OpenCPI
Documentation Writer Guide

OpenCPI Release: v2.4.8

Revision History

Revision	Description of Change	Date
1.0	Creation	2022-08-26
1.1	Add updated man page information and updated chapter on creating doc assets from User Guide	2022-10-28
1.2	Update versions	2025-02-13

Table of Contents

1 Working with OpenCPI Documents

OpenCPI supports the following document types:

- Reference documents
- Tutorials
- Briefings
- Man pages
- Datasheets (and other OpenCPI asset documents
- Getting started guides

All these document types are available on the OpenCPI website opencpi.gitlab.io. The figure on the next page illustrates the different audiences and reading paths for these document types. The chapters that follow describe how to create and maintain each of these document types.

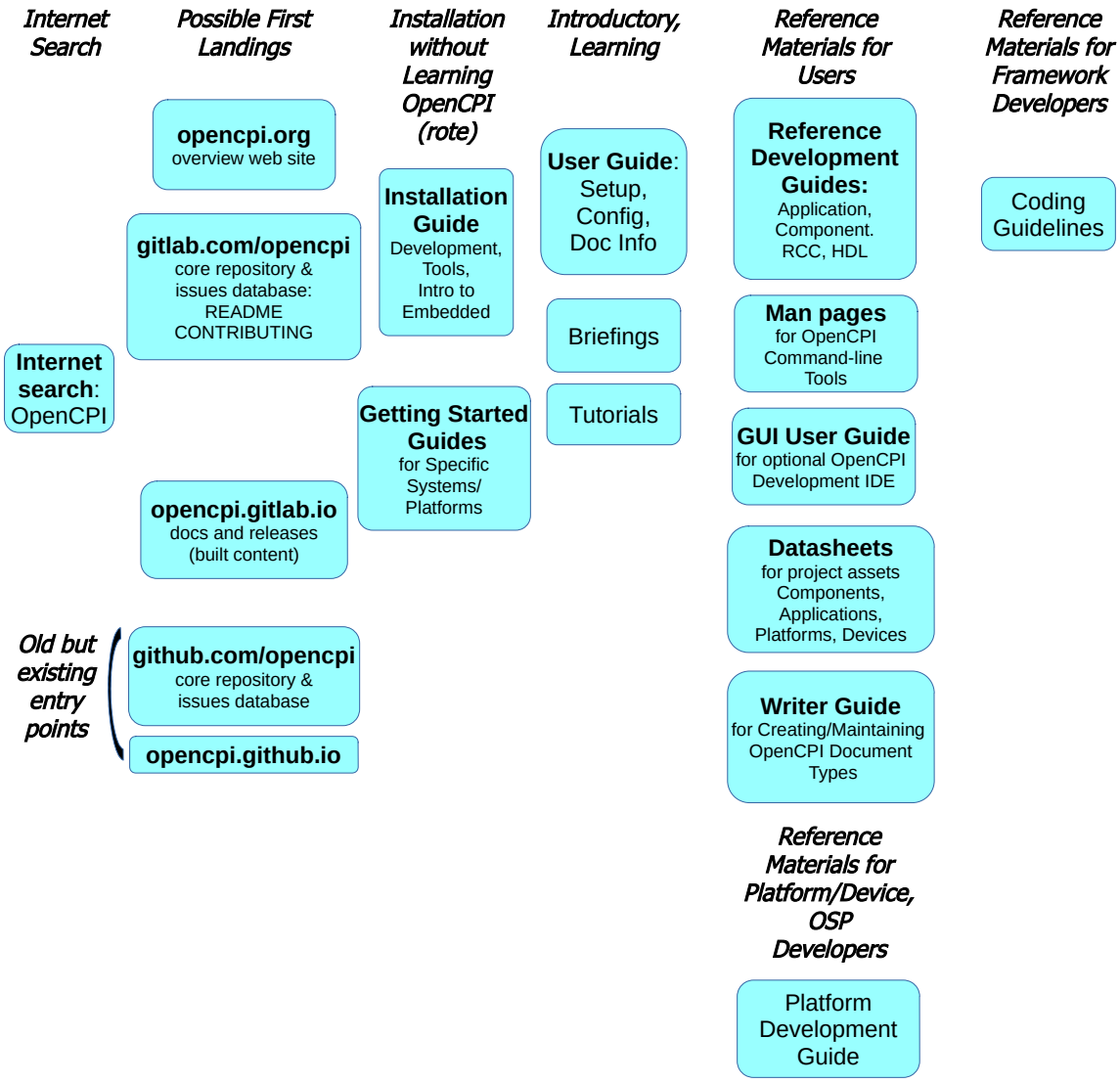


Figure 1: Documentation for OpenCPI

2 Writing Best Practices That Apply to All OpenCPI Documents

Spell out acronyms the first time you use them. For example, analog-to-digital converter (ADC), software-defined radio (SDR). The spelled-out versions don't have to be initial capitalized but sometimes it makes sense to do it.

Blank sections followed by subsections are not allowed. Every section heading must have at least a sentence stating what the subsection contains.

Use section titles or text, not section numbers, for cross references between sections. Section numbers change more frequently than section titles. For example, don't use "Section 3.4 provides…", use "the section "developing an RCC worker" or "the section that describes RCC worker development". Create the hyperlink to the text (LibreOffice).

Use double quotation marks to highlight a term, not single quotes. Better practice is to use italics (FirstEmphasis character style in LibreOffice).

In general, it's best to avoid using Latin abbreviations like i.e, e.g., etc. Instead of i.e., use "that is", instead of e.g., use "for example", instead of etc., use "and so on" or "and others". The abbreviations I.e and e.g. should be followed by a comma (e.g., "i.e., "). However, Latin abbreviations are frequently used in the OpenCPI reference documents.

Don't use "this" by itself. Always identify what "this" is. For example, don't say "this causes", say "this problem causes…"

Use "use", not "utilize". They mean the same thing but "use" is one syllable and thus simpler to read through. Using "utilize" makes things unnecessarily complex.

The free tool Grammarly is a good source for writing hints.

3 Working with OpenCPI Reference Documents

An OpenCPI reference document should provide a complete description of all supported features and concepts in a development area of OpenCPI. Existing OpenCPI reference documents target the three different types of OpenCPI development – application, component, and platform – and the Resource-Constrained C (RCC) and Hardware Definition Language (HDL) authoring models.

The ***OpenCPI Installation Guide***, ***OpenCPI User Guide***, and ***OpenCPI Glossary*** are also types of reference documents.

OpenCPI reference document sources are located in the directory **`$OCPI_ROOT_DIR/doc/reference`** in the OpenCPI repository on GitLab. They are LibreOffice Writer flat ODT (FODT) files formatted using an OpenCPI-defined template named **`OCPI_ODT.ott`** located in the **`shared`** subdirectory. This template supplies OpenCPI-defined paragraph and character styles and must be associated with every OpenCPI reference document. The section "Working with the OpenCPI Documentation Template" provides more detail about the template.

In addition to the OpenCPI document template, the **`shared`** subdirectory also supplies:

- The OpenCPI Glossary **`glossary.fodt`**. This file is included as a LibreOffice "section" as the last chapter of each reference document and in the standalone ***OpenCPI Glossary*** reference document. This is the file where new terms and their definitions should be added. Changes to this file are automatically propagated to the reference documents when their FODT file is opened with LibreOffice and updating the links to the most recent version is confirmed.
- The release subtitle file **`release.fodt`**. This file is included as a LibreOffice "section" on the title page of each reference document. It allows the OpenCPI version number to be automatically set on the title page and should not be edited.
- The OpenCPI supported systems and platforms file **`systems.fodt`**. This file is included as a LibreOffice "section" in the bodies of the ***OpenCPI Installation Guide*** and ***OpenCPI User Guide***. This file needs to be updated when new supported systems and platforms are added to OpenCPI. The changes are automatically propagated to the ***OpenCPI Installation Guide*** and ***OpenCPI User Guide*** when the FODT file is opened with LibreOffice and updating the links to the most recent version is confirmed.
- The OpenCPI version file **`release.txt`**. This file is used by the script that produces the document types for the opencpi.gitlab.io website and should not be edited.

Note: The **`reference`** directory in the OpenCPI repository does not currently store:

- Sources for diagrams and screenshots used in the ***`.fodt`** documents

- PDF files generated from the `*.fodt` documents

Reference document sources for optional OpenCPI features like the OpenCPI GUI can be found in the GitLab repository for the feature.

To work with a reference document:

- Decide which version of LibreOffice to use
- Create the new LibreOffice document (optional)
- Follow the OpenCPI documentation conventions when creating/adding content
- Create hyperlinks between sections in the reference document as necessary
- Create hyperlinks to other OpenCPI documents or relevant URLs as necessary
- Create diagrams from PowerPoint slides or WaveDrom timing diagrams as necessary
- Create a PDF with bookmarks for document review
- Prepare the document for a GitLab merge

To be supplied: a checklist for confirming that the document conforms to all documentation conventions and all relevant procedures have been followed.

4 Working with OpenCPI Tutorials

An OpenCPI tutorial is an interactive, “how to” learning document that teaches by example and supplies the information necessary to complete a particular task. A tutorial presents one or more objectives and goals and one or more step-by-step examples that demonstrate how to achieve these objectives and goals. A tutorial requires a reader to participate by performing each step that the tutorial describes. OpenCPI provides a series of tutorials that introduce concepts and procedures that build on each other and move from simple to complex tasks.

The OpenCPI tutorials are located in the directory **`$OCPI_ROOT_DIR/doc/tutorials`** in the OpenCPI repository on GitLab. They are LibreOffice Writer flat ODT (FODT) files formatted using the OpenCPI-defined template **`$OCPI_ROOT_DIR/doc/reference/shared/OCPI_ODT.ott`** that supplies OpenCPI-defined paragraph and character styles. This template must be associated with every tutorial.

Besides the OpenCPI document template, there are two files in **`$OCPI_ROOT_DIR/doc/reference/shared`** that apply to OpenCPI tutorials:

- The release subtitle file **`release.fodt`**. This file is included as a LibreOffice “section” on the title page of each tutorial. It allows the OpenCPI version number to be automatically set on the title page and should not be edited.
- The OpenCPI version file **`release.txt`**. This file is used by the script that produces the document types for the opencpi.gitlab.io website and should not be edited.

Note: The **`tutorials`** directory in the OpenCPI repository does not currently store:

- Sources for diagrams and screenshots used in the ***`.fodt`** documents
- PDF files generated from the ***`.fodt`** documents

To work with a tutorial:

- Decide which version of LibreOffice to use
- Create the new LibreOffice document (optional)
- Follow the OpenCPI documentation conventions when creating/adding content
- Create hyperlinks between sections in the tutorial as necessary
- Create hyperlinks to other OpenCPI tutorials, reference documents or relevant URLs as necessary
- Create conditional sections for all GUI instructions

- Create diagrams from GUI screenshots, PowerPoint slides or WaveDrom timing diagrams as necessary
- Hide the conditional sections to make sure they all disappear and only the CLI text remains. Ensure that the CLI-only text flows properly when the GUI sections are hidden and there are no empty paragraphs.
- Create a PDF with bookmarks for document review
- Create a CLI-only PDF for document review (optional)
- Prepare the document for a GitLab merge

To be supplied: a checklist for confirming that the document conforms to all documentation conventions and all relevant procedures have been followed.

5 Working with OpenCPI Briefings

The documents in the **`$OCPI_ROOT_DIR/doc/briefings`** directory in the OpenCPI repository on GitLab are LibreOffice Impress presentation flat ODP (FODP) files. Each briefing is associated with an OpenCPI template (**`Briefing_template.otp`**). However, as of this writing, work needs to be done on the template for it to control the briefings in the same way as the OpenCPI document template controls the reference and tutorial documents.

To work on the briefings, you use the same LibreOffice application to open a briefing as you do for reference and tutorial documents.

You need to follow the steps described in "Preparing a LibreOffice Document for Merge" before submitting a briefing.

The briefings use the Tahoma font, while the reference and tutorial documents use Arial.

There are five master slides available, but not all of them are used. To edit a master slide, for example, to change the date on a briefing footer:

- Click Master Slides in the toolbar on the right.
- Right-click on a master slide and select **Edit Master...**
- Click in the footer and edit the date.
- Click **View** > **Normal** to exit the master slide. The new date should appear in the footer.

To see which master slides are applied to which presentation slides, click on a slide with the Master Slide window open. The master slide that the selected slide is using is highlighted in the Master Slide window.

When creating or editing a briefing, follow the conventions described in "Documentation Conventions" for code styles and italics and bold.

See this page for LibreOffice Impress documentation.

To be supplied: a checklist for confirming that the document conforms to all documentation conventions and all relevant procedures have been followed.

6 Working with LibreOffice Writer Documents

This section provides information about using LibreOffice Writer as it applies to working with OpenCPI reference and tutorial documents.

6.1 Which Version of LibreOffice to Use

You can use any version of LibreOffice on any operating system the version runs on to work on OpenCPI documents. For example, this document was prepared with LibreOffice 6.4.7.2 running on an Ubuntu computer. That said, the current OpenCPI target environment is LibreOffice 6.x running on a Rocky8 operating system, and the tool that generates PDF files from the LibreOffice documents and publishes them on opencpi.gitlab.io operates in this target environment. Document authors are encouraged to be kind to the OpenCPI maintainers who coordinate OpenCPI documentation releases either by:

- Doing their authoring/editing work in the target environment, or:
- Following the procedures described in "Preparing a LibreOffice Document for Merge" to ensure compatibility with the current OpenCPI target environment before committing and pushing the final document version and creating a merge request.

Be aware that all merge requests that contain documents that have been authored/edited with LibreOffice (and any other Office-related tools) must be reviewed and tested for compatibility with the target environment before the merge request can be granted.

See this document for a description of the latest version of LibreOffice Writer.

The following screenshot shows the recommended LibreOffice settings (the image is from version 6.4.7.2) when working with OpenCPI documents. See the LibreOffice documentation for your version for details.

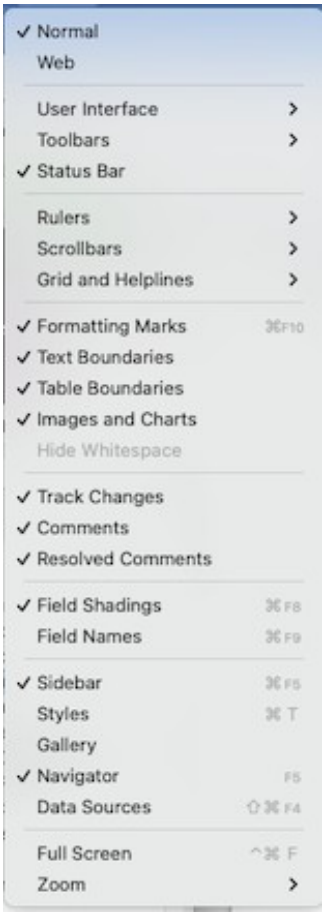


Figure 2: Recommended LibreOffice Settings (6.4.7.2)

6.2 Creating a New LibreOffice Document

To create a new document, it's best to copy one of the existing LibreOffice documents, like the User Guide or a tutorial, and then remove the text you don't want, leaving a skeleton with the title page and the TOC. This way, you get the correct (OpenCPI) template by default and a title page and TOC already formatted for you. See "Working with the OpenCPI Document Template" for details about the template. The rest of this section contains information about:

- Document title naming conventions
- Source file naming conventions
- Table of Contents
- Footer format
- OpenCPI Release section

6.2.1 Document Title Naming Conventions

Reference documents always include "OpenCPI" as the first word in the title. Reference documents are nearly always called "Guides".

Tutorials always include "OpenCPI Tutorial <n>: " as the beginning of the title.

6.2.2 Source File Naming Conventions

Source file names for the documents in **`$OCPI_ROOT_DIR/doc/reference/`** always use "**`OpenCPI_`**" in the filename and should end with "**`_Guide`**" unless that name doesn't make sense (for example, for "Glossary"). When creating a new reference document, follow the naming conventions used for the other files in the **`reference`** directory.

When creating a new tutorial, follow the naming conventions used for the other files in the **`tutorial`** directory. Source file names for the tutorials in **`$OCPI_ROOT_DIR/doc/tutorials/`** always begin with **`Tutorial_<n>_`** and always end with **`-cg`**.

Use the **`.fodt`** format for all files, and not the **`.odt`** format.

6.2.3 Table of Contents

If you've created a new reference document or tutorial from an existing document, a formatted Table of Contents is already there. To update the TOC, right-click in the gray area and then select **Update Index**.

6.2.4 Footer Format

The footer of an OpenCPI document contains the document title on the left and an automatically generated page number on the right. To update the footer to the correct document title, scroll down to the bottom of the page (on any page, but usually the title page) and click anywhere in the existing footer title. Type the title into the footer and then click in the page body to exit the footer. There is no additional formatting applied to a title in a footer.

6.2.5 OpenCPI Release Section

The title page of every reference and tutorial document contains a special LibreOffice section located underneath the document title. The purpose of this section is to include the **`shared/release.fodt`** file, which refers to the **`shared/release.txt`** file. When a new version of OpenCPI is released, the **`shared/release.txt`** file is updated with the latest OpenCPI version. The **`shared/release.fodt`** file picks up this update, and because it is included as a section in the reference or tutorial document, the main document displays it on its title page.

When you work on a reference or tutorial document in your local working branch, the OpenCPI release section is set to “develop” by default. If you take your document out of your working branch to work on it “offline”, the OpenCPI release section will be empty because LibreOffice can't locate the **`shared/release.fodt`** file. (References to other files in **`shared/`**, like the glossary (**`shared/glossary.fodt`**) and the tables of supported systems and platforms (**`shared/systems.fodt`**), will also be empty.) If you then copy this “offline” file back into your working branch, you may need to update the OpenCPI release section with the path to **`shared/release.fodt`** from your working branch if it doesn't automatically reset to this value and instead shows as an empty section on the title page.

To update the OpenCPI release section:

- Right-click on the paragraph marker in the section and select **Edit Section**.
- In the dialog, click **Browse** and then navigate to the location of **`shared/release.fodt`** in your working branch. Select **`release.fodt`** and then click **OK**.

The OpenCPI release should now appear on the title page.

6.3 Documentation Conventions

The most important convention is to *use the paragraph and character styles in the template* and *stay away from direct formatting*.

The next sections provide some general conventions and details on how to use the paragraph and character styles in the template.

6.3.1 General Conventions

When referring to the ***OpenCPI framework***, use OpenCPI, *not* ocpi, OCPI, opencpi, etc.

When a term or document name is introduced for the first time, use the ***FirstEmphasis*** character style. If a term is used afterward that still should not read as "just a regular word", continue to use this style.

Use two spaces to separate sentences.

Use the Body paragraph style for “body” text, which is the main descriptive text that falls within section headings in a typical OpenCPI document. For example, all the sentences above use the Body paragraph style.

6.3.2 Heading Styles

Use the Heading *n* paragraph styles for heading levels 1-5. For all levels, heading titles must be in “title” case, not “sentence” case. For example, “Adding a Screenshot to a Tutorial”, and not “Adding a screenshot to a tutorial”.

Heading titles generally use the “gerund” naming convention (adding “ing” to the verb), like “Creating…”, “Editing”, “Installing”. When a section has subheadings that correspond to installation and setup steps, these subheadings can use the “imperative” convention, like “Create”, “Edit”, “Install”. See the OpenCPI tutorials for examples.

Both Heading 1 and Heading 2 paragraph styles automatically insert a page break between Heading 1 and Heading 2 sections. This automatic page break can often result in a lot of white space on the page if the heading contains only one or two paragraphs.

6.3.3 Code Styles

Use the **`Code`** character style in the text for terms or names meant to be literal names in code or XML files. For example, OpenCPI commands like **`ocpidev`**, **`ocpirun`**, **`ocpiadmin`**.

Use the Code paragraph style for code/XML examples that are paragraphs by themselves. Lines in the paragraph must end in shift-return (line break), not separate paragraphs. Here is an example:

```
This is the code paragraph style
This is another line of code in the same paragraph
This is the end of the code example
```

Code in table cells should be in the "table code" paragraph style.

6.3.4 Table Styles

Use the Table Caption (before) paragraph style for table captions. The caption should precede the table.

In the heading line of a table, use the Table Heading paragraph style.

In the body of a table, use the Table Contents paragraph style.

When code is in table cells, use the Table Code paragraph style.

When symbols that are case insensitive are put into table entries, capitalize/camel-case them.

When literal symbols that are case sensitive are in table entries, leave them in their actual case.

See the ***OpenCPI Application Development Guide***, the ***OpenCPI Component Development Guide***, and **`shared/systems.fodt`** for examples of tables.

6.3.5 Diagram (Figure) Styles

Use the Diagram paragraph style for figures, aka “diagrams”. Diagrams can be:

- Illustrations, which are usually created with LibreOffice Draw or MicroSoft PowerPoint. These should always be saved as Scalable Vector Graphics **`.svg`** files. (Older files have been saved as **`*.png`** and **`*.jpg`**, but **`*.svg`** is current best practice.)
- Screenshots, which are usually copied from a screen capture tool and possibly then to an editing tool (like Preview on a Mac computer).

Always use **Anchor** > **As character** to anchor the diagram. See the section on adding screenshots to OpenCPI tutorials for details. The section “Adding a PowerPoint Slide as a Diagram” describes how to import a diagram from PowerPoint into LibreOffice, while the section “Adding a Timing Diagram Created with WaveDrom” explains how to add an SVG file generated from WaveDrom.

Use the Caption paragraph style for diagram captions. The caption should follow the diagram. Here is an example:

The following diagram shows the ZCU102 board layout:

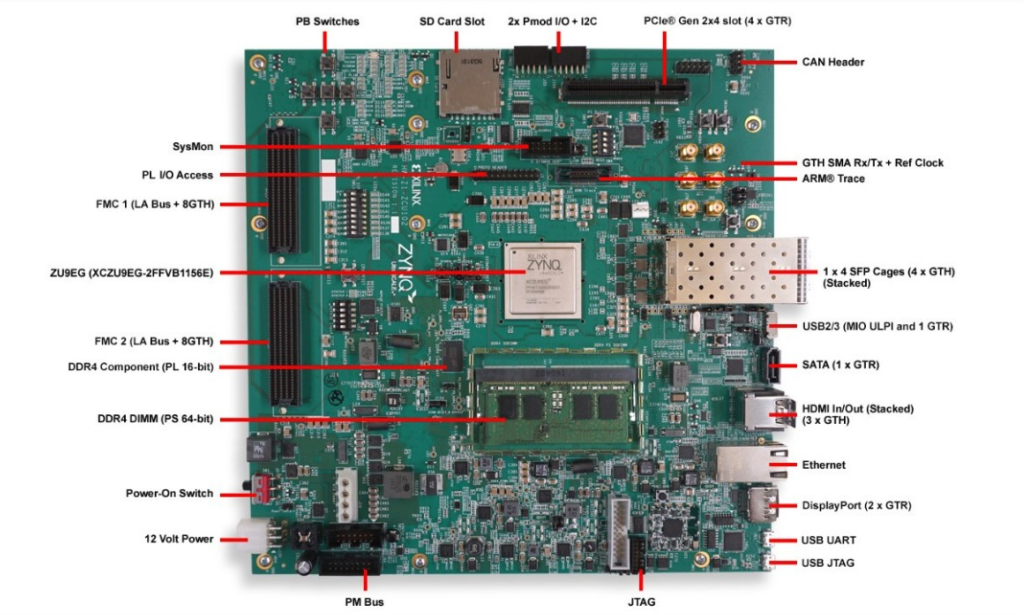


Figure 1: Xilinx ZCU102 Board Layout

To create a numbered figure caption:

- Create an empty Caption paragraph.
- In the paragraph, type in **Figure** and then select **Insert** > **Field** > **More Fields**.
- In the pop-up dialog, in Type, select **Number Range**, in Select, select **Diagram**, and in Format, select **Arabic 1 2 3**. Click **Insert**. The number (which increments automatically when you save the document file) appears. Type in the colon character, two spaces, and the remaining text for the caption.

Note: screenshots in tutorial steps don't need figure captions.

6.3.6 List Styles

Use the List Bullet or List Bullet 2 paragraph styles for bulleted lists. In bulleted lists, use periods when the bullets are complete sentences, otherwise do not use periods. All items in the list should be the same in this regard. For example, if the first two bullets are not complete sentences, but the third one contains two sentences, all bullets should end with periods regardless of whether they're a sentence or just a phrase. Here is an example:

- My first bullet point.
- My second bullet point.
- My third bullet point. This one has two sentences and sub-bullets:

Sub-bullet 1

Sub-bullet 2, which has a third level:

Note there is no bullet character on second or third levels.

Use the ListNumber paragraph style for numbered lists.

Use initial caps for each list item in a bulleted or numbered list. For example:

These are the things you need to do:

- First thing
- Second thing
- Third thing

Don't do it this way:

- first thing
- second thing
- third thing

When you're creating a sentence in the format <word> <hyphen> <text>, LibreOffice Writer automatically changes the hyphen to an em-dash when you start typing <text>. Use this format for bulleted list items when you want to introduce a term or concept and then describe it in more detail. For example:

- Concept – description of concept.
- Concept 2 – description of concept 2.

6.3.7 Italics and Bold

Use italics by applying the *variable* character style for typical emphasis, as in all writing.

Use the **Strong Emphasis** character style, which bolds the text, for GUI dialog fields, actions, and literal values to be supplied. For example:

In **Associated Project**, select **DemoProject** and then click **OK**.

In **Component Name,** enter **source**.

Note that tutorials 1-4 have used inline bold formatting in GUI procedures. This is not best practice and will be changed in the future.

Aside from GUI procedures, use bold for extra strong emphasis very rarely (Strong Emphasis character style).

Use the ***FirstEmphasis*** character style for terms used specially in the document, until it truly seems routine and then drop it.

Use ***FirstEmphasis*** for reference document and tutorial titles, like the ***OpenCPI Application Development Guide***, ***OpenCPI Tutorial 1: Component-based Development*** or hardware guides like the ***OpenCPI Avnet MicroZed Getting Started Guide***.

Use the ***FirstEmphasis*** character style when citing OpenCPI documents like the ***OpenCPI Installation Guide***. Use the LibreOffice hyperlink feature to create a link to the referenced document on opencpi.gitlab.io.

6.3.8 Glossary Styles

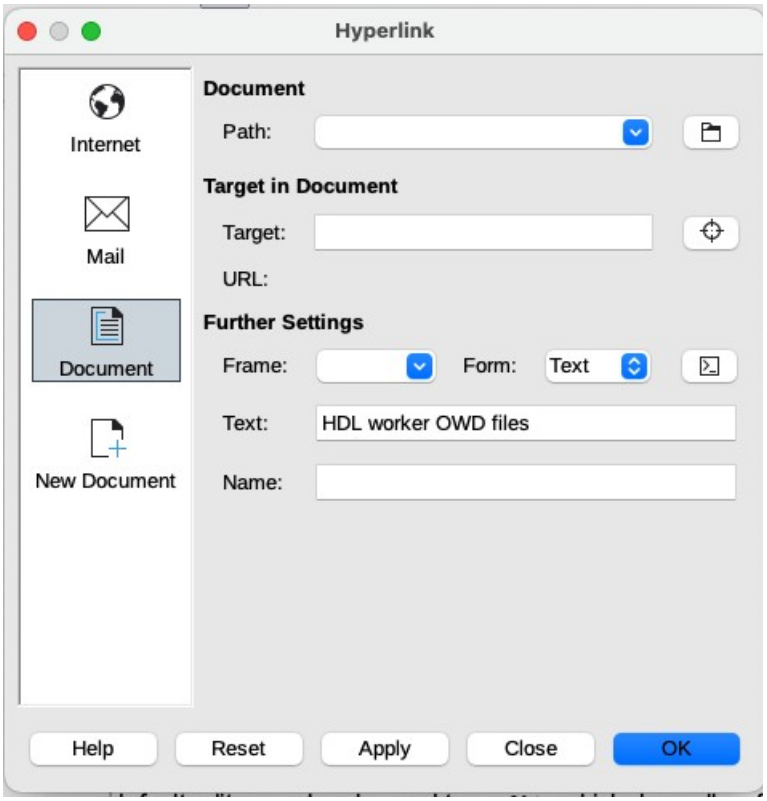
When working with the OpenCPI Glossary, use the Glossary Term paragraph style for the term and the Glossary Definition paragraph style for the definition. Note that these styles are defined in the **`shared/glossary.fodt`** file and not in the template **`shared/OCPI_ODT.ott`**. This is done to keep their definitions from being overwritten by other styles when the **`glossary.fodt`** file is included into other reference documents.

6.4 Creating Hyperlinks

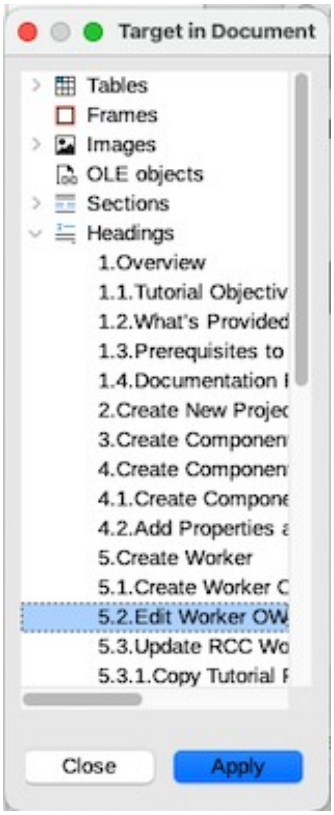
Use the LibreOffice hyperlink feature (the  icon at the top menu bar) to create a link to another section in the same document or to an external document.

To link to another section within the same document:

- Select the text you want to use as a link to the target section, and then click the Hyperlink icon in the top menu bar.
- In the dialog that appears, select Document on the left.



- Click the icon to the right of the **Target:** field. A table of contents is displayed with all the headings in the document.



- Select the heading you want, and then click **Apply** and **Close**.
- Click **Apply** and **Close** in the main Document dialog.

Now you have a link to the selected section heading in the text you selected.

To create a hyperlink to an external document, copy the URL to the document into the Path field in the Document section of the dialog at the top. Note that hyperlinks to other OpenCPI documents on opencpi.gitlab.io should be to the latest release, not to "develop". For example: https://opencpi.gitlab.io/releases/latest/ to find the documentation for the latest released OpenCPI version.

6.5 Using Conditional Sections in OpenCPI Tutorials

The OpenCPI tutorials use conditional sections for their GUI content so that a non-GUI version of the tutorial can be created for command-line interface (CLI) users who do not want the GUI information. The next sections describe how to use them.

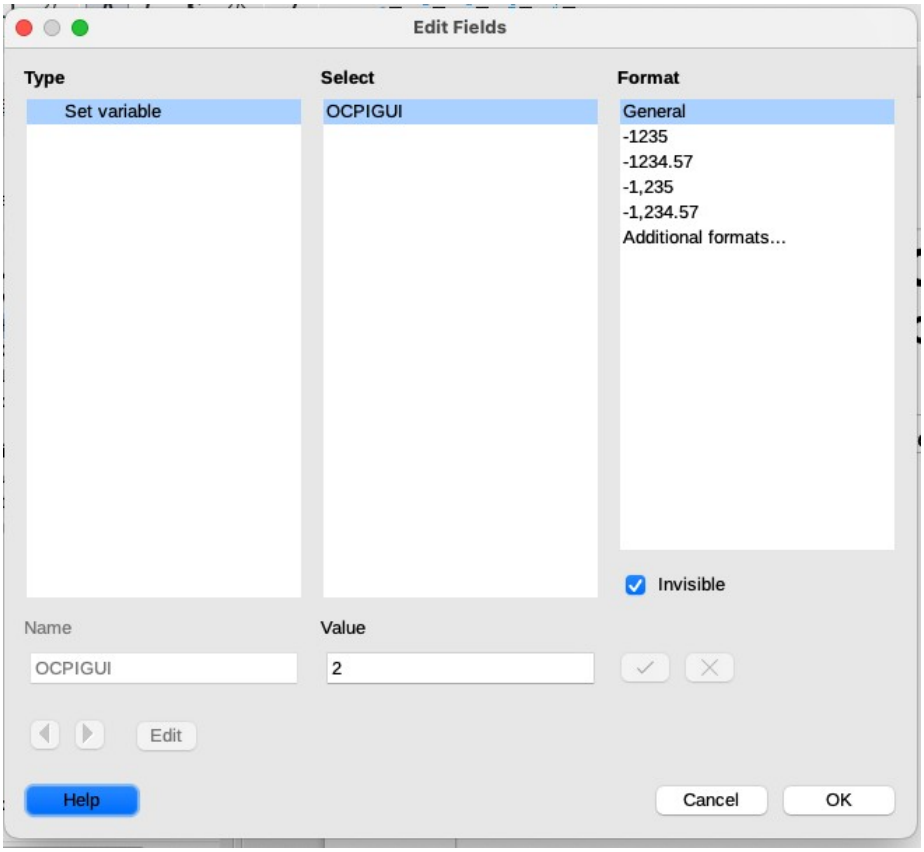
6.5.1 Showing/Hiding Conditional Sections in Tutorials

There is a variable named “OCPIGUI” that is accessible on the title page of each OpenCPI tutorial. It controls whether both the GUI conditional text and CLI main text are displayed (the default), or whether the GUI conditional text is hidden and only the CLI text is displayed.

To test whether your GUI conditional text disappears when the variable that controls it is set to hide it, you need to change the variable that exists on the title page.

- Go to the title page. You’ll see a gray rectangle right after the title text “OpenCPI”. (If you don't see it, in the LibreOffice menu, select **View** > **Formatting Marks** > **Field Shadings**). Right click on it and select **Edit Fields…**

You’ll see the following dialog:



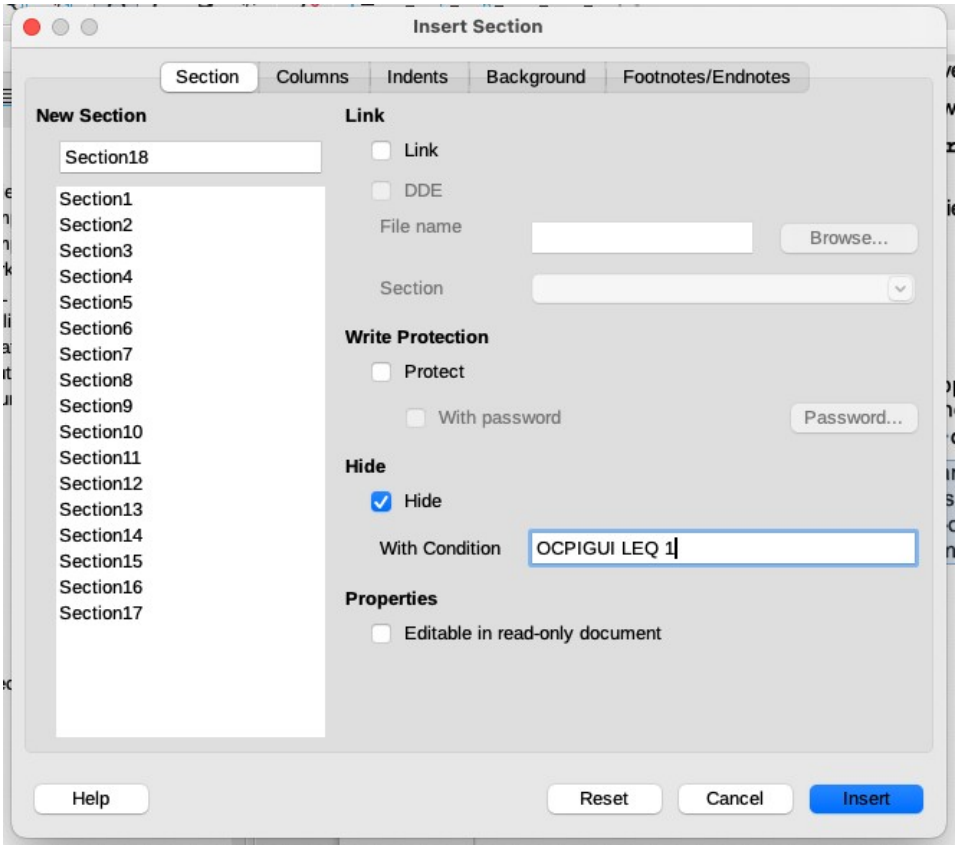
- Change the number in the **Value** field from “2” to “1”. This action hides the GUI conditional texts, so you’ll only see the CLI text. Click **OK**.
- Don’t forget to change it back to “2” when you finish with a tutorial.

6.5.2 Creating a Conditional GUI Section in a Tutorial

GUI instructions in OpenCPI tutorials are set up as “conditional sections” in the tutorial source file so that they can be hidden to create a CLI-only version of the tutorial. When you add a new GUI procedure to a tutorial, you need to enclose it in a conditional section and define the condition with the “OCPIGUI” conditional variable.

To create a conditional section in a tutorial for describing an OpenCPI GUI procedure:

- Go to where you want your section to appear. Select **Insert** > **Section**.
- LO automatically assigns a name, (in the screenshot below, “Section18”), to your section.
- Check **Hide**.
- In **With Condition**, enter **OCPIGUI LEQ 1**.
- Click **Insert**.



- Enter your conditional text into the section between the upper and lower lines.

Make sure there aren't any empty paragraphs (like what's shown in the figure above), when you're finished creating your conditional text.

Alternatively, you can write your conditional GUI procedure first, and then select it and use **Insert** > **Section** as described above to enclose it in a conditional section.

6.5.3 Redefining the "OCPIGUI" GUI Conditional Variable in a Tutorial

The "OCPIGUI" GUI conditional section variable is defined in every OpenCPI tutorial and should not be changed or deleted. However, if you need to redefine the "OCPIGUI" GUI conditional variable in a tutorial for some reason, here is the procedure to do so:

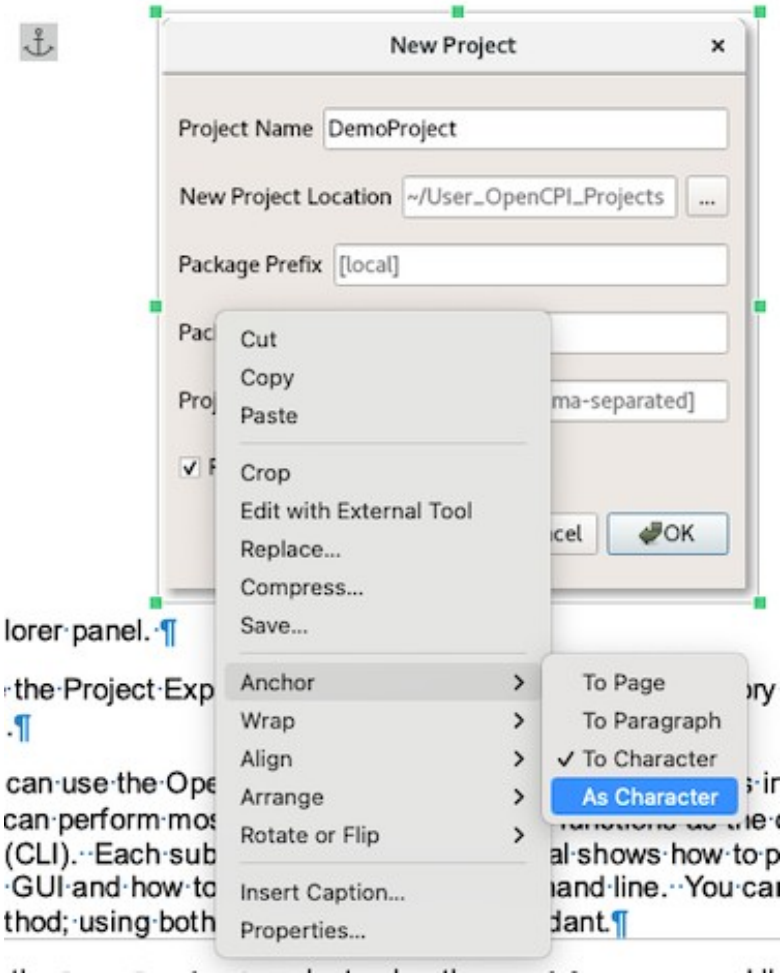
- Go to the top of the document and put the cursor after the "I" in "OpenCPI".
- Select **Insert** > **Field** > **More Fields**.
- Click the Variable tab and select **Set Variable**.
- In **Name**, enter **OCPIGUI**.

- In **Value**, enter **2**.
- In **Format**, select **General**.
- Check **Invisible** so that the value of the variable isn’t displayed in the text.

6.6 Adding a Screenshot to an OpenCPI Tutorial

To add a GUI screenshot to an OpenCPI tutorial:

- From the Styles pane, create an empty Diagram paragraph style where you want to insert the screenshot.
- Copy (Ctrl/C) and paste (Ctrl/V) the screenshot (for example, from a screen capture program, an image editor like Preview on a Mac computer or directly from the Clipboard) into the Diagram paragraph.
- With the screenshot image selected, (green markers on corners), use the arrow tool at an edge of the image (usually bottom right) to adjust the size as necessary. Do this before the following next step.
- With the image selected (right click on it to select), select **Anchor** > **As Character**.



- Delete any empty paragraphs before and after the image.
- You don’t need to add figure captions to screenshots in tutorial procedures.

6.7 Adding a PowerPoint Slide as a Diagram

This chapter describes how to export a slide in PowerPoint (PPT) into a LibreOffice (LO) Draw document and then into a LibreOffice Writer document as a diagram.

- In LO: Make a pair of empty paragraphs: style Diagram over style Caption.
- In PPT: In **Save as**, select **OpenDocument presentation (.odp)**. It should default to the current slide only.
- In LO: Open the ODP file. LO opens the file in LO Draw.
- In LO-Draw: click to select drawing object to get green or blue corners.
- In LO-Draw: click the crop button:

  (older LO: has scissors-with-paper icon with orange handles, not scissors-alone with red handle, which is CUT)

  (newer LO: has dashed rectangle with L angle inside, results in blue corners)
- In LO-Draw: crop by dragging crop edges, tightly since we want doc styles to specify spacing, not white space in diagram.
- In LO-Draw: copy and then switch to the LO Writer doc window.
- In LO-Writer: in the document, place the text cursor AND MOUSE CURSOR EXACTLY OVER TEXT CURSOR into the (empty) Diagram paragraph and paste.
- In LO-Writer: right-click on the pasted diagram (which should already be selected with green corners). Select **Anchor** > **As character**.
- Click on the diagram to select it as a diagram (with green/blue corners).
- Scale the diagram by holding down the shift key to scale in X and Y at the same time proportionally.
- Insert the text “Figure” in the Caption paragraph under the Diagram paragraph. Use **Insert** > **Field** > **More-fields…** and then select type: "number range", select: Diagram, Format: Arabic 1 2 3, click **Insert**, and then finish the diagram title.
- When you re-edit the PowerPoint presentation, just do another save as ODP, and just use the same file name (if the whole pptx file full of diagrams is **`abc123.pptx`**, Just continue to use **`abc123.odp`**, since it is just a temp file anyway.
- Then, since you have the diagrams open in LO-Draw anyway (don't close them after copy-paste into the Writer document), you can just use **File** > **Reload** in LO Writer to bring in the diagram and re-crop.

- Then, copying the cropped diagram, you can paste over the selected diagram and not have to redo any of the other steps. It even remembers the initial scaling.

6.8 Adding a Timing Diagram Created with WaveDrom

WaveDrom is a tool for creating timing diagrams. To add a timing diagram created in WaveDrom as an SVG file to a reference document or tutorial:

- Create a pair of empty paragraphs: paragraph style Diagram over paragraph style Caption.
- In the document, place the text cursor AND THE MOUSE CURSOR EXACTLY OVER THE TEXT CURSOR into the (empty) Diagram paragraph.
- Use **Insert** > **Image** to open and insert the SVG file.
- Right-click on the inserted diagram (which should already be be selected with green/blue corners) and then select **Anchor** > **As character**.
- Right-click on the diagram and select **Crop** to crop any white space around it.
- Hit escape to exit crop mode and click on the diagram again to select it.
- Drag a corner with no modifiers to maintain proportions while resizing as needed.
- Insert text in the Caption paragraph after the Diagram paragraph, using **Insert** > **Field** > **More fields...** to add the diagram number and then finish adding the diagram title.

See this document for information on how to use WaveDrom.

6.9 Creating PDFs for Document Review

This section describes how to generate:

- PDFs with bookmarks for all kinds of OpenCPI reference documents
- CLI-only PDFs for OpenCPI tutorials

Note that the OpenCPI document generator (**`build_pages.py`**) run by the OpenCPI release engineer is the tool that generates PDFs for the documents on the opencpi.gitlab.io website. You do **not** need to generate final PDFs when you commit and push your final reference document or tutorial to Gitlab.

6.9.1 Generating a PDF with Bookmarks

To generate a PDF with bookmarks:

- Select **File** > **Export as** > **Export as PDF**.
- Check **Tagged PDF (add document structure)** and **Export bookmarks**.
- Check **view PDF after export** if you want to have the PDF open automatically.
- Click **Export**.
- Select the filename and location (if you don't like the defaults) and then click **Save**.

To check the PDF document you just created, open it in a viewer that displays and supports bookmarks (Acrobat Reader, Evince, Okular, …).

6.9.2 Creating CLI-only Tutorial PDFs

You may want to create CLI-only PDF versions of the tutorial for review purposes and especially to check that the text flow between paragraphs still makes sense and there aren't any extra spaces between headings or paragraphs when the GUI parts are hidden.

To create a PDF of a tutorial that just shows the CLI:

- Follow the instructions in "Showing/Hiding Conditional Sections in Tutorials" to hide the GUI text.
- Follow the instructions in "Generating a PDF with Bookmarks".

6.10 Working with the OpenCPI Document Template

The documents in **`$OCPI_ROOT_DIR/doc/reference`** and **`$OCPI_ROOT_DIR/doc/tutorials`** use the template named **`shared/OCPI_ODT.ott`**. Unfortunately, the LibreOffice Writer GUI does not let you specify a relative pathname for a template directly. So, as mentioned in "Creating a New LibreOffice Document", new documents are best copied from older documents that already have the right template association. You can see the current template in a **`.fodt`** file by using the command:

```
sed -n 's|.*<meta:template[^>]* xlink:href="\([^"]*\)".*|\1|p' \
<filename>.fodt
```

If it does not print:

```
shared/OCPI_ODT.ott
```

the template association is wrong.

When the template is changed (i.e. by editing it), the next time you load a document the LibreOffice GUI will ask if you want to update with the changed template or "keep old styles". **DO NOT** select "keep old styles" since this action permanently disassociates the document from the template: to re-associate the document with the template, you must create a new empty file from the template and then copy/paste the whole disassociated document into the new empty one. See the LibreOffice documentation for details about working with LO templates.

7 Preparing a LibreOffice Document for Merge

There are two compatibility steps that need to be followed for all OpenCPI LibreOffice documents (reference documents, tutorials, and briefings) that have been created or edited with versions of LibreOffice that are newer than version 6.x:

1. Save the newer-version LibreOffice document in LibreOffice 6.x that is running on a Rocky8 system.
2. Build the opencpi.gitlab.io website locally and then view the results to ensure the generated PDF displays correctly.

7.1 Saving Newer-Version LibreOffice Documents in LibreOffice Version 6.x

The OpenCPI document publishing tool (**`build_pages.py`**) uses LibreOffice 6.x, which is the default version that is available on the Rocky8 system. This version of LibreOffice cannot currently process images in FODT or FODP files that have been saved with versions of LibreOffice that are newer than 6.x.

If you are using a version of LibreOffice that is newer than 6.x, you must save your newer-version document in the format that the document publisher expects before you commit and push your final document to the main "develop" branch on the Gitlab server. Here is the procedure:

- Use **Save as** to save the open file in the newer LO version into ODT (**`*.odt`**) format (use ODP (**`*.odp`**) for briefings). Close the file.
- On a Rocky8 system that has LO 6.x installed (for example, a virtual Rocky8 machine) and that you have set up so that your newer-version LO files in your local work branch are visible to it, open your saved ODT or ODP file with LO 6.x.
- Use **Save as** and select to save the file in FODT (**`*.fodt`**) format (use ODP (**`*.fodp`**) for briefings). LO 6.x on the Rocky8 system will prompt you that there is already a FODT/FODP file in your working branch and do you want to replace it. Select "yes".

Now the FODT file (or FODP file, for briefings) in your working branch has been saved in the LO 6.x FODT/FODP format that the document publisher can handle.

7.2 Building and Viewing a Local Copy of the OpenCPI Website

The next compatibility test is to use the document publisher to build a local copy of the opencpi.gitlab.io website and then view the results to make sure the PDF generated for the document displays correctly. To do this:

- On a Rocky8 system (for example, a virtual Rocky8 machine) that you have set up so that the files in your local work branch are visible to it, change directory to the OpenCPI root (top level) of your local work branch and run the command:

```
scripts/install-opencpi-docs.sh --no-build
```

- Now build the website with the command:

```
doc/build-pages.py HEAD
```

The documents for the website are generated in a **`.public/`** directory in the OpenCPI root directory. To view them, navigate into the **`.public`** directory and then open the file "**`index.html`**" with FireFox or another available browser. This action brings up the "website" document tree, as if you were looking at opencpi.gitlab.io.

Click on your document in the website documentation tree to open the PDF. Examine it for formatting problems, especially missing or incorrectly formatted images.

Note that some links in the local generated copy of the website behave differently in a file context relative to how they behave when you are interacting with an actual web server. Specifically, clicking on some hyperlinks in the local website copy results in a directory listing rather than the possibly-expected automatic opening of the "**`index.html`**" page in that directory.

8 Working with OpenCPI Man Pages

OpenCPI provides manual ("man") pages for all OpenCPI command-line tools. Currently, all existing man pages are in the Linux "man1" category. Man page sources are located in the **`$OCPI_ROOT_DIR/doc/man/src`** directory and have the name format:

`<manpage-name>.1.txt`

The top-level man page is **`opencpi.1.txt`**. References to all the other man pages are contained in this man page.

The **`ocpidev`** tool has per-noun and per-verb "sub"-man pages. The format is:

- **`ocpidev-<noun>`**, like **`ocpidev-project`**, **`ocpidev-worker`**, **`ocpidev-application`**, ...
- **`ocpidev-<verb>`**, like **`ocpidev-create`**, **`ocpidev-delete`**, ...

`ocpidev` is the only tool that currently has multiple sub man pages.

Man pages are formatted in the **`asciidoc3`** markup language (see the ***asciidoc3 User Guide*** for details) following the Linux manual page conventions and the relevant OpenCPI documentation conventions that can be applied using **`asciidoc3`** markup.

The OpenCPI document publisher generates HTML versions of the source man pages for publication on the opencpi.gitlab.io website.

To work with OpenCPI man pages:

- Set up the man page development environment
- Create new man page source files or edit existing ones following the conventions for OpenCPI documents and man page markup
- Generate troff and HTML versions of new or updated man page source files and check that they are correctly formatted and display properly when viewed with the "man" command and a Web browser on a Linux system
- Delete all generated files and only "git commit"/"git push" new and/or updated man page source files to the OpenCPI repository
- Prepare a merge request for the committed/pushed man page sources

The next sections describe these steps in detail.

8.1 Setting up the Man Page Development Environment

To work on OpenCPI man pages, **`asciidoc3`** needs to be installed and a Linux system that has access to the OpenCPI man page files needs to be available. Installation instructions for **`asciidoc3`** are provided here. The asciidoc3 quickstart describes **`asciidoc3`** dependencies and lists their URLs.

As of this writing, the man page development environment in use by OpenCPI maintainers consists of an Ubuntu machine and a Rocky8 virtual machine created with a VMWare® Workstation license. File sharing between the Windows host and the Linux VM is done via VMWare Workstation features. Testing has also been performed on the Linux VM. The current man page development is based on the **`asciidoc3-3.2.0.tar`** installation using the local "tarball" installation instead of the system-wide "installscript" installation; see the ***asciidoc3 User Guide*** for details. Using the local installation instead of the system-wide one requires typing the command:

```
python3 a2x3.py
```

to generate a man page and using absolute pathnames in commands. See the sections that describe how to generate troff- and HTML-formatted man page for examples.

8.2 Creating a Man Page with asciidoc3

All the existing man pages are in the "man1" category (see the Linux manual page conventions) and are located in the **$OCPI_ROOT_DIR/doc/man/src** directory. They have the name format:

<manpage-name>.1.txt

asciidoc3 requires the ***.txt** suffix.

Man page development is generally performed in the **doc/man/src** directory of your local work branch. To create a new man page:

- Use a text editor to create the file **<manpage-name>.1.txt** in the **doc/man/src** directory in your local work branch.
- Follow the conventions for man page headings described in the Linux manual page conventions and used in the existing man pages.
- Use the markup language for man page headings and text described in the ***asciidoc3 User Guide*** and used in the existing man pages.
- Follow the OpenCPI documentation conventions for text that is relevant to man pages

8.3 Generating a troff-format Man Page with asciidoc3

Generating a troff-formatted man page tests that your man page source formats correctly for a **man** command. A troff version can also be useful during man page development for generating PDF for team review. To create a troff-formatted man page with the development environment previously described, the command is:

```
python3 <asciidoc3-install-path>/a2x3.py --doctype manpage --format
manpage \
<opencpi-repo-path>/doc/man/src/<manpage-name>.1.txt
```

This operation generates the file **<manpage-name>.1** in the **doc/man/src** directory in your work branch.

To create a PDF from the generated man page on the development environment previously described, you can use the **man** command to create a PDF from the generated troff-formatted man page and then use (Mac computer) Preview to view it. The command is:

```
man -t <opencpi-repo-path>/doc/man/src/<manpage-name>.1 | open \
-fa "Preview"
```

To test that a troff-formatted man page displays with the **man** command executed on a Linux system in the development environment previously described, in a terminal window on the Linux system, run the **man** command on a **<manpage-name>.1** troff-format man page created in your work branch. The formatted man page should display on the Linux system.

To generate a PDF for a man page on a Linux system in the development environment previously described, use the following command in a terminal window on the Linux system:

```
rubber --pdf <manpage-name>.1.txt
```

You should get a PDF file (and some other files, like **.aux**). (Note that the **rubber** command may need to be manually installed on the Linux system before executing the command.)

8.4 Generating an HTML-format Man Page with asciidoc3

Generating an HTML-format man page tests that your man page source formats correctly for HTML presentation on the opencpi.gitlab.io website. To generate an HTML-formatted man page with the development environment previously described, the command is:

```
python3 <asciidoc3-install-path>/a2x3.py --doctype manpage -f \
xhtml <opencpi-repo-path>/doc/man/src/<manpage-name>.1.txt
```

This operation generates the file **`<manpage-name>.1.html`** in the **`doc/man/src`** directory in your work branch that you can open with a browser (by default, Safari in the development environment previously described). It also generates a DocBook stylesheet named **`docbook-xsl.css`** in the **`/doc/man/src`** directory (that is, the directory in which you run the XHTML generation command). You need to have this file present for the HTML pages to render properly during your local development process. However, this file should *not* be committed to the OpenCPI repository on the Gitlab server since this stylesheet is already provided by the OpenCPI document publisher.

To test an HTML-formatted version of the man page on a Linux system in the development environment previously described, in a terminal window on the Linux system, run the command:

```
firefox <manpage-name>.1.html
```

The FireFox browser should open the HTML version on the Linux system. (Note that the FireFox browser may need to be manually installed on the Linux system before executing the command.)

8.5 Preparing to Commit Man Pages

Doing man page development in **`doc/man/src`** in your work branch generates ***`.1`** files, ***.`html`** files, and the **`docbook-xsl.css`** style sheet. Be sure to delete all of these generated files from your work branch when you are ready to commit your new and/or updated ***`.1.tx`**t source files. Generated files and the locally-created DocBook CSS style sheet should *not* be committed and pushed!

9 Documenting OpenCPI Assets

The **ocpidev** development tool supports a Sphinx-based asset documentation system that allows OpenCPI developers to document their assets as part of the OpenCPI development process. This system is an integral part of the **ocpidev**-based asset development workflow described in the OpenCPI reference guides and demonstrated in the OpenCPI tutorials, such that:

- Creating an OpenCPI asset creates both the skeleton files and the template documentation files relating to the asset. The documentation files are created in the same directory as the asset files and are named the same as the asset XML files but with the suffix ***.rst** rather than **.xml**.
- Building an OpenCPI asset builds both the asset files and the documentation files unless you use an option to specify that only assets file or only documentation files should be built. The build process creates readable documents locally to where the build command is issued, and different levels do not conflict (project, library, component, worker). For example, building the documentation in a component library creates a separate document that includes the library's worker documents without affecting any worker documents built in an individual worker's directory.
- Cleaning an OpenCPI asset removes the asset's generated files and the built/generated documentation files.
- Deleting an OpenCPI asset deletes both the related asset files and the built/generated documentation files.

The **ocpidev(1)** man page provides general syntax and usage information about the tool, while the **ocpidev-create(1)**, **ocpidev-build(1)**, **ocpidev-clean(1)**, and **ocpidev-delete(1)** man pages provide syntax and usage information specific to these **ocpidev** operations (called “verbs”, in **ocpidev** syntax). There are also asset-specific (aka “noun”) man pages that provide information about using **ocpidev** with a specific asset, like projects, applications, components, workers, protocols, and so on.

9.1 General Information

This section provides information that applies to documenting all OpenCPI asset types.

9.1.1 Documentation Development Workflow

The general procedure is:

1. Create the asset with the **`ocpidev create`** command. This step automatically creates a template document in the same directory as the created asset. For example, when you create an HDL worker, **`ocpidev`** creates a template for documenting this worker with the suffix **`*.rst`** in the same directory where it creates the HDL worker's OpenCPI Worker Description (OWD) and source files.
2. Fill out the template document with a text editor, following the skeleton instructions provided in the template document and the hints provided in this chapter. The template documents created by **`ocpidev`** are formatted in reStructuredText (reST) markup language (referred to as "RST" in OpenCPI documentation) and use a combination of standard Sphinx directives and OpenCPI documentation-specific Sphinx directives. Use this markup and these directives as necessary when filling out your document.
3. Build the asset with the **`ocpidev build`** command. In addition to generating runtime files that relate to the asset, this step generates HTML-formatted output from the documentation source files and places it in the subdirectory **`gen/doc/*.html`** in the location where the build command is issued (after using the --**`directory`** option, if specified). To build just the documentation, use the --**`doc-only`** option.
4. Open the file **`gen/doc/index.html`** with a browser to view the generated HTML documentation. **Note**: if the documentation build fails, the **`gen/doc/`** subdirectory is not created. In this case, examine the Sphinx-related error messages generated by the **`ocpidev build`** command to identify and fix the problem in the document source file(s).
5. Fix any errors in the document source file(s) that are causing problems in the HTML output and then build and check the HTML output again. Do this step until the HTML output is error-free.

For any directory-based asset, you can build the documentation locally to that asset and then view the results in **`gen/doc/*.html`**. At higher levels of the projects (the project, libraries, or library level), the build is recursive, with all the results placed in the **`gen/doc/`** directory for that asset. For example, building the documentation for a project produces the full HTML tree of documentation (project, library, component, worker, …) under the project's **`gen/doc/`** directory, but does not affect (rebuild or otherwise touch) the local documentation built in the lower-level directories.

9.1.2 Standard Sphinx Directives used in OpenCPI Asset Documentation

The following table lists the Sphinx directives most frequently used in the asset documentation templates provided by the OpenCPI asset documentation system.

Table 1: Standard Sphinx Directives used in OpenCPI Asset Documentation

Directive	Used in
`.. code-block::`	Code examples
`.. figure::`	Creating diagrams in any asset document
`.. literalinclude::`	Code examples
`.. math::`	Creating equations in any asset document
`.. note::`	Text of any asset document
`.. toctree::`	Project asset documents, document structure files for component libraries, primitive libraries, ...
`.. warning::`	Text of any asset document

See the Directives section of the Sphinx documentation for descriptions of these directives.

9.1.3 OpenCPI-specific Directives

The OpenCPI asset documentation system provides a set of OpenCPI-specific directives used in the asset documentation templates that automate some of the asset documentation tasks.

Table 2: OpenCPI Sphinx Directives

Directive	Used in
`.. ocpi_documentation_implementation::`	Component asset documents
`.. ocpi_documentation_include::`	Any asset document
`.. ocpi_documentation_ports::`	Component asset documents
`.. ocpi_documentation_properties::`	Component asset documents
`.. ocpi_documentation_test_platform::`	Component asset documents
`.. ocpi_documentation_worker::`	Worker asset documents

See the referenced asset document sections for descriptions of these directives.

9.1.4 Hints That Apply to Documenting all OpenCPI Asset Types

Here are some general hints for working with the asset documentation system:

- OpenCPI assets are file based or directory based. When directory-based, the name of the asset's XML file (in that directory) is the directory name, with periods replaced by dashes and the **`.xml`** suffix added. Projects, which are also directory-based, are a special case where the XML file describing the project is always named **`Project.xml`**, regardless of the name of the project's directory.

- Make sure the properties and ports defined in your component specifications (OCSes) and worker definitions (OWDs) include **`<Description>`** XML elements. The asset documentation system automatically generates the properties and ports documentation from these elements. If they're not there, the system won't generate any documentation for them, and you'll have to add the descriptions by hand.

- Make sure that any subsidiary files that are included into an asset document, whether or not they are local, are named with the suffix **`.inc`** , not **`.rst`**. Using the **`*.inc`** suffix for subsidiary included files is currently necessary to prevent them from being mistakenly generated by the asset documentation system during a documentation build operation.

- The preferred graphics format for figures and illustrations is SVG. Use an external tool to create your images that can save files in SVG. For example, LibreOffice Draw is a free drawing tool that supports SVG format and is included with the LibreOffice package.

- Use title case in section headings and in figure and equation captions as described in the OpenCPI documentation conventions in this guide. Examples of title case are: "Running the Application", "Worker Ports", "MyComponent Block Diagram".

9.2 Documenting OpenCPI Asset Types

This section describes how to document the assets that can be created with the **`ocpidev create`** command.

9.2.1 Documenting an OpenCPI Application

The name of the documentation file that describes an OpenCPI application is the application's XML file name with the **`.rst`** suffix instead of **`.xml`**. The "**`ocpidev create application`**" command creates this file in the **`applications/`** subdirectory of an OpenCPI project in either in the same directory as the application it describes or directly in the **`applications/`** directory if the application XML does not have its own directory.

Use the **`<application-name>.rst`** file to document your application, filling in the sections in the suggested outline in the template with the information about your application, adding new sections as necessary and/or deleting any section headings you aren't going to use.

An application document should provide:

- One or two introductory paragraphs that describe the application's purpose and what it does
- Known limitations as to which OpenCPI platforms it can use, either inclusively or exclusively
- Things that must be in place before the application can be run, like hardware and build prerequisites
- How to build and run the application or whether it can simply be built and run with **`ocpidev`**
- How to verify its success or failure
- Troubleshooting information and/or a description of any known issues

9.2.2 Documenting an OpenCPI Component

The name of the documentation file that describes an OpenCPI component is the component's XML file name with the **`.rst`** suffix instead of **`.xml`**. The "**`ocpidev create component`**" command creates this file within a component library in the same directory as the component specification. This file is the main template for documenting the component.

The "**`ocpidev create component`**" command also creates a template OpenCPI Application Specification (OAS) file named **`example_app.xml`** in the same directory as the component specification and the main component document. This example

application file is automatically included verbatim into the skeleton file for the main component document. You can either customize it to your component's features or remove the reference to it from the main document and delete the file from the component's directory.

The general workflow for documenting an OpenCPI component is to:

- Add the details about your component to the main template.
- (optional) Customize the example application template to illustrate how your component is to be used.
- Create worker documents for all the HDL and/or RCC workers that use the component specification. The component document will contain links to these worker documents via the "implementation" OpenCPI directive.

9.2.2.1 Write the Main Component Document

This section provides some hints for filling out the main documentation template file **`<component-name>-comp.rst`** in addition to what's given in the template.

Add a Top-Level Description

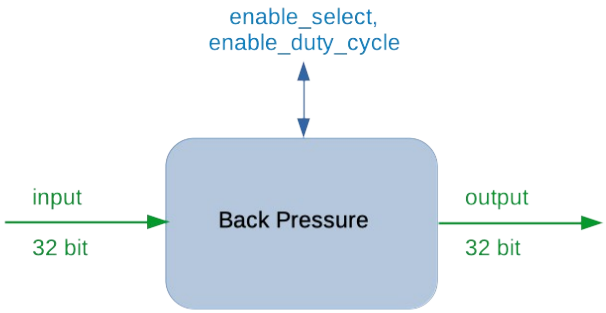
The template provides a placeholder for adding a one-line description that summarizes the component's purpose. It should be an incomplete sentence, like a "man" page description. Here's the one-sentence description given in the **`backpressure`** component RST document in the component library of the OpenCPI built-in project **`ocpi.core`**:

```
Emulates the back pressure that is present in a system.
```

The descriptions used in the other **`ocpi.core`** component RST documents provide additional examples.

Create a Component Block Diagram Figure

Use the "Function" section in the template to describe the component's purpose and operation. It is recommended to provide a block diagram of the component's data flow, ports, and properties as part of the component's description in the "Function" section. Use different colors for text and/or arrows to distinguish between the property interface and the data interface. The block diagram for the component should only show the component properties and ports, not the worker-specific ones. Here is an example block diagram for the **`backpressure`** component:



In this example:

- The component properties text and arrow are blue
- I/O data text and arrows are green
- The proper name for the component is used (i.e., “Back Pressure” instead of “backpressure”)

To create the block diagram and any other figures you want to include that describe the component, use an external tool that supports saving to SVG format, like LibreOffice Draw. Save your figure source and SVG output in the **`<component-name>.comp`** directory.

The template provides a sample block diagram RST markup with the name of your component inserted. To this template, add:

- The path to your block diagram SVG file.
- The title for your caption, in title case. For example:

```
MyComponent Block Diagram
```

- (optional) An alternate caption title. For example:

```
:alt: Block Diagram of MyComponent Properties and Ports
```

Follow the same conventions for any other figures you include.

Describe Any Component Equations

A component's description may benefit from documenting the equations that define the component's required functionality. You use the standard Sphinx “math” directive to create equations. The template provides a sample equation markup that includes a bulleted list for describing its arguments, with the name of your component inserted. To this template:

- Add the markup for your equation

- (optional) Edit the bulleted list with the markup for your arguments and a description of what they are, or delete the list if you don't need it.

See the RST documents for the DSP components in the Software Defined Radio (SDR) Component Library OSP (**`ocpi.comp.sdr`**) for examples of equation and argument description markup; for example, the Carrier/Mixer DSP component document.

Add Subsections to "Function" As Necessary

A component's description may benefit from having topics organized into subsections within the "Function" section. See the File Write component document in the component library for the OpenCPI built-in "core" project (**`ocpi.core`**) for an example of how to organize a "Function" section into different subsections in RST markup.

Describe the Component's Properties

The "Properties" section of the template contains the OpenCPI-specific **`ocpi_documentation_properties`** directive that automatically populates it with bulleted lists of property information. Its usage is:

```
.. ocpi_documentation_properties::
```

No arguments are needed. The options that can be used are:

- **`component_spec`** - allows overriding the automatically-determined component specification path. When used, the file path of the component specification relative to the current documentation file must be provided. Unless this option is used, the directive searches the library that contains the component for a component specification based on the current directory's name. A component's spec is normally found in a component's directory (and thus in the same directory as the component asset document) with a file name based on the directory's name, with periods replaced by dashes and a **`.xml`** suffix added. For backward compatibility, the directive also searches the **`specs`** directory of the encompassing component library for the component's spec file. See the hints section for more information.

- Additional text to describe each property, added in the "body" of the directive, which consists of a blank line underneath the directive followed by one or more text lines indented three spaces to line up with the first character of the directive (see the example below). This text must be the name of the property followed by a colon symbol (:), with each property text on a new line. (Additional property text is not passed as an option of the directive, as directive option names must be known at the time of writing the directive, which for property names is not possible.)

If the property definitions in the component spec contain descriptions (using the **`<Description>`** element), there is nothing you need to do here. The directive uses the description supplied in the element (see the section "Description Attribute/Element of Property (and Member) Elements" in the ***OpenCPI Component Development Guide***

for details). If not, you'll need to add descriptions of your component's properties by hand. The template provides placeholders below the "properties" directive for adding additional text to property descriptions. Use these placeholders to add your property descriptions. Be sure to keep each description on a single line (no line breaks).

You can also use the placeholder to provide property property information that's additional to what's provided in the component spec. Change the **`property_name`** or string in the placeholder to the name of your property and then provide the additional information as the text value.

Here is an example from the core project's **`backpressure`** component's document, where two property descriptions under the directive are added by hand:

```
.. ocpi_documentation_properties::

   enable_select: Selects the back pressure scheme to control the
"take" from the upstream worker.  ``true`` uses lfsr-15. ``false``
uses the setting in ``enable_duty_cycle``.

   enable_duty_cycle: Sets the "take" duty cycle. Possible values
are:``1`` = constant (default), ``2`` = toggle (off/on), ``3`` =
1/on, 2/off, ``4`` = 1/on, 3/off.
```

The HTML output looks like this:

Properties

- `enable_select` : Selects the back pressure scheme to control the "take" from the upstream worker. `true` uses lfsr-15. `false` uses the setting in `enable_duty_cycle` .
  - Type: `bool`
  - Access:
    - Parameter: `False`
    - Writable: `False`
    - Initial: `True`
    - Volatile: `False`
  - Default value: `false`
- `enable_duty_cycle` : Sets the "take" duty cycle. Possible values are: `1` = constant (default), `2` = toggle (off/on), `3` = 1/on, 2/off, `4` = 1/on, 3/off.
  - Type: `ushort`
  - Access:
    - Parameter: `False`
    - Writable: `False`
    - Initial: `True`
    - Volatile: `False`
  - Default value: `1`

Describe the Component's Ports

The template's "Ports" section contains the OpenCPI-specific **`ocpi_documentation_ports`** directive that automatically populates the section with bulleted lists of port information taken from the component spec under the headings "Input" and "Output" when the RST file is built. Its usage is:

```
.. ocpi_documentation_ports::
```

No arguments are needed. The options that can be used with the directive are:

- **component_spec** - allows overriding the automatically-determined component specification path. When used, the file path of the component specification relative to the current documentation file must be provided. Unless this option is used, the directive searches the library that contains the component for a component specification based on the current directory's name. A component's spec is normally found in a component's directory (and thus in the same directory as the component asset document) with a file name based on the directory's name, with periods replaced by dashes and an xml suffix added. For backward compatibility, the directive also searches the **specs** directory of the encompassing component library for the component's spec file. See the hints section for more information.

- Additional text to describe each port, added in the "body" of the directive, which consists of a blank line underneath the directive followed by one or more text lines indented three spaces to line up with the first character of the directive (see the example below). This text must be the name of the port followed by a colon symbol (:), with each port text on a new line. (Additional port text is not passed as an option of the directive, as directive option names must be known at the time of writing the directive, which for port names is not possible.)

If the port definitions in your component spec do not contain description attributes or elements, you'll need to add them by hand. The template provides placeholder lines for one input port named **input** and one output port named **output**:

```
.. ocpi_documentation_ports::

   input: Primary input samples port.
   output: Primary output samples port.
```

Edit these lines with the name your component specification uses for the input and output ports and a brief description. Make sure you keep these lines aligned with the directive.

If you have additional descriptive information that's not in your description attributes and elements, you can add it below the "ports" directive, separated by a space and not indented.

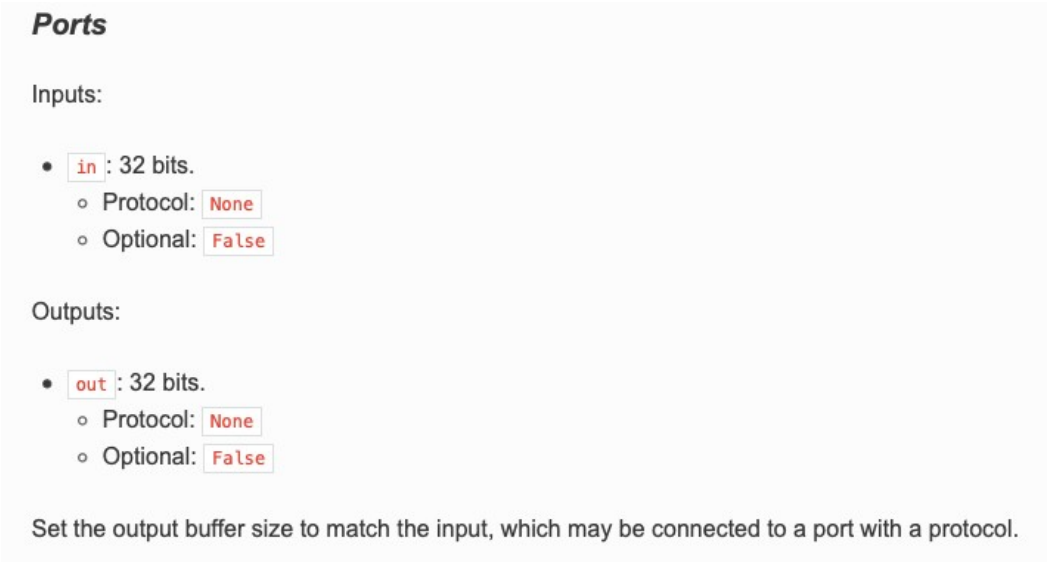
Here is an example from the core project's **backpressure** component's document, where the port descriptions and additional text are added by hand:

```
.. ocpi_documentation_ports::

   in: 32 bits.
   out: 32 bits.

Set the output buffer size to match the input, which may be
connected to a port with a protocol.
```

The HTML output looks like this:



Ports

Inputs:

- `in`: 32 bits.
  - Protocol: `None`
  - Optional: `False`

Outputs:

- `out`: 32 bits.
  - Protocol: `None`
  - Optional: `False`

Set the output buffer size to match the input, which may be connected to a port with a protocol.

Describe the Component Test Suite (Optional)

In this section, you can manually add a description of your component test suite's purpose, function, test cases and expected results. The component asset document template currently supplies the OpenCPI-specific directive **`ocpi_documentation_test_platforms`** to list the platforms on which the tests in the component test suite have been run. Its usage is:

```
.. ocpi_documentation_test_platforms::
```

By default, no arguments are needed. The directive generates a list of platforms on which tests have been run or the output "Tested platforms: None" if no tests have been run. The directive is optional and can be deleted if not used.

In a future release, an RST template for documenting a component test suite asset with OpenCPI-specific directives for automating the generation of test suite details will be created when the component test suite is created with ocpidev and co-located with the test suite assets in the component library. A directive that links the main component document to these component test suite document will be provided, similar to how the "implementation" directive works to link to worker asset documents.

9.2.2.2 Customize the Example Application Template (Optional)

The purpose of the "Example Application" section is to provide a meaningful example of how your component is used. The **`example_app.xml`** template is a simple OpenCPI Application Specification (OAS) for illustrating how your component can be used. The template application uses the built-in OpenCPI core components **`file_read`** and **`file_write`** with your component connected between the two. To this template, you add the properties and ports defined for your component and any additional XML that's

required to demonstrate how it works. If the application template is too simple for your component, you can provide your own example OAS.

9.2.2.3 Reference the Worker Documentation

OpenCPI workers are generally developed after an OpenCPI component spec (and thus its asset document) exists. The results of worker authoring include the worker's directory and the worker asset's documentation. The “Implementation” section in the main component document template contains the OpenCPI-specific directive **`ocpi_documentation_implementation`** for referencing the documentation for the workers that implement the component.

After worker authoring is complete and the worker documentation exists, the “implementation” directive in the main component doc should be updated to reference the HDL and/or RCC directories that contain the relevant worker documents. The directive's syntax is:

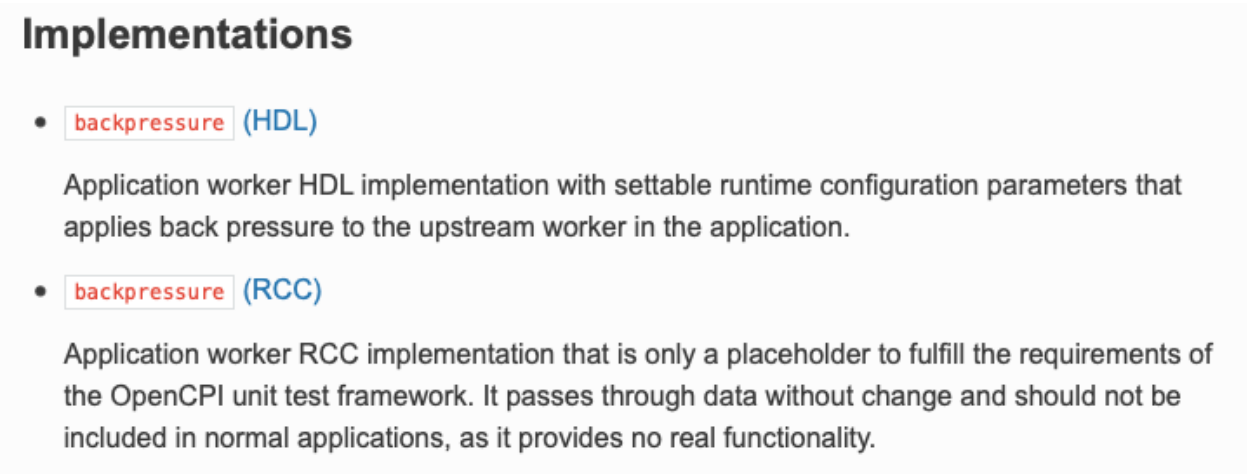
```
.. ocpi_documentation_implementation:: <worker-path> [<worker-path> ...]
```

Each **`<worker-path>`** argument specifies the path to the worker directory that contains the worker to be listed, up to a maximum of 10 worker paths.

Here is an example from the core project's **`backpressure`** component:

```
.. ocpi_documentation_implementations:: ../backpressure.hdl ../backpressure.rcc
```

The HTML output looks like this:



The worker section describes how create a worker's asset documents.

9.2.3 Documenting an OpenCPI Component Library

The name of the documentation file that describes an OpenCPI component library is the component library's XML file name with the **`.rst`** suffix instead of **`.xml`**. The “**`ocpidev create library`**” command creates this file in the **`components/`** subdirectory of an OpenCPI project in the same directory as the component library it describes.

A component library asset document describes the general purpose of the components contained in the library and contains a **toctree** directive that points to the documentation for the components contained in the library. Replace the "Skeleton outline:" line in the **<library-name>.rst** template file with a description of your component library and its scope. The template sets up everything else for you.

Here is an example of an OpenCPI component library asset document:

```
.. dsp_comps library index page


 DSP components library
 ======================
The DSP component library (``dsp_comps``) provides components for
digital signal processing (DSP).

.. toctree::
   :maxdepth: 1
   :glob:
   :caption: DSP components

   *.comp/*-comp
```

9.2.4 Documenting an OpenCPI Component Test Suite

The documentation for a component's test suite is currently part of the component documentation.

In a future release, a component test suite asset's documentation will be a separate document that is co-located with the component test suite asset and referenced from the main component document with an OpenCPI-specific Sphinx directive.

9.2.5 Documenting an HDL Assembly

The name of the documentation file that describes an OpenCPI HDL assembly is the assembly's XML file name with the **.rst** suffix instead of **.xml**. It is located in the **hdl/assemblies/** directory of an OpenCPI project in the same **<assembly-name>** directory as the HDL assembly it describes.

An HDL assembly document should provide information about:

- Why the assembly was created
- Any specific applications it is intended to support
- Which containers are specified for the HDL assembly, and for each one, how it maps the inputs and outputs of the assembly to other devices or CPUs in the system.

The following points about the function of HDL assemblies and containers may be helpful in the HDL assembly documentation development process:

- The XML for an HDL assembly basically specifies: which workers, how they are (statically) parameterized, how they are connected, and what the "external ports" of the assembly are as a whole.
- A container specifies, for each external port of the assembly, which devices they are connected to or whether they are "connected to the system's interconnect", which usually means they are connectable to software workers.
- The "default container" specifies to connect all external ports to the system interconnect so they are connectable to software workers somewhere else.

The asset documentation system does not currently provide a template for documenting an OpenCPI HDL assembly, so "**`ocpidev create hdl assembly`**" does not create a skeleton RST file when it creates an HDL assembly. Use the file system to create a file named **`<assembly-name>.rst`** in the same directory as the HDL assembly XML, and then use standard reStructuredText markup and Sphinx directives to create headings and content within **`<assembly-name>.rst`**.

9.2.6 Documenting an HDL Card

The name of the documentation file that describes an OpenCPI HDL card is the card's XML file name with the **`.rst`** suffix instead of **`.xml`**. It is located with the card's XML file in the **`hdl/cards/specs/`** directory of an OpenCPI project.

General information on what an HDL card document should contain to be supplied.

The asset documentation system does not currently provide a template for documenting an OpenCPI HDL card, so "**`ocpidev create hdl card`**" does not create a skeleton RST file when it creates an HDL card. To document an HDL card, use the file system to create a file named **`<card-name>.rst`** in the same directory as the HDL card XML, and then use standard reStructuredText markup and Sphinx directives to create headings and content within **`<card-name>.rst`**.

9.2.7 Documenting an HDL Device Worker

The name of the documentation file that describes an OpenCPI HDL device worker is the HDL device worker's XML file name with the **`.rst`** suffix instead of **`.xml`**. It is located in the **`hdl/devices/`** directory of an OpenCPI project in the same **`<device-worker-name>.hdl`** directory as the HDL device worker it describes.

General information on what an HDL device worker document should contain to be supplied.

The asset documentation system does not currently provide a template for documenting an OpenCPI HDL device worker, so "**`ocpidev create hdl device`**" does not create a skeleton RST file when it creates an HDL device worker. Use the file system to copy the generic worker template (located in **`$OCPI_ROOT_DIR/tools/ocpidoc/opencpi_documentation/rst_templates/`**

worker.rst) into the directory that contains the HDL device worker XML (**hdl/devices/<device-worker-name>.hdl**) and then rename it to **<device-worker-name>.rst.** Fill in the copied generic worker template with details about the HDL device worker.

Examples of OpenCPI HDL device worker asset documents to be supplied.

9.2.8 Documenting an HDL Platform Worker

The name of the documentation file that describes an OpenCPI HDL platform worker is the HDL platform worker's XML file name with the **.rst** suffix instead of **.xml**. It is located in the **hdl/platforms/<platform-name>/** directory of an OpenCPI project in the same **<platform-worker-name>.hdl** directory as the HDL platform worker it describes.

(General information on what an HDL platform worker document should contain to be supplied.)

The asset documentation system does not currently provide a template for documenting an OpenCPI HDL platform worker, so "**ocpidev create hdl platform**" does not create a skeleton RST file when it creates an HDL platform worker. To document an HDL platform worker, follow the instructions given for an HDL device worker: copy the generic worker template to the directory that contains the HDL platform worker asset, rename it to **<platform-worker-name>.rst**, and fill in the copied worker template with details about the HDL platform worker.

Examples of OpenCPI HDL platform worker asset documents to be supplied.

9.2.9 Documenting an HDL Primitive Core

The name of the documentation file that describes an OpenCPI HDL primitive core is the primitive core's XML file name with the **.rst** suffix instead of **.xml**. The "**ocpidev create hdl primitive core**" command creates this file in the project's **hdl/primitives/<primitive-name>** directory along with the HDL primitive core's XML.

Replace the "Skeleton outline:" line underneath the document title with a description of the HDL primitive core and its scope. Create subsections below the title as necessary. Everything you add to the template must be entered manually. There are currently no directives for automatically generating the information.

The HDL primitives assets provided with OpenCPI are not currently documented in asset documentation RST files. However, you can look at the README files in the **projects/core/hdl/primitives/** subdirectories for examples of the information that should be supplied for this asset type. You can also look at the HDL primitives documents provided in the OpenCPI SDR components library.

9.2.10 Documenting an HDL Primitive Library

The name of the documentation file that describes an OpenCPI HDL primitive library is the primitive library's XML file name with the **`.rst`** suffix instead of **`.xml`**. The "**`ocpidev create hdl primitive library`**" command creates this file in the **`hdl/primitives/<hdl-primitive-library-name>/`** subdirectory of an OpenCPI project in the same directory as the HDL primitive library it describes.

An HDL primitive library asset document describes the general purpose of the primitive modules contained in the library and contains a **`toctree`** directive that points to their documentation. Replace the "Skeleton outline:" line in the **`<hdl-primitive-library-name>.rst`** file with a description of your HDL primitive library and its scope. The template sets up everything else for you.

Here is an example of an OpenCPI HDL primitive library asset document:

```
Communication Primitive Library
===============================
Primitive library for dealing with communication protocols.

.. toctree::
   :maxdepth: 1
   :glob:
   :caption: Primitives

   */*
```

9.2.11 Documenting an HDL Slot

The name of the documentation file that describes an OpenCPI HDL slot is the slot's XML file name with the **`.rst`** suffix instead of **`.xml`**. As with all OpenCPI assets, the documentation file is co-located with the HDL slot's XML file, which is the **`hdl/cards/specs/`** directory of an OpenCPI project for platform-independent HDL slots and the **`hdl/platforms/<platform>/devices/specs/`**directory of an OpenCPI project for platform-specific HDL slots.

(General information on what an HDL slot type definition document should contain to be supplied.)

The asset documentation system does not currently provide a template for documenting an OpenCPI HDL slot, so "**`ocpidev create hdl slot`**" does not create a skeleton RST file when it creates an HDL slot definition. Use the file system to create a file named **`<slot-name>.rst`** in the same directory as the HDL slot definition, and then use standard reStructuredText markup and Sphinx directives to create headings and content within **`<slot-name>.rst`**.

Examples of OpenCPI HDL slot definition documents to be supplied.

9.2.12 Documenting an OpenCPI Project

The name of the documentation file that describes an OpenCPI project is the project's XML file name with the **`.rst`** suffix instead of **`.xml`**. The "**`ocpidev create project`**" command creates this file in the project's top-level directory along with the other project-related assets.

A project document serves as the "welcome" page for the project and as a directory to the assets documented in the project. A project document contains:

- A brief description of the purpose of the project and the general types of items that can be found there, similar to a README file. Avoid listing or describing project contents as it is redundant and creates a maintenance burden.
- A description of the criteria or rules in place for adding assets to the project or for excluding them from the project.
- A **`toctree`** directive that specifies the paths to project subdirectories/libraries that contain asset documentation for different asset types in the project. Each subdirectory path in the **`toctree`** directive points to a "directory index" file that provides the paths (aka "table of contents") to the asset documents in that subdirectory. For example, the notation **`/components/components`** tells the asset documentation system to look in the project's **`components`** library for a directory index file named **`components.rst`** that provides the paths to the asset documents for components that are contained in the **`components`** library. The template's **`toctree`** directive provides paths to the **`components`** library and index file (**`components/components`**), the HDL **`primitives`** library and index file (**`hdl/primitives/primitives`**), and the project-level **`specs`** directory and index file (**`specs/specs`**). For information on how a project's assets are structured, see the section "Developing OpenCPI Assets in Projects" in the ***OpenCPI Component Development Guide***.

  In a future release, an OpenCPI-specific directive that automatically locates the documentation assets in a project will be provided.

To document an OpenCPI project, modify the **`Project.rst`** file as follows:

- Replace the "Description of project" line with a brief description of your project and a description of the criteria for placing content into the project.
- Add any subdirectory paths to the **`toctree`** directive that are not already there by default. For example, if your project has a library of applications under the **`applications`** subdirectory, you would add the line **`applications/applications`** to the **`toctree`** directive. This line directs the system to look for a directory index file named **`applications.rst`** that provides the paths (usually as a wildcard expression) to the application asset documents contained in subdirectories below the **`applications`** subdirectory. Note that

currently you must create the directory index files **`applications.rst`** and **`specs.rst`** yourself; they are not automatically created by "**`ocpidev create`**". See the section "Creating the OpenCPI Asset Documentation Structure" for details.

9.2.13 Documenting an OpenCPI Protocol

The name of the documentation file that describes an OpenCPI protocol is the protocol's XML file name with the **`.rst`** suffix instead of **`.xml`**. The "**`ocpidev create protocol`**" command creates this file in the same directory as the protocol's specification (OPS).

The template file for describing a protocol has only one section: "Protocol", which includes the protocol specification (OPS) XML file. To this template, add:

- A one-sentence description (incomplete sentence) of the protocol's purpose underneath the title. For example:

```
Protocol for streamed complex short data samples, with time and
metadata opcodes which are used to provide additional information
about the sample data while maintaining alignment with samples in
the streamed data.
```

- An introductory sentence and a "literalinclude" link to the metadata XML file that the OPS references. For example, if the OPS references the metadata file **`timed_samples_metadata.xml`**, add the following lines:

```
The included ``timed_samples_metadata.xml`` protocol structure is:

.. literalinclude:: timed_samples_metadata.xml
   :language: xml
```

9.2.14 Documenting an OpenCPI Worker

The name of the documentation file that describes an OpenCPI worker is the worker's XML file name with the **`.rst`** suffix instead of **`.xml`**. The "**`ocpidev create worker`**" command creates this file, along with the worker source code and worker description (OWD) skeleton files, in the **`<worker-name>.rcc`** or **`<worker-name>.hdl`** directory, depending on the authoring model specified to the "**`ocpidev create worker`**" command.

To document an OpenCPI worker, edit the **`<worker-name>.rst`** file to describe your worker, following the skeleton outline in the template and the instructions given below.

9.2.14.1 Add a Worker Summary (optional)

You can optionally create a short description of the worker that summarizes its function and purpose and place it under the worker title in the worker document. The asset documentation system will then automatically include this summary in the HTML output

(in the “Implementations” section) of any component document that references the worker with the “implementation” directive.

9.2.14.2 Include a Block Diagram of the Worker Implementation

If there’s a need to illustrate your worker with a block diagram, follow the instructions given in the section on creating component figures, with the following differences:

- Describe the worker-specific properties and ports, not the component-specific ones.
- You can’t use the figure numbering reference markup in a worker document. Remove the figure template markup and instead introduce the figure with “the following figure”. Here is an example markup:

```
The following figure shows a block diagram representation of the HDL
implementation:

.. figure:: ../<component-name>.comp/figures/myworker_diagram.svg
   :alt: HDL Implementation for MyWorker
   :align: center

   MyWorker Block Diagram
```

Keep worker block diagrams and other worker-specific illustrations with the worker asset documents.

9.2.14.3 Describe the Worker Properties and Ports

The “Details” section of the template contains the OpenCPI-specific directive **`ocpi_documentation_worker`** that automatically creates “Worker Properties” and “Worker Ports” subsections underneath any information about the worker that you add manually and fills them with bulleted lists of worker-specific property and port information taken from the OWD. Its usage is:

```
.. ocpi_documentation_worker::
```

By default, no arguments are needed. The option that can be used with the directive is:

- Additional text to describe any worker properties (including parameters) or ports, added in the “body” of the directive, which consists of a blank line underneath the directive followed by one or more text lines indented three spaces to line up with the first character of the directive (see the example below). This text must be the name of the property or port followed by a colon symbol (:), with each property or port text on a new line. (Additional property or port text is not passed as an option of the directive, as directive option names must be known at the time of writing the directive, which for property names, parameter names and port names is not possible.)

Here is an example of using this directive from the core component's **`backpressure`** HDL worker, where descriptions for the worker ports are added by hand:

```
.. ocpi_documentation_worker::

   in: Size defined by ``IDATA_WIDTH_p``.

   out: Sample size defined by ``ODATA_WITH_p``.
```

Look at the HTML output for the **backpressure** HDL worker to view the results.

Note that the "Properties" section only describes worker-specific properties; the component properties that can be used in the worker are described in the component document's "Properties" section. For an example, look at the HTML output for the **metadata_stressor HDL worker** document and compare the properties shown in the "Worker Properties" subsection with the properties shown in the "Properties" section of the HTML output for the **metadata_stressor component** document.

9.2.14.4 Add Subsections to "Details" As Necessary

Some workers have additional information that needs to be described, like finite state machines or control timing and signals information. An RCC worker that is a device proxy worker may also require additional information. Create new subsections under "Details" as necessary to cover these topics. See the Metadata Stressor HDL worker RST document (**projects/core/components/metadata_stressor.hdl/metadata_stressor-hdl.rst**) for an example of how to format this information.

9.3 Creating the OpenCPI Asset Documentation Structure

OpenCPI asset documentation structure is a nested series of RST document files that follows the OpenCPI project directory layout described in the section “Developing OpenCPI Assets in Projects” in the ***OpenCPI Component Development Guide***. The project document is the top-level document. It gives the paths to RST directory index files in the next level of subdirectories (**`specs/`**, **`components/`**, **`hdl/`**, **`applications/`**). These files either list the asset documents in the subdirectory or give the paths to directory index files in the next level of subdirectories. For example, the index file in the **`specs/`** directory usually lists (as a wildcard expression) the protocol documents contained in the subdirectory. The index file in the **`components/`** directory usually gives the paths to component library index files, because this directory often contains multiple component libraries, each with their own index file that points to the components in their library.

Most, but not all, of these directory index files are created automatically when the relevant asset is created with **`ocpidev`**. Directory index files that are not created automatically can be created by hand if necessary by copying their RST templates to the desired location, renaming them, and optionally editing their contents. The RST templates are located at:
`$OCPI_ROOT_DIR/tools/ocpidoc/ocpi_documentation/rst_templates/`

The next sections provide details.

Note that subsidiary files that are included into an asset document, whether or not they are local, should be named with the suffix **`.inc`** , *not* **`.rst`**. Using the **`.inc`** suffix for included subsidiary files is currently necessary to prevent them from being mistakenly generated by the asset documentation system during a documentation build operation.

In a future release, parts or all of this structure will be automatically constructed as part of OpenCPI asset building with ocpidev and will not persist in the file system.

9.3.1 Documenting a Library of OpenCPI Applications

If you're creating asset documentation for multiple applications in the **`applications/`** directory, you'll need to create a directory index RST file named **`applications.rst`** in that directory to point to these documents. To create this directory index file, copy the RST template **`applications-directory.rst`** to the top-level **`applications/`** directory in the project and rename it **`applications.rst`**. No template file editing is necessary. Note, however, that you *do* currently need to modify the project's RST file to include the path to this directory index file; see Documenting an OpenCPI Project for details.

9.3.2 Documenting a Directory of Protocol Specifications

If you're creating asset documentation for multiple protocol specs in the **`specs/`** directory, you'll need to create a directory index RST file named **`specs.rst`** in this directory to point to these documents. To create this directory index file:

- Copy the RST template **`specs-directory.rst`** to the top-level **`specs/`** directory in the project and rename it **`specs.rst`**.
- Remove the **`%NAME_PROPER%`** notation in the heading of the copied template file and optionally replace it with something more descriptive (the template expects that the **`specs/`** directory contains protocols, as shown in the default heading "Protocols").
- Replace the "skeleton outline" in the copied template file with a short, generic description of the directory contents and purpose.

9.4 Documenting Hardware Getting Started Guides

Hardware getting started guides are “secondary” asset documents and are not created with the asset by **`ocpidev`**. A getting started guide is only necessary for a hardware asset when it has installation issues beyond the generic installation for its type described in the ***OpenCPI Installation Guide***. When this is the case, the Getting Started Guide for the hardware asset describes these installation-specific details.

9.4.1 Creating a Hardware Getting Started Guide

OpenCPI provides getting started guide RST templates for system, platform, and HDL card asset types located at:

`$OCPI_ROOT_DIR/tools/ocpidoc/ocpi_documentation/rst_templates/`

Each RST template provides a skeleton outline and instructions for adding content. To create a getting started guide for a hardware asset, copy the appropriate template to the directory in which the asset is located and rename it to **`<asset-name>-gsg.rst`**. Next, follow the rest of the steps described in the section “Documentation Development Workflow”.

9.4.2 Documenting a Directory of Getting Started Guides

Some OpenCPI System support Projects (OSPs) provide a set of hardware assets, where each asset requires its own getting started guide. In this scenario, there needs to be a directory index file in the directory above the individual asset subdirectories that specifies the path to the getting started guides located in those subdirectories. To create this file, copy one of the **`*-directory.rst`** RST templates (for example, **`applications-directory.rst`** or **`primitives-directory.rst`**) to the directory above the asset subdirectories and rename it to **`gsg.rst`**. Change the pathname given in the copied file to specify the path to the individual asset directories.

Here is the directory index file for the suite of platform getting started guides in the Avnet Microzed/PicoZed OSP (**`ocpi.osp.avnet`**), located at **`hdl/platforms/`**:

```
.. Getting started guide directory index page


Getting Started Guides
======================
Available MicroZed and PicoZed getting started guides.


.. toctree::
   :maxdepth: 2
   :glob:

   */*-gsg
```

In a future release, this structure will be automatically constructed as part of OpenCPI asset building with ocpidev and will not persist in the file system.

9.5 Using Include Files and Substitution Strings to Share Common Information

If you need to document multiple assets and information about one asset is common to other assets, you can create a directory of "include" files to be shared between some or all of the assets and then use the **`ocpi_documentation_include`** directive in the individual asset's document to include them at document build time, optionally replacing variables used in the “included” documents with specific values defined in the “including” document. The directive allows you to:

- Share common information between documents for similar assets – for example, where individual component assets in a component library share common functionality and thus share common diagrams and equations – without generating Sphinx “duplicate label” warnings.
- Write generic shared documents that are customized to individual asset documents at build time. You create variables (called "substitutions” in Sphinx) in the shared documentation for things like asset names and path names and then use the directive in the individual asset documents to define replacement values for these strings that are applied when the document is built and the HTML is generated.

This directive is similar to the Sphinx include directive but also includes “find and replace” functionality. Its usage is:

```
.. ocpi_documentation_include:: <include-file-path>

   |<substitution_string_name>|: <value>
    ...
```

where the argument to the directive is the path to the directory of files to be shared and options to the directive are one or more “substitution string” name-value pairs, in Sphinx substitutions format. The next sections provide more detail and usage examples.

9.5.1 Creating the “Include” File Directory

Name the directory **`include`** and locate it one level above the individual asset documents that are to share its contents. Examples are:

- **`components/include`** for individual component asset documents located in the directory **`components/<component-name.comp>`**.
- **`hdl/devices/include`** for individual HDL device worker asset documents located in the directory **`hdl/devices/<device-name-hdl>`**
- **`hdl/platforms/include`** for individual platform “getting started guide” asset documents located in the directory **`hdl/platforms/<platform-name>`**

Text files in the directory should have the suffix **.inc**. Shared images are SVG files (see the hints section for details) and have the suffix **.svg**.

9.5.2 Using Substitution Strings

As mentioned above, a substitution string is a name-value pair in Sphinx substitutions format:

```
|<substitution_string_name>|: <value>
```

where **<substitution_string_name>** is the variable used in the shared document and **<value>** is the text to be substituted for the variable when it is included in the individual asset document.

Substitution strings can be used to create placeholders in shared file content for asset-specific items like asset name, paths to images, URLs for vendor documents, labels on boards, and so on that are resolved to the asset-specific values specified in the directive in the individual asset document. For example, you can create a "**|component_name|**" substitution string variable to be used in a shared file, and then define a value for the variable within the **ocpi_documentation_include** directive in the individual asset document, for example:

```
|component_name|: backpressure
```

The value you specify with the directive – in this case, "**backpressure**" – is substituted for all occurrences of the string in the included file when the document is built and the HTML is generated.

Another example of using substitution strings is to create subdirectories of asset-specific images in the "include" directory instead of with the individual asset document and then specify the location of these image subdirectories in the **ocpi_documentation_include** directive as the value of a **|path_to_figures|** substitution string (see the example below). Make sure you label these subdirectories appropriately by using a descriptive filename, or provide a README file in the include directory to identify the different subdirectories. See the getting started guide setup in the Avnet and the Xilinx OSPs for examples.

9.5.3 Examples

Here is an example include directive in an individual component document. It references a shared document and defines values for the **|component_name|** and **|path_to_figures|** substitution strings used in the referenced document:

```
.. ocpi_documentation_include:: ../include/shared_section.inc

   |component_name|: polyphase_clock_synthesizer
...|path_to_figures|: ../include/
```

The example include directive allows for the following common figure definition in **shared_section.inc**:

```
This problem is shown in :numref:`|component_name|-sampling-points-
diagram` below.

   .. _|component_name|-sampling-points-diagram:

   .. figure:: |path_to_figures|symbol_sampling_points.svg
      :alt: Symbol Sampling Points
      :align: center

      Symbol Sampling Points
```

The **|component_name|** and **|path_to_figures|** substitution strings in the shared file are resolved to the values specified in the directive when the individual component document is built and HTML is generated.

Here is an example include directive from the ***ZCU102 Getting Started Guide.*** It references an overview section in the shared directory that is common to all ZCUxxx devices, specifies the path to the ZCU102-specific images subdirectories that are stored in the shared directory, and defines values for the substitution strings used in the common overview section:

```
.. ocpi_documentation_include:: ../include/zcu_overview.inc
   |device_name|: ZCU102
   |platform_name|: zcu102
   |path_to_figures|: ../include/zcu102_figures/
   |device_type|: Multiprocessor System-on-Chip (MPSoC)
   |product_brief|: product brief at
https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html
```

The getting started guide asset documents in the Avnet and Xilinx OSPs provide more examples of how to use the **ocpi_documentation_include** directive in this way.

9.5.4 Asset Document Templates That Use Substitution Strings

Some OpenCPI asset documentation templates define substitution strings and use them in section headings and text as placeholders for the names to be supplied by the asset document author. For example, the template for a “system” getting started guide defines substitution strings for names typically used in system getting started guides, like vendor name, system name, product family, and device family. The template supplies placeholder values for each string that are meant to be replaced with strings like "Xilinx Zynq UltraScale+ ZCU102", "Avnet MicroZed", or "Digilent ZedBoard". If you don't plan to re-use your asset document for similar assets and/or your asset names are unique, you can delete the string definitions that you inherited from the template and simply type the given names directly into the section headings and text. Alternatively, you can continue to use the substitution strings by supplying the appropriate values for your asset to the string definitions provided by the template.