments will only appear in dev-build. To add a TODO comment:

MyST (.md)

````
```{todo}
This is an example of a TODO comment, it can also have several paragraphs.
```
````

reStructuredText (.rst)

```
.. todo:: This is an example of a TODO comment, it can also have several paragraphs.
```

Rendered as:

Todo: This is an example of a TODO comment, it can also have several paragraphs.

To create a list containing all the TODO comments:

MyST (.md)

```
```{todolist}
```
```

reStructuredText (.rst)

```
.. todolist::
```

Admonitions

Admonitions are used for warnings, info and so on. Here is a collection of admonitions examples.

MyST (.md)

```
```{note}

For providing notes and information to the user.

The admonition can contain several paragraphs and also other elements, like:

* Lists
* Math
```

```{attention}
Pay attention.
```

```{important}
For marking very important things.
```

```{tip}
Tips for the reader.
```

```{hint}
Provide hints.
```
```

(continues on next page)

(continued from previous page)

````
```{warning}
Warn about dangerous things.
```

```{seealso}
For providing more references.
```

```{admonition} General admonition - any title you want
The freedom to admonish.
```
````

reStructuredText (.rst)

```
.. note::

   For providing notes and information to the user.

   The admonition can contain several paragraphs and also other elements, like:

   * Lists
   * Math

.. attention::

   Pay attention,

.. important::

   For marking very important things.

.. tip::

   Tips for the reader.

.. hint::

   Provide hints.

.. warning::

   Warn about dangerous things.

.. seealso::

   For providing more references.

.. admonition:: General admonition - any title you want

   The freedom to admonish.
```

Rendered as:

Note: For providing notes and information to the user.

The admonition can contain several paragraphs and also other elements, like:

- Lists
- Math

Attention: Pay attention,

Important: For marking very important things.

Tip: Tips for the reader.

Hint: Provide hints.

Warning: Warn about dangerous things.

See also:

For providing more references.

General admonition - any title you want

The freedom to admonish.

Panels

The panels use the sphinx-design extension. Follow the link for more details.

Single card

MyST (.md)

```
```{card} Card Title

Content of the card. See
`sphinx-design <https://sphinx-design.readthedocs.io/en/rtd-theme/index.html>`_
for more details.
```
```

reStructuredText (.rst)

```
.. card:: Card Title

   Content of the card. See
   `sphinx-design <https://sphinx-design.readthedocs.io/en/rtd-theme/index.html>`_
   for more details.
```

Rendered as:

Card Title Content of the card. See sphinx-design for more details.

Grid with two cards

MyST (.md)

````
````{grid} 2

```{grid-item-card}  Title 1

Left card
```

```{grid-item-card}  Title 2

Right card
```
````
````

reStructuredText (.rst)

```
.. grid:: 2

    .. grid-item-card::  Title 1

        Left card

    .. grid-item-card::  Title 2

        Right card
```

Rendered as:

Title 1 Left card

Title 2 Right card

Placing code side by side

Note the limited width of the columns!

Spec

Listing 9: Invariant

```
invariant totalIsBiggest(address user)
    balanceOf(user) <= totalBalance();
```

Solidity

Listing 10: Solidity

```
function balanceOf(
  address user
) external view returns (bool) {
  return _balances[user];
}
```

Drop-down

Drop-down content is useful for providing hidden hints. Here is a simple drop-down:

MyST (.md)

````
```{dropdown} Dropdown title
:animate: fade-in-slide-down

Dropdown content, for example an important hint.

See `sphinx-design - dropdowns
<https://sphinx-design.readthedocs.io/en/rtd-theme/dropdowns.html>`_ for more options.
```
````

reStructuredText (.rst)

```
.. dropdown:: Dropdown title
   :animate: fade-in-slide-down

    Dropdown content, for example an important hint.

    See `sphinx-design - dropdowns
    <https://sphinx-design.readthedocs.io/en/rtd-theme/dropdowns.html>`_ for more␣
↪options.
```

Rendered as:

Dropdown title

Dropdown content, for example an important hint.

See sphinx-design - dropdowns for more options.

Using Latex

In-line math

For inline math use the `:math:` role. For example:

MyST (.md)

```
Let {math}`\mathcal{C}` be the category of groups and {math}`f: G \to H` be a
morphism in {math}`\mathcal{C}`.
```

reStructuredText (.rst)

```
Let :math:`\mathcal{C}` be the category of groups and :math:`f: G \to H` be a
morphism in :math:`\mathcal{C}`.
```

Rendered as:

Let $\mathcal{C}$ be the category of groups and $f : G \to H$ be a morphism in $\mathcal{C}$.

Centered math

Use the `math` directive, as shown below. See Directives - math for additional options and examples.

MyST (.md)

````
```{math}
(a + b)^2  &=  (a + b)(a + b) \\
           &=  a^2 + 2ab + b^2
```
````

reStructuredText (.rst)

```
.. math::

   (a + b)^2  &=  (a + b)(a + b) \\
              &=  a^2 + 2ab + b^2
```

Rendered as:

$$
\begin{aligned}(a + b)^2 &= (a + b)(a + b) \\
           &= a^2 + 2ab + b^2\end{aligned}
$$

Advanced use

Here is an example of showing a conditional function.

MyST (.md)

Listing 11: Conditional function in Latex

````
```{math}
:nowrap:

\begin{equation}
f(x) =
\begin{cases}
     0  & \text{if } x \leq 0 \\
     x^2 & \text{otherwise}
\end{cases}
\end{equation}
```
````

reStructuredText (.rst)

Listing 12: Conditional function in Latex

```
.. math::
   :nowrap:

   \begin{equation}
   f(x) =
   \begin{cases}
        0  & \text{if } x \leq 0 \\
        x^2 & \text{otherwise}
   \end{cases}
   \end{equation}
```

Rendered as:

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x^2 & \text{otherwise} \end{cases} \tag{1}$$

Note: When using the `.. math::` directive, Sphinx will wrap the latex code inside the Latex `split` environment before rendering it. Using the `:nowrap:` option disables this behavior.

For example, the code from *Centered math* is rendered as the following Latex code:

```
\begin{split}
    (a + b)^2  &=  (a + b)(a + b) \\
               &=  a^2 + 2ab + b^2
\end{split}
```

Miscellaneous

Attention: This section is currently only available for reStructuredText. See MyST Roles and Directives for how to use these directives in MyST.

Tables

There are several ways to add tables in reStructuredText, there are described in

- reStructuredText Primer - Tables
- CSV Tables
- List Tables

Here is an example of a *list table*.

```
.. list-table:: Table title
   :header-rows: 1

   * - Column Header
     - 2nd Column Header
     - 3rd Column Header

   * - Row 1 Column 1 item
     - Row 1 Column 2 item
     - An item

   * - An item
     - Row 2 Column 2 item
     - Row 2 Column 3 item
```

Rendered as:

Table 1: Table title

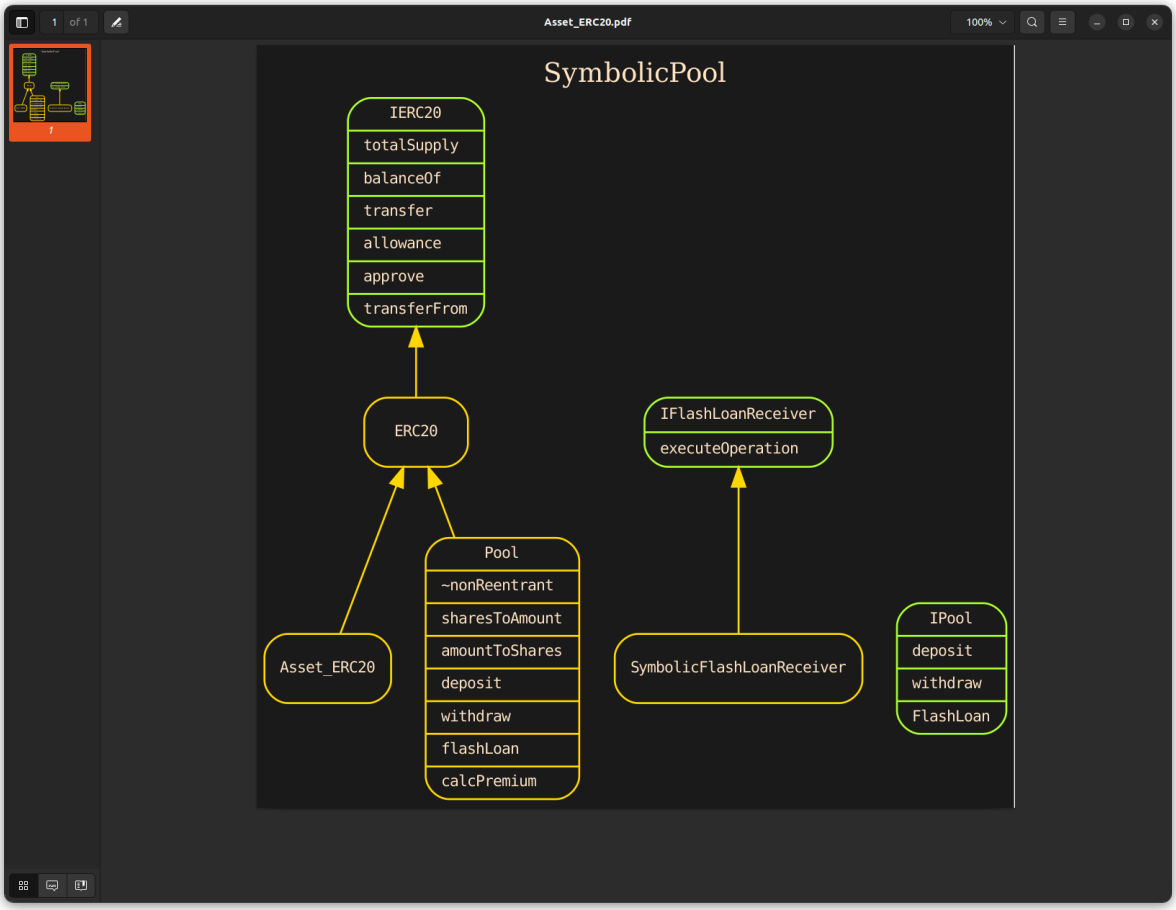
Column Header	2nd Column Header	3rd Column Header
Row 1 Column 1 item	Row 1 Column 2 item	An item
An item	Row 2 Column 2 item	Row 2 Column 3 item

Adding an image

To insert an image or picture use the `.. image` directive, as shown below. The specified path to the image `images/symbolic_pool_diagram.png` is relative to the file containing the directive.

```
.. image:: images/symbolic_pool_diagram.png
   :alt: This text will be displayed if the image is broken
```

Rendered as:



Notes

The image path
: A relative path should be relative to the `.rst` file. An absolute path is treated as relative to the top `source/` directory. See Sphinx Image Directive for more on this.

Additional options
: Options, such as alternative text for missing images and scaling, are described in Docutils Image Directive.

Adding a video clip

To add a video clip file we use the sphinxcontrib-video extension. Note that the preferred folder to place the video file is the `source/_static/` folder. For example:

```
.. video:: ../_static/lesson4_invariants/ball_game/InvariantsClip_subtitles.mp4
   :alt: The text shown when the video cannot be displayed
   :height: 250
```

Rendered as:

See also:

See sphinxcontrib-video Quickstart for additional options.

Combining closed captions

You cannot use a separate file for the closed captions (subtitles). Instead you must embed the closed captions inside the video itself.

Here is one recipe to include a closed captions file in your video. Suppose you have an mp4 video `InvariantsClip.mp4` and a closed captions file named `InvariantsClip.srt`, you can combine them using the FFmpeg package with the following command:

```
ffmpeg -i InvariantsClip.mp4 -vf subtitles=InvariantsClip.srt InvariantsClip_
↪subtitles.mp4
```

Todo: Missing topics to add:

- table ot contents (mainly the `hidden` option)
- adding images and using the `only-light` and `only-dark` classes in furo
- tabs (from sphinx-design)
- footnotes
- `.. rubric`, `.. centered` and `.. hlist`

2.4 Reference guide

Scripts reference

Quickstart script

Quickly start a Certora document project

```
usage: certora-doc-quickstart [-h] -p PROJECT [-v VERSION] [-r RELEASE] [--theme HTML_
↪THEME] [--code CODE_PATH]
                              [--no-link-to-github]
                              [PROJECT_DIR]
```

Positional Arguments

PROJECT_DIR project root path, defaults to current working dir

Named Arguments

-p, --project project name

Versioning

Sphinx supports a notion of a "version" and a "release" for the project.

-v, --version	version of project
-r, --release	release of project

Style

Available themes: insipid - clean and minimal, light mode only; furo - clean customisable theme, light and dark modes; piccolo_theme - minimal, light and dark modes; sphinx_rtd_theme - Read The Docs theme, light mode only; classic - builtin, light mode only; sphinxdoc - builtin, light mode only

--theme	Possible choices: insipid, furo, piccolo_theme, sphinx_rtd_theme, classic, sphinxdoc html theme for the project, defaults to furo

Code links

Determine location to search for code and link style.

--code	path of code folder, relative to the source dir, defaults to source dir
--no-link-to-github	link to local files instead of github

Configuration

The configuration is determined by the `/source/conf.py` file. See Sphinx - Configuration for a full list of configurable properties.

In addition, various extensions and themes have their own configurations possible, for example:

- The Furo theme
- The spelling extension sphinxcontrib.spelling

Main configurable options

`project`
: The project's name, also the title of the html and pdf documents.

`html_title`
: Optional title to use in the side-bar, defaults to `project`.

`exclude_patterns`
: A list of paths and patterns to ignore. This helps reduce warnings regarding paths under the `/source/` folder that are not part of the document tree.

`rst_prolog`
: A string of reStructuredText that will be included at the beginning of every source read. This is useful for adding default roles.

Codelink extension

This is a Sphinx extension for linking source code files. The resulting links are either to local files, or to Github, depending on the configuration.

The code for this extension is at *docsinfra.sphinx_utils.codelink_extension*.

Configuration

Adding the extension

To use the extension you must add `docsinfra.sphinx_utils.codelink_extension` to the `extensions` list in the `source/conf.py` file, as shown below. This is done automatically by the *Quickstart script*.

```
extensions = [
    "docsinfra.sphinx_utils.codelink_extension",
    "docsinfra.sphinx_utils.includecvl",
```

Options

code_path_override
: Optional string, determines the absolute code path. Absolute paths in `:clink:` are considered as relative to the *absolute code path*. By default, this path is the source directory (e.g. `docs/source/`). This options changes the absolute code path to the one given in `code_path_override`. Note `code_path_override` must be relative to the source directory.

link_to_github
: Boolean, if true the links will be to the Github remote repository (deduced from the repository of the path given in `:clink:`). Otherwise will link to local files.

Usage

Syntax

```
* :clink:`Optional name <path-to-code>` - in this case "Optional name"
  will be displayed. As noted above, absolute links will be considered as relative
  to the *absolute code path*.
* :clink:`path-to-code` - in this case the "path-to-code" will be the link's text.
```

Examples

Listing 13: rst

```
* Reference to a folder: :clink:`Voting folder </voting>`
* Reference to a file: :clink:`Voting_solution.spec </voting/Voting_solution.spec>`
* Reference without text: :clink:`/voting/Voting_solution.spec`
```

Rendered as:

- Reference to a folder: Voting folder
- Reference to a file: Voting_solution.spec
- Reference without text: /voting/Voting_solution.spec

Github linking notes

- If the *code* folder is not part of a git repository, the extension will fall back to local links.
- Determining the link to the correct file depends on Github's current conventions, and will likely fail for other hosting services.
- The extension will use the *current active branch* for the link. If the git repository is in *detached head* state (common in git sub-modules), it will try to deduce the correct branch.

Include CVL extension

This Sphinx extension for including CVL elements from spec files in the document. It is able to include invariants, rules and ghosts by name. The code for this extension is at `docsinfra.sphinx_utils.cvlinclude`.

Important: This extension uses the CVLDoc package.

Configuration

Adding the extension

To use the extension you must add `docsinfra.sphinx_utils.cvlinclud` to the `extensions` list in the `sourc/conf.py` configuration file, as shown below. This is done automatically by the *Quickstart script*.

```
extensions = [
    "docsinfra.sphinx_utils.codelink_extension",
    "docsinfra.sphinx_utils.includecvl",
```

Usage

Syntax

```
.. cvlinclude:: <spec-file-path>
   :cvlobject: <rule-name> <another-rule-name> ...
   :spacing: 2
```

`spec-file-path`
: Path to spec file. If relative should be relative to the current file. If absolute, it will be considered as relative to the `/source/` directory.

`:cvlobject:`
: A list of names of to include. Accepts rules, invariants and ghosts. To include the methods block, add `methods` to this list. The source code for these elements will appear in the order they are given, including the documentation.

`:spacing:`
: The number of lines between two elements, defaults to one.

In addition, this extension support all the options of the literalinclude directive, such as `:caption:` and `:emphasize-lines:`.

Important: Since CVLDoc omits **hook** statements, this extension cannot be used to include hooks. Use `literalinclude` if you need a **hook** code snippet.

Important: If omitting the `:cvlobject:` option, you must add the `:language: cvl` option, since the extension will not assume this code is CVL.

Example

```
.. cvlinclude:: /../../code/voting/Voting_solution.spec
   :cvlobject: methods onlyLegalVotedChanges sumResultsEqualsTotalVotes
   :spacing: 2
   :caption: Voting rules
   :emphasize-lines: 2
```

Rendered as:

Listing 14: Voting rules

```
/**
 * # Simple voting contract complete spec
 *
 * To use gambit, run from the tutorials-code root folder the following command:
 * `certoraMutate --prover_conf solutions/lesson4_invariants/simple_voting/Voting_
↪solution.conf --mutation_conf solutions/lesson4_invariants/simple_voting/mutate.
↪json`
 */
methods
{
    function votesInFavor() external returns (uint256) envfree;
    function votesAgainst() external returns (uint256) envfree;
    function totalVotes() external returns (uint256) envfree;
    function hasVoted(address) external returns (bool) envfree;
}


/// @title No illegal changes to `_hasVoted`
invariant onlyLegalVotedChanges()
    !illegalStore;


/// @title Sum of voter in favor and against equals total number of voted
invariant sumResultsEqualsTotalVotes()
    votesInFavor() + votesAgainst() == to_mathint(totalVotes());
```

Code documentation

Codelink extension `docsinfra.sphinx_utils.codelink_extension`

A Sphinx extension for linking source code files, either locally or to Github.

class CodeLinkConfig(*env: BuildEnvironment*)

The configuration needed for code links.

classmethod add_config_values(*app: Sphinx*)

Add the config values neede for `CodeLink`.

get_abs_path(*path: str*) → Path

Returns an absolute path to the file. If the path is relative, or there is no code-path override, we use `BuildEnvironment.relfn2path` to compute the path. Otherwise, the path is considered sa relative to the overridden code path.

Examples:

```
>>> codelinkconfig._code_path_override
PosixPath('/home/shoham/dev/docs-infrastructure/code')
>>> codelinkconfig.get_abs_path('voting/Voting_solution.spec')
PosixPath('/home/shoham/dev/docs-infrastructure/docs/source/voting/Voting_
↪solution.spec')
```

class GithubUrlsMaker

Computes the url in Github of a given file, returns None if the computation failed for any reason.

> **Warning:** The url is computed by reverse engineering Github's urls. This is prone to breaking.

Todo: Cache repositories and their url's.

---

get_repo(*path: Path*) → Repo | None

Returns
the path's repository

is_github_url(*url: str*) → bool

Returns whether the given url is in Github.com.

normalize_url(*url: str*) → str

Convert remote repo urls to `https://` urls. For example:

```
>>> GithubUrlsMaker().normalize_url('git@github.com:Certora/docs-
↪infrastructure.git')
'https://github.com/Certora/docs-infrastructure/'
```

```
>>> GithubUrlsMaker().normalize_url('https://github.com/Certora/Examples.git
↪')
'https://github.com/Certora/Examples/'
```

class TutorialsCodeLink(*fix_parens: bool = False*, *lowercase: bool = False*, *nodeclass: type[docutils.nodes.Element] | None = None*, *innernodeclass: type[docutils.nodes.TextElement] | None = None*, *warn_dangling: bool = False*)

Sphinx role extension for linking source code files locally in the user's chosen code path.

process_link(*env: BuildEnvironment*, *refnode: Element*, *has_explicit_title: bool*, *title: str*, *target: str*) → tuple[str, str]

Called after parsing title and target text, and creating the reference node (given in *refnode*). This method can alter the reference node and must return a new (or the same) `(title, target)` tuple.

result_nodes(*document: document*, *env: BuildEnvironment*, *node: Element*, *is_ref: bool*) → tuple[list[docutils.nodes.Node], list[docutils.nodes.system_message]]

Called before returning the finished nodes. *node* is the reference node if one was created (*is_ref* is then true), else the content node. This method can add other nodes and must return a `(nodes, messages)` tuple (the usual return value of a role function).

Include CVL extension `docsinfra.sphinx_utils.includecvl`

A Sphinx extension which adds a Sphinx directive for including CVL snippets from spec files.

class CVLInclude(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Extends LiteralInclude to enable including CVL elements. To include cvl elements use the `cvlobject` option and provide a list of CVL elements names, separated by spaces. To include the methods block use `methods`. Also adds the `spacing` option which determines the number of lines between CVL elements.

option_spec: OptionSpec = {'append': <function unchanged_required>, 'caption': <function unchanged>, 'class': <function class_option>, 'cvlobject': <function unchanged_required>, 'dedent': <function optional_int>, 'diff': <function unchanged_required>, 'emphasize-lines': <function unchanged_required>, 'encoding': <function encoding>, 'end-at': <function unchanged_required>, 'end-before': <function unchanged_required>, 'force': <function flag>, 'language': <function unchanged_required>, 'lineno-match': <function flag>, 'lineno-start': <class 'int'>, 'linenos': <function flag>, 'lines': <function unchanged_required>, 'name': <function unchanged>, 'prepend': <function unchanged_required>, 'pyobject': <function unchanged_required>, 'spacing': <class 'int'>, 'start-after': <function unchanged_required>, 'start-at': <function unchanged_required>, 'tab-width': <class 'int'>}

Mapping of option names to validator functions.

class CVLIncludeReader(*filename: str*, *options: dict[str, Any]*, *config: Config*)

Extends LiteralIncludeReader by allowing to access CVL elements in spec files.

3 Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index