TRUSTED® COMPUTING GROUP

GUIDANCE

TCG Markdown User's Guide

Version 0.7.2
January 20, 2024

Contact: admin@trustedcomputinggroup.org

TCG Draft

DISCLAIMERS, NOTICES, AND LICENSE TERMS

CHANGE HISTORY

Revision	Date	Description
0.1/1	2023/12/17	Initial draft

DOCUMENT STYLE

Key Words

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this document’s normative statements are to be interpreted as described in RFC 2119: Key words for use in RFCs to Indicate Requirement Levels.

Statement Type

Please note an important distinction between different sections of text throughout this document. There are two distinctive kinds of text: *informative comments* and *normative statements*. Because most of the text in this specification will be of the kind *normative statements*, the authors have informally defined it as the default and, as such, have specifically called out text of the kind *informative comment*. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind *informative comment*, it can be considered a *normative statement*.

EXAMPLE:

Start of informative comment

This is the first paragraph of 1–n paragraphs containing text of the kind informative comment …

This is the second paragraph of text of the kind informative comment …

This is the nth paragraph of text of the kind informative comment …

To understand the TCG specification, the user must read the specification. (This use of MUST does not require any action).

End of informative comment

Contents

1 Scope and Purpose **8**

2 Getting Started **9**
2.1 Creating a Repository 9
2.2 GitHub Actions 9
2.3 Local Testing 11
2.4 TCG Document Boilerplate 11
2.4.1 Front Matter Variables 11
2.4.2 Backslash Macros 12

3 Collaboration Model **13**
3.1 Sending a Pull Request 13
3.2 Reviewing a Pull Request 14

4 Using Markdown **16**
4.1 Basic Formatting 16
4.2 Cross-References 17
4.2.1 Sections 18
4.2.2 Tables 18
4.2.3 Figures 18
4.2.4 Equations 18
4.3 Informative Blocks 18

5 Figures **20**
5.1 Images 20
5.2 Mermaid Charts 20
5.2.1 Sequence Diagrams 20
5.2.2 Flow Charts 21

6 Tables **23**
6.1 Markdown Tables 23
6.2 HTML Tables 23

7 Math **25**
7.1 Equations 25
7.2 Inline math 25
7.3 Words in equations 25

8 Advanced Features **26**
8.1 Git Version Parsing 26
8.1.1 Conventions for Release Naming 26
8.2 Git Status Parsing 26
8.3 Running Pandoc with Releases 26

Appendices **29**

A Reporting Issues with the Tools **30**

List of Tables

3 Shapes 23
4 Fruits 24

List of Figures

1 The “Use this template” button . . . 9
2 GitHub Collaboration Workflow . . . 13
3 GitHub Edit Button . . . 13
4 Propose Changes Dialog . . . 14
5 Plus Button . . . 15
6 Finish Review . . . 15
7 Adding an Image . . . 20
8 Startup Sequence . . . 21
9 Flowchart . . . 22

1 Scope and Purpose

The purpose of this guide is to demonstrate the usage of Markdown-plus-GitHub document-authorship flows for TCG workgroup usage.

This document contains a boilerplate section at the front called Document Style. This section is typically included in TCG Specifications and isn't as relevant for Guidance and Reference documents. It's included here, mainly to demonstrate the usage of Markdown for specifications.

2 Getting Started

2.1 Creating a Repository

You can create a repository from scratch, or you can use the template repository to get started a little more quickly. There's a little green "Use this template" button in the top right (see Figure 1).

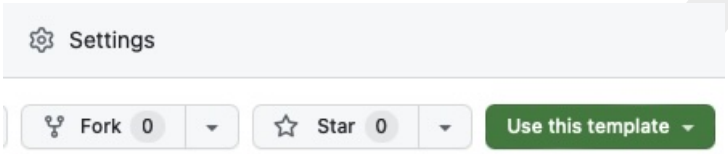

Figure 1: The "Use this template" button

2.2 GitHub Actions

Even if you used the template repository, please double-check this. As the tools are being actively developed, there is probably a newer version of the tools available for you!

`.github/workflows/main.yml` might look a bit like this:

```yaml
# Render the spec to PDF on pull requests, and check in the rendered PDF on commits
# to main.

name: Render

on:
  push:
    branches:
      - main
  pull_request:
  workflow_dispatch:

jobs:
  render:
    runs-on: ubuntu-latest
    # This Docker container contains all the Pandoc dependencies for rendering TCG
    # Markdown docs.
    container:
      # IMPORTANT: Check https://github.com/TrustedComputingGroup/pandoc/releases
      # for the latest!
      image: ghcr.io/trustedcomputinggroup/pandoc:0.6.2
    name: Render PDF
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Render
        # This GitHub action provides an easy way to use the Docker container above
        # with your document.
        # IMPORTANT: Check https://github.com/TrustedComputingGroup/markdown/releases
        # for the latest!
        uses: trustedcomputinggroup/markdown@v0.4.0
        with:
          input-md: main.md
          output-pdf: spec.pdf
          output-docx: spec.docx

      - name: Upload samples
        # This GitHub action uploads samples into the "Checks" tab (for pull requests),
        # so that reviewers can easily see how a proposed change will look in the
        # finished spec.
        uses: actions/upload-artifact@master
        with:
          name: preview
          path: |
            spec.pdf
            spec.docx

      - name: Check in latest render
        # This GitHub action automatically renders and checks-in the PDF produced in
        # the above step. This makes it easier for people to grab the latest rendered
        # version of the document.
        uses: stefanzweifel/git-auto-commit-action@v4
        with:
          commit_message: Generate latest PDF
          file_pattern: spec.pdf
        if: github.event_name != 'pull_request'
```


2.3 Local Testing

These tools have a number of dependencies on LaTeX and LaTeX plugins. The simplest way to get a consistent build is to use the docker container that gets used for the GitHub actions.

`docker_run` is provided as a convenience script for Linux systems.

Usage:

```
./docker_run --pdf=output.pdf ./input.md
```

You can specify a particular version of the docker container using the `DOCKER_IMAGE` environment variable:

```
DOCKER_IMAGE=ghcr.io/trustedcomputinggroup/pandoc:0.6.5 ./docker_run --pdf=output.pdf
 ↪  ./input.md
```

If you're working on a change to these tools, it can be beneficial to build and tag a local version of the container and then run it locally:

```
docker build --tag working .

DOCKER_IMAGE=working:latest ./docker_run --pdf=output.pdf ./input.md
```

2.4 TCG Document Boilerplate

There are several sections that are recommended for use in every TCG Markdown document.

The trickiest section is the YAML front matter at the very top of the Markdown file. It looks like this:

```
---
title: "TCG Markdown User's Guide"
version: 0.1
revision: 1
date: 12/17/2023
type: GUIDANCE
status: Draft
...
```

This section provides metadata to the tools.

2.4.1 Front Matter Variables

2.4.1.1 title

REQUIRED.

`title` is the title of the document.

2.4.1.2 version

REQUIRED.

`version` is the version of the document.

2.4.1.3 revision

REQUIRED.

`revision` is the revision of the document.

2.4.1.4 date

REQUIRED.

`date` is the full date of the document, in YYYY/MM/DD form.

2.4.1.5 type

REQUIRED.

`type` should be one of: “SPECIFICATION”, “GUIDANCE”, or “REFERENCE”. It appears on the title page on the left-hand side.

2.4.1.6 status

REQUIRED.

`status` should be one of: “Draft”, “Review”, or “Published”.

If it is not “Published”, then a gray watermark “DRAFT” will appear on all pages after the title page.

2.4.1.7 template

OPTIONAL.

`template` should be one of: `greentop`, `bluetop`. It customizes the title page style. If not provided, `greentop` is the current default.

Typically, specifications use the `greentop` template, and guidance or reference documents use the `bluetop` template.

2.4.2 Backslash Macros

Understanding of LaTeX is not required in order to use TCG Markdown tools. However, a few special macros can be used from Markdown to fully specify a TCG document.

2.4.2.1 Table of Contents

After the boilerplate sections, most TCG documents should set up the tables of contents, and lists of tables and figures.

```
\tableofcontents

\listoftables

\listoffigures
```

Almost every document should have a table of contents. Some documents may not need lists of tables or figures.

2.4.2.2 Appendices

At the end of the last “regular” section of the document, use

```
\beginappendices
```

to mark the transition to the “Appendix” portion of the document. Only documents that have appendices are expected to use this macro.

3 Collaboration Model

Users familiar with Git and who prefer to use their own tools may choose to skip this section.

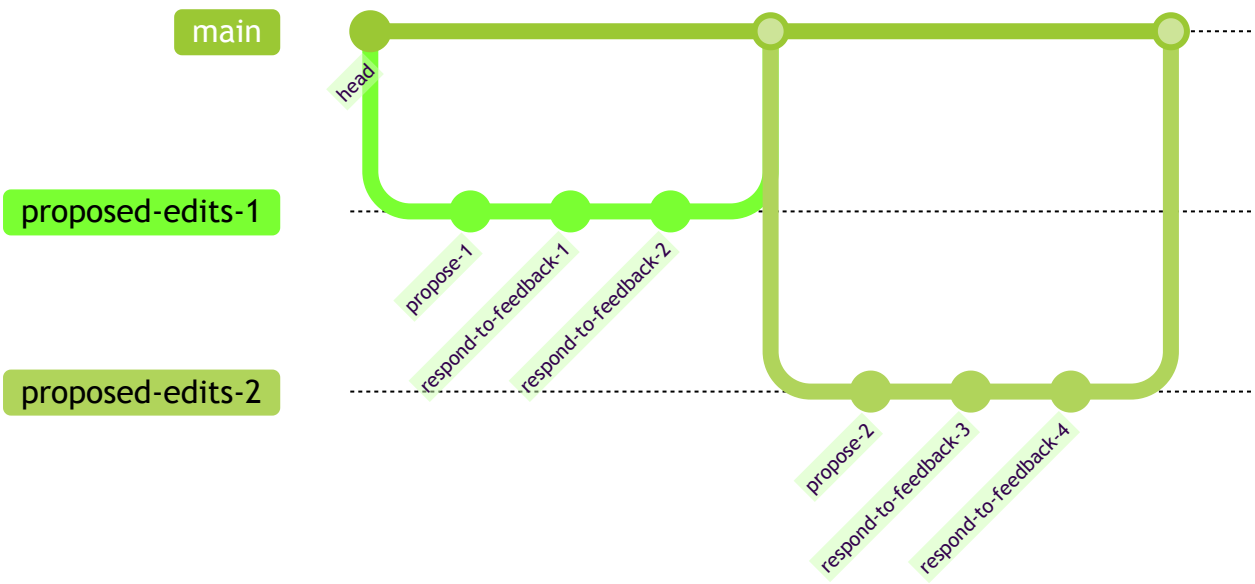

Figure 2: GitHub Collaboration Workflow

As visualized in Figure 2, proposed changes to a GitHub Markdown repository take the form of "Pull Requests" (PRs). A *proposer* of a change proposes a PR that changes some files in the repository. This PR contains an initial *commit*, which is a unit of change to files. *Reviewers* can provide comments and suggestions on the proposed edits. The *proposer* can respond to the feedback by adding additional *commits* into their PR. When all parties are satisfied, the PR is *approved* and *merged* into the main repository.

3.1 Sending a Pull Request

When you navigate to a GitHub repository containing Markdown, you can view the Markdown files by clicking on them.

From this view, there is an "Edit" (pencil) button in the upper right-hand corner, pictured in Figure 3:

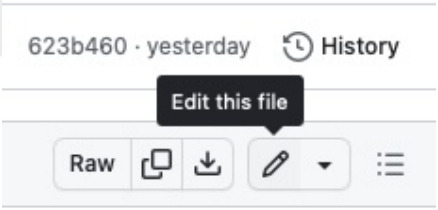

Figure 3: GitHub Edit Button

This will take you to a view where you can edit the file. There is a "Preview" button that you can use to see roughly how the changes will look when viewed from GitHub. Most everyday changes to TCG docs can be previewed in high enough fidelity with this tool.

When you're satisfied with your changes, use the green "Commit changes…" button. This will bring up the dialog box pictured in Figure 4:

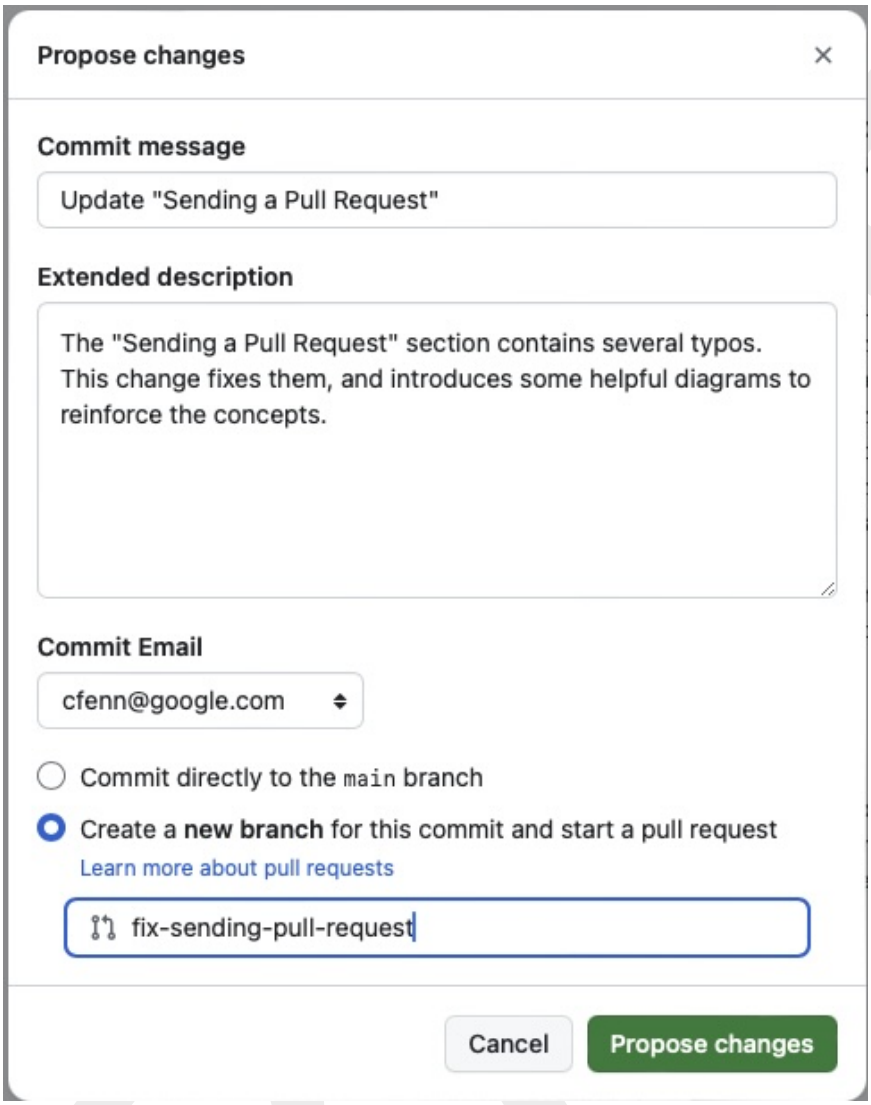

Figure 4: Propose Changes Dialog

Include a descriptive commit message, extended description, and new branch name for your change, then click the green “Propose changes” button.

You’re almost done! This will take you to a page called “Open a pull request”. You can provide some additional context to reviewers about why you want to make this change. When you’re satisfied, click “Create pull request.”

3.2 Reviewing a Pull Request

After a PR has been sent by someone else, you can review the changes with the “Add your review” button in the upper right-hand corner of the change description page.

This button takes you to a review flow, where you can provide comments on individual lines of the changes. You can leave a comment on an individual line by mousing over the line and clicking the blue “+” button, which looks like Figure 5:

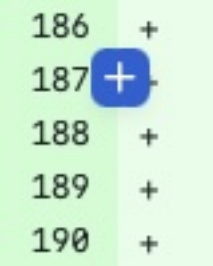

Figure 5: Plus Button

After you've gone through all the changed files and provided your comments, you can click "Review changes" to finish the review. This dialog looks like Figure 6:

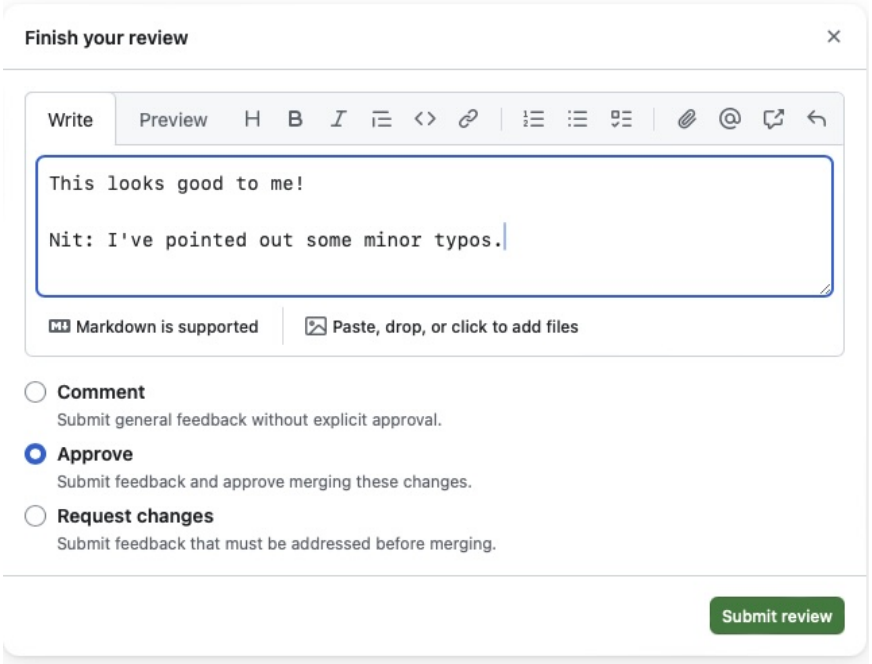

Figure 6: Finish Review

Here, you can provide summary comments and mark your review as one of:

- Comment (Just providing feedback)
- Approve (Approving the changes)
- Request changes (Explicitly not approving the changes, with specific actionable feedback)

4 Using Markdown

Markdown is intended to be a lightweight language for authoring documents. Most of the time, it looks exactly the same as plain text.

4.1 Basic Formatting

The structure of the document is guided by lines that begin with #:

```
# Section Titles

## Subsection Titles

### Sub-subsection Titles
```

and so on.

When you put `*asterisks*` around a word, it renders as *italics*.

When you put `**double asterisks**` around a word, it renders as **boldface**.

When you put `` `backticks` `` around a word, it renders as `monospace`.

To force a new page, use `---`. This will appear as a horizontal line in GitHub and a page break in the PDF.

When you need to write math, use `$ dollar signs $` for inline math notation, or `$$ double dollar signs $$` for equations. This is explained in more detail in Clause 7.

Numbered and bulleted lists begin with numbers and asterisks:

```
* Something
* Something else

1. First thing
2. Second thing
```

Becomes:

- Something
- Something else

1. First thing
2. Second thing

Hyperlink syntax for the TCG Website looks like: `[TCG Website](https://trustedcomputinggroup.org)`.

Hyperlink syntax for the Using Markdown section looks like: `[Using Markdown](#using-markdown)`. If you provided a stable cross-referencing link like this document for Cross-References, you can use it like: `[Cross-References](#sec:cross-references)`.

You can use triple backticks like so to create blocks of code:

````
```
int i = 42;
```
````

The result looks like this:

```
int i = 42;
```

You can tell Markdown what language the code is in, to get syntax highlighting:

````
```c
// Awesome!
int i = 42;
```
````

The result looks like this:

```c
// Awesome!
int i = 42;
```

4.2 Cross-References

In general, sections, tables, figures, and equations can be referenced using the @ symbol. These cross-references to not show up in the GitHub markdown, but will appear in the final document.

4.2.1 Sections

When you add `{#sec:section-reference}` at the end of a section title, as in:

```
## Cross-References {#sec:cross-references}
```

it creates a cross-reference that you can use with `@sec:section-reference`. For example, `@sec:cross-references` Clause 4.2.

4.2.2 Tables

See Clause 6 for more information about cross-references to tables.

4.2.3 Figures

See Clause 5 for more information about cross-references to figures.

4.2.4 Equations

See Clause 7 for more information about cross-references to equations.

4.3 Informative Blocks

TCG uses a special visual style to demarcate informative non-binding remarks within specifications.

We use the Markdown “quote block” for this purpose. A quote looks like this:

```
> This is the only informative text block in this document.
>
> These blocks can contain multiple paragraphs, tied together by lines containing just
> ">".
>
> These blocks can even contain tables! However, be wary of providing tables that are
> too large in an Informative Text block.
>
> | **Document Type** | **Informative Blocks** |
> | ----------------- | ---------------------- |
> | SPECIFICATION     | Usually                |
> | GUIDANCE          | Rarely                 |
> | REFERENCE         | Rarely                 |
```

The above Markdown code becomes:

Start of informative comment

This is the only informative text block in this document.

These blocks can contain multiple paragraphs, tied together by lines containing just “>”.

These blocks can even contain tables! However, be wary of providing tables that are too large in an Informative Text block.

Document Type	Informative Blocks
SPECIFICATION	Usually
GUIDANCE	Rarely
REFERENCE	Rarely

End of informative comment

5 Figures

There are two ways to include a figure in a document: as an image file checked into the repository, and as a Mermaid diagram.

5.1 Images

Upload plain image files into the repository with the “Add file” button.

For compatibility reasons, all image files should be in the root of the repository (main directory). In the future, better support for organizing figures may be added to these tools.

See Clause 3.1 for the flow that needs to be followed for getting your image uploads reviewed. You can add more changes to the branch for the PR that reference the image, or you can do it in a subsequent PR.

Markdown syntax for including an image looks like `![Figure Title](filename)`. For example:

```
![Adding an Image](add_plus_button.jpg){#fig:add-plus-button width=60%}
```

becomes:

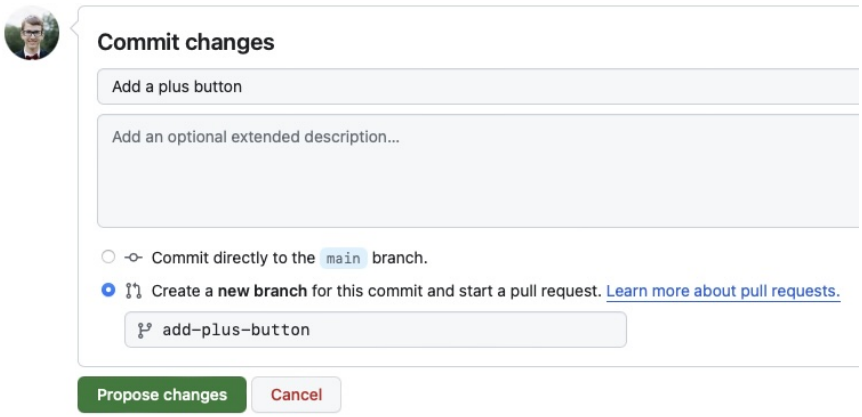

Figure 7: Adding an Image

The `{#fig:add-plus-button}` attribute (note there are no spaces between the `)` and the `{!)` does two things:

1. Includes the figure in the List of Figures (if you used `\listoffigures` as described in Clause 2.4.2.1).
2. Numbers the figure so you can reference it as Figure 7 by just typing `@fig:add-plus-button`.

Including `width=60%` here specifies that the image should take up 60% of the page’s width.

5.2 Mermaid Charts

Mermaid is a language for text-based diagrams for inclusion in Markdown documents. See the Mermaid website for a more exhaustive list of types of diagrams.

5.2.1 Sequence Diagrams

Mermaid supports swim-lane digrams like Figure 8 with the following notation:

```mermaid {caption="Startup Sequence" #fig:startup}
sequenceDiagram
Host->>TPM: TPM2_Startup
loop Measurements
    Host->>TPM: TPM2_PCR_Extend
end
Host->>TPM: TPM2_Quote
TPM->>Host: <quoted PCRs>
```

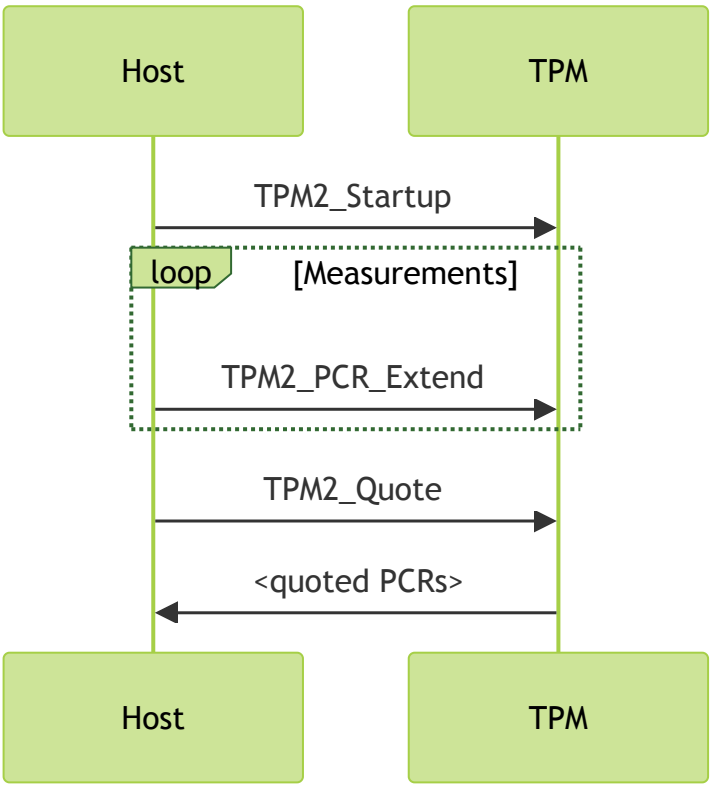

Figure 8: Startup Sequence

Crossreferences to Mermaid diagrams are supported by providing both `caption` and `#fig:xxxxx` classes in curly braces.

5.2.2 Flow Charts

Mermaid supports flow-charts like Figure 9 with the following notation:

```mermaid {caption="Flowchart" #fig:flowchart}
graph TD;
    A-->B;
    A-->C;
    B-->D;
    C-->D;
```


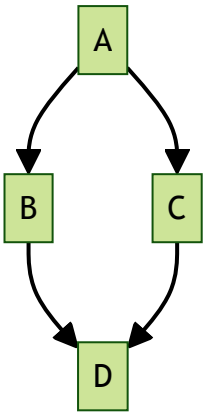

Figure 9: Flowchart

6 Tables

We support two notation styles for tables: Markdown and HTML.

6.1 Markdown Tables

Small, simple tables like Table 3 are easier to read in raw Markdown form in the following style:

```
Table: Shapes {#tbl:shapes}

| **Shape**    | **Number of sides** |
| ------------ | ------------------- |
| Square       | 4                   |
| Triangle     | 3                   |
| Möbius strip | 1                   |
```

Table 3: Shapes

Shape	Number of sides
Square	4
Triangle	3
Möbius strip	1

Note the table caption and cross-reference in curly braces above the table.

6.2 HTML Tables

For larger, or more complex tables like Table 4, it may be preferable to use HTML. If you want to make table cells span rows or columns, this is the only way to do it.

```
<table id="tbl:fruits">
    <caption>Fruits</caption>
    <tr>
        <th colspan="2">Color and Fruit</th>
        <th>Mistaken for Vegetable</th>
    </tr>
    <tr>
        <td>Red</td>
        <td rowspan="2">Apple</td>
        <td>No</td>
    </tr>
    <tr>
        <td>Green</td>
        <td>No</td>
    </tr>
    <tr>
        <td>Red</td>
        <td>Tomato</td>
        <td>Yes</td>
    </tr>
    <tr>
        <td>Yellow</td>
        <td>Banana</td>
        <td>No</td>
    </tr>
</table>
```

The above HTML table becomes the below:

Table 4: Fruits

Color and Fruit		Mistaken for Vegetable
Red	Apple	No
Green		No
Red	Tomato	Yes
Yellow	Banana	No

Note the table caption in the `<caption>` element, and the table cross-reference in the `id` attribute of the `<table>` element.

To get an HTML table to word-wrap its contents, use `<colgroup>` to style the width of the columns.

```
<colgroup>
<col style="width: 25%" />
<col style="width: 75%" />
</colgroup>
```


7 Math

7.1 Equations

Markdown supports inline math notation. For example, Equation 1 can be typeset as:

```
$$ \nexists {n \ge 3; a, b, c \in \mathbb{Z}} \mid a^n + b^n = c^n $$ {#eq:fermat}
```

Note the `{#eq:fermat}` at the end of the equation. This allows referencing Equation 1 with `@eq:fermat`.

$$\nexists n \geq 3; a, b, c \in \mathbb{Z} \mid a^n + b^n = c^n \tag{1}$$

7.2 Inline math

Sometimes, you just need a little inline math in the middle of a sentence, like with `$a^2 + b^2 = c^2$` to get $a^2 + b^2 = c^2$.

7.3 Words in equations

To typeset complex equations with multi-character identifiers (such as the function “HMAC” or the word “OPAD”) in Equation 2, we recommend using the functions `\symbf` (for functions) and `\symit` (for identifiers). This avoids strange kerning issues where a string is treated as a product of single-character symbols, like in Equation 3:

```
$$ \symbf{HMAC}(K, \symit{"text"}) \coloneq H((\bar{K} \oplus \symit{OPAD}) \Vert
 ↪  H((\bar{K} \oplus \symit{IPAD}) \Vert \symit{"text"})) $$ {#eq:hmac-iso}
```

$$\mathbf{HMAC}(K, "text") \coloneqq H((\bar{K} \oplus OPAD) \| H((\bar{K} \oplus IPAD) \| "text")) \tag{2}$$

```
$$ HMAC(K, "text") \coloneq H((\bar{K} \oplus OPAD) \Vert H((\bar{K} \oplus IPAD) \Vert
 ↪  "text")) $$ {#eq:hmac-iso-bad-kerning}
```

$$HMAC(K, "text") \coloneqq H((\bar{K} \oplus OPAD) \| H((\bar{K} \oplus IPAD) \| "text")) \tag{3}$$

8 Advanced Features

In the GitHub action YAML, you can enable some advanced features.

8.1 Git Version Parsing

Use `extra-build-options: "--gitversion"` to let Git number the document for you.

```
      - name: Run the action
        uses: trustedcomputinggroup/markdown@latest
        with:
          extra-build-options: "--gitversion"
```

When you do this, the tool will check for a recent release in the repository. It will use the major.minor version number from the tag as the document version, and the number of commits since that tag as the revision. This way, you don't have to manually update the version or revision numbers in your document!

8.1.1 Conventions for Release Naming

The tooling expects the following conventions for tagging your releases:

- `vX.Y` indicates a regular draft of version X.Y.
- `rX.Y` indicates a review draft of version X.Y.
- `pX.Y` indicates a published version.

8.2 Git Status Parsing

Use `extra-build-options: "--gitstatus"` to let Git number AND set the status of the document for you.

```
      - name: Run the action
        uses: trustedcomputinggroup/markdown@latest
        with:
          extra-build-options: "--gitstatus"
```

See Conventions. When `--gitstatus` is enabled, the leading character (which is expected to be one of: `v`, `r`, or `p`) is used to determine the document's status at revision 0. Commits on top of any type of version are always considered to be drafts.

8.3 Running Pandoc with Releases

Clause 2.2 shows an example of a GitHub action that automatically runs Pandoc on every pull request and push to the repository.

You may wish to run the workflow on releases, and attach the results to the release page, for example to have it generate a docx file to send to the Technical Committee for review, or when publishing a final version of a document.

Use the example below as a guide for how you can have Pandoc automatically render the doc (maybe basing its status on the released tag).


```yaml
# Render the spec to PDF and Word on releases.

name: Render (PDF and Word)

on:
  release:
    types: [released]

jobs:
  render-spec-pdf:
    runs-on: ubuntu-latest
    container:
      image: ghcr.io/trustedcomputinggroup/pandoc:0.7.1
    name: Render (pdf)
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Render
        uses: trustedcomputinggroup/markdown@v0.4.2
        with:
          input-md: spec.md
          extra-build-options: "--gitstatus"
          output-pdf: spec.pdf

      - name: Upload to release
        uses: svenstaro/upload-release-action@v2
        with:
          repo_token: ${{ secrets.GITHUB_TOKEN }}
          file: spec.pdf
          tag: ${{ github.ref }}
          overwrite: true
          body: "Part 1 (PDF)"

  render-spec-docx:
    runs-on: ubuntu-latest
    container:
      image: ghcr.io/trustedcomputinggroup/pandoc:0.6.8
    name: Render Part 1 (docx)
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Render
        uses: trustedcomputinggroup/markdown@v0.4.2
        with:
          input-md: spec.md
          extra-build-options: "--gitstatus"
          output-docx: spec.docx

      - name: Upload to release
        uses: svenstaro/upload-release-action@v2
        with:
          repo_token: ${{ secrets.GITHUB_TOKEN }}
          file: spec.docx
          tag: ${{ github.ref }}
          overwrite: true
          body: "Part 1 (Word)"
```


Appendices

A Reporting Issues with the Tools

Please report issues with the tooling at https://github.com/TrustedComputingGroup/pandoc/issues.

