ITUPublications
Recommendations

International Telecommunication Union
Standardization Sector

Recommendation
<acronym>

<series or category>

<subject line>

<subtitle of this document>

<subtitle of this document>

ABSTRACT

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

HISTORY

Version	Document	Approval	Focus Group	Working Group
<version>	<title>	<date>	TIDA	<acronym>

To access this Document as a PDF, open the published documentation site in your web browser and click the PDF download link, or navigate directly to https://debora-com.github.io/CUTEspecs/specification.pdf.

AUTHORS

Name	Affiliation	Contact
<Jane Doe>	<Organization / Company>	<jane.doe@example.com>
<John Smith>	<Organization / Company>	<john.smith@example.com>

KEYWORDS

Lorem, ipsum, dolor, consectetur, adipiscing.

FOREWORD

Agentic AI systems now act as autonomous entities capable of reasoning, planning, and executing tasks across services, infrastructures, and organizational boundaries. This fundamentally changes the nature of identity in digital systems: identity is no longer limited to human users and static machine instances but must also encompass autonomous, dynamic, and goal-oriented agents. And identity alone is insufficient — the core question extends beyond "what an entity is" to "whether, and under what conditions, that entity should be trusted to act".

The Focus Group on Trust and Identity for humans and agentic AI (FG-TIDA) addresses trust management and interoperable digital identity infrastructure for humans and for agentic AI, supporting the development of secure, trustworthy digital ecosystems in which humans and agentic AI can safely interact and collaborate.

FG-TIDA was established under Recommendation ITU-T A.7 (Focus groups: Establishment and working procedures), with ITU-T Study Group 17 as its parent group. Further details are available in the group's terms of reference.

For more news and updates about the TIDA Community, including its supporter organizations, please refer to the community's published resources.

NOTE

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium.

CONTENTS

Writing content

> **Note**
>
> This template is *self-documenting*: each chapter both explains and demonstrates a part of the authoring system. Read the rendered page to see what is possible, then open the matching `.rst` source file to see exactly how it was produced. Replace this instructional content with your own.

This chapter covers the everyday building blocks of a document: headings, text, lists, links, cross-references, highlighted notes, and code. Chapter 2 covers figures, tables and diagrams; Chapter 4 covers API reference documentation.

0.0.1 Headings and structure

Headings define the structure of a chapter. A heading is a line of text with a row of punctuation underneath it, at least as long as the text. The *character* used sets the level:

- = for the chapter title (top level),
- - for a section (like this one),
- " for a subsection.

Subsection example

This is a subsection. Chapters are numbered automatically, so you never type "1.2.3" yourself — Sphinx computes it from the heading levels and the order of files.

0.0.2 Text styling

Within a paragraph you can mark text as *italic* (one asterisk), **bold** (two asterisks), or `inline code` / literal text (two backticks). Use inline code for field names, values, and anything a reader should type verbatim, such as `transactionId` or `application/json`.

0.0.3 Lists

Use a bulleted list for items with no particular order:

- First point.
- Second point, which can wrap onto several lines and still belong to the same item as long as the following lines are indented to match.
- Third point.

Use a numbered list when order or count matters. Start every item with `#.` and the numbers are filled in for you:

1. Collect the input.
2. Validate it.
3. Produce the result.

A *definition list* pairs a term with its explanation — useful for glossaries or parameter descriptions:

Term
: The explanation of the term, indented underneath it.

Another term
: Its explanation.

0.0.4 Links and cross-references

Link to an external site with the text and URL together, followed by an underscore, like the TIDA site.

To point at another part of *this* document, give a target a label and refer to it by name. This paragraph can, for example, send the reader to *Figures, tables and diagrams* — the link text and page number update automatically if the target moves. Labels are defined with `.. _name:` immediately above a heading (see the top of any chapter source file).

0.0.5 Highlighting: notes, warnings, tips

Admonitions draw the eye to something important. Several types are available, each with its own colour and icon:

Note

A note adds supporting information the reader should be aware of.

Tip

A tip offers helpful, optional advice.

Warning

A warning flags something that can cause errors or data loss.

Important

An “important” box stresses a point that must not be missed.

0.0.6 Code and data samples

Show code or data in a highlighted block. Name the language after `.. code-block::` to get syntax colouring — here, `json`:

```
{
    "key": "value",
    "anotherKey": 23
}
```

0.0.7 Requirement keywords

Specifications rely on precise requirement language. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

Keep this section in your specification and use those words deliberately — readers and implementers rely on their exact meaning.

Figures, tables and diagrams

This chapter demonstrates the visual elements you will use most in a specification: images, tables in two different styles, and diagrams generated from text. As before, read the rendered page, then open `chapter2.rst` to see the markup.

0.0.8 Figures

A *figure* is an image with a caption. Put your image file in the top-level `images/` folder and reference it with a relative path. The `:width:` option scales it to a share of the page width.

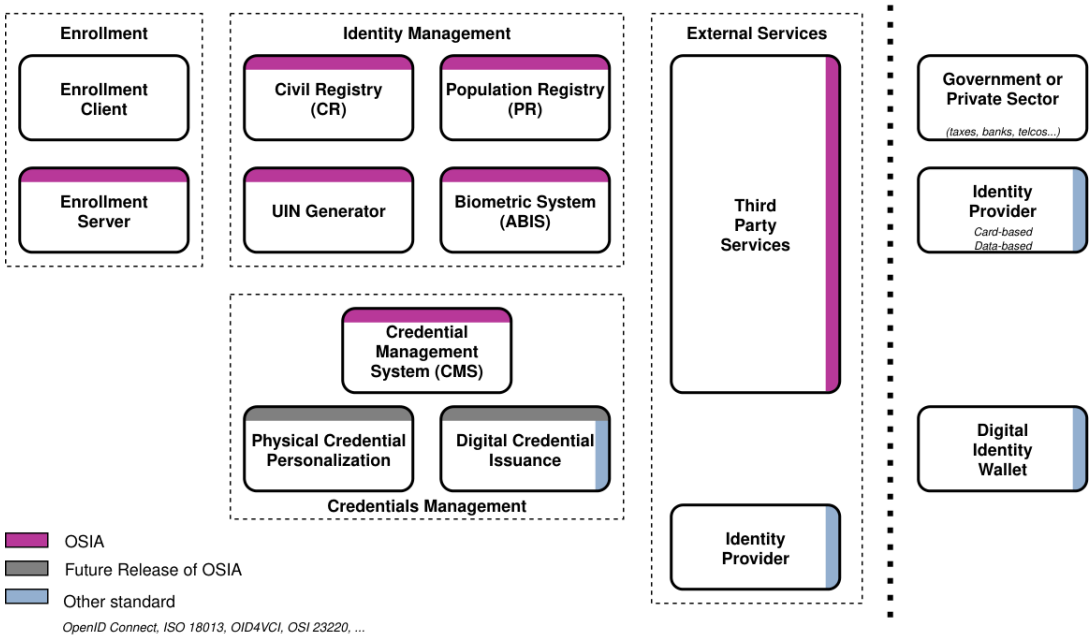

Fig. 1: A figure caption appears below the image and is numbered automatically.

Use a single raster image (PNG) so the same file works in both the web pages and the PDF. Vector formats (SVG) look sharper on screen but need a separate PDF copy, so PNG keeps the template simple.

0.0.9 Tables

Two table styles are available; pick whichever is easier for the content.

A **list-table** is the easiest to edit: every cell is a bullet, so you never have to align columns by hand. Add or remove rows freely.

Table 1: A simple list-table

Field	Description
id	The unique identifier.
name	A human-readable name.

A **grid-style table** can group rows under spanning headers, which is useful for mapping matrices. Its one rule: the cell text MUST line up under the = and - border characters, or the build fails with “Malformed table”. Edit it carefully, keeping each column within its width.

Table 2: A grouped table

Rows	Column A	Column B	Column C
Group One			
Item One	X	X	
Item Two	X	X	X
Group Two			
Item Three	X		X

0.0.10 Diagrams

Diagrams are written as *text* and rendered to images at build time using PlantUML — so they stay editable, version-controlled, and consistent in style. This is a sequence diagram describing an interaction between three parties:

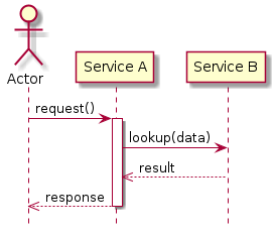

Fig. 2: A sequence diagram, written as text

The same `.. uml::` block draws many other diagram types. A *class diagram*, for example, is a convenient way to document a data model — the entities, their fields, and the relationships between them:

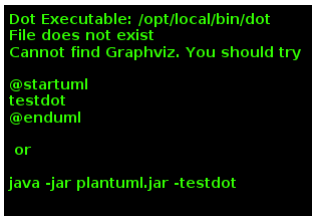

Fig. 3: A class diagram describing a data model

PlantUML can also draw use-case diagrams, activity diagrams, state machines and more — see https://plantuml.com for the full syntax.

> **Note**
>
> Diagrams require PlantUML to be available at build time. The online build (GitHub Pages) installs it automatically; for local builds see the project HOWTO.

Organising a large document

As a specification grows, you will want to split it across several source files. Sphinx offers two mechanisms, and they produce **different results in the navigation sidebar** — so it matters which you choose.

0.0.11 toctree

A `toctree` links to other files as *separate pages*, each with its own entry in the navigation sidebar. This is how the chapters of this document are joined together (see `index.rst`).

You write it like this, listing each sub-page (without the `.rst` extension):

```
.. toctree::

   _part1
   _part2
```

Each listed file becomes its own page. In the sidebar, the parent gets an **expand arrow** and the sub-pages appear nested beneath it. The reader clicks through to a separate page for each part.

0.0.12 include

An `include` pastes another file's content directly into the current page, as if you had typed it there. The result is a **single continuous page** with no extra sidebar entries and no expand arrow.

You write it like this, giving the path to each fragment:

```
.. include:: _part1.rst

.. include:: _part2.rst
```

Use it to break one long chapter into smaller source files — for example, so several authors can edit different parts without touching the same file — while the reader still sees one seamless chapter.

Two rules keep includes working:

- **Name included fragments with a leading underscore** (for example `_part1.rst`) so the build skips them as standalone pages.
- **Demote the headings inside a fragment by one level.** The chapter title uses = underlines; a heading inside an included fragment must use - so it becomes a *section* of the chapter rather than a new chapter of its own.

0.0.13 Which to use

Important

Because the two approaches look different in the sidebar, mixing them makes the navigation inconsistent. **Pick one and use it throughout your document.** This template recommends the `include` approach: it keeps each chapter on a single page with a clean, uniform sidebar, and lets you still split the source into manageable files. Chapters 4, 5 and 6 of this template are all built with `include`.

Tip

A change to an *included* file does not always trigger a rebuild of the page that includes it. If an edit does not appear, rebuild fully — the project `HOWTO` explains how, and the live-preview helper is already configured to do this for you.

Documenting an API

This chapter demonstrates the template's most powerful feature: generating a complete REST API reference automatically from an OpenAPI specification. You write the API once, as a standard OpenAPI (Swagger) file, and the endpoints, parameters, request and response schemas, and examples are rendered for you.

This is why the template is built on Sphinx rather than plain Markdown — no Markdown toolchain offers this.

0.0.14 How it works

1. Put your OpenAPI 3.0 file in the `yaml/` folder (see `yaml/example.yaml` for a minimal, complete example you can copy).
2. Create one page per interface that points at the file (see `chapter4/interface1.rst`). Copy it for each additional API.
3. List those pages in the `toctree` below so each gets its own page in the navigation sidebar.

The example interface on the next page is generated entirely from `yaml/example.yaml` — edit that file and the documentation follows.

0.0.15 Example Interface

This is version 1.0.0 of this interface.

Get the OpenAPI file: example.yaml

Operations

- *listItems*
- *createItem*

Services

default

`GET /v1/items`

Retrieve the list of items.

Scope required: `items.read`

Query Parameters

- **`limit`** (*integer*) – Maximum number of items to return.

Status Codes

- 200 OK – The list of items.
- 400 Bad Request – Bad request.

Example request:

```
GET /v1/items?limit=1 HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": "1",
        "name": "First item"
    },
    {
```

(continues on next page)

(continued from previous page)

```
        "id": "2",
        "name": "Second item"
    }
]
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "code": "string",
    "message": "string"
}
```

`POST /v1/items`

Create a new item.

Scope required: `items.write`

Form Parameters

- **body** – The item to create.

Status Codes

- 201 Created – The item was created.
- 400 Bad Request – Bad request.

Example request:

```
POST /v1/items HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "id": "3",
    "name": "New item"
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": "string",
    "name": "string"
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "code": "string",
    "message": "string"
}
```

Annex

0.0.16 Annex title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

0.0.17 Annex title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Appendix

0.0.18 Appendix title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

References

[JSON Web Token]
IETF RFC 8725, *JSON Web Token Best Current Practices*,

https://datatracker.ietf.org/doc/html/rfc8725

[OpenAPI]
OpenAPI Specification v3.0.3 (2020),

https://spec.openapis.org/oas/v3.0.3

[WSQ]
NIST; FBI; *Los Alamos Natinal Laboratory. Wavelet Scalar Quantization algorithm 2: PDF 2.0*,

https://www.nist.gov/system/files/documents/2020/09/03/11-wsq_bradley_brislawn_standard_ieee_iscs_-_19940530.pdf

LIST OF TABLES

LIST OF FIGURES

HTTP ROUTING TABLE

/v1