What are some best practices for integrating code samples with my writing?

Code samples are an important part of software documentation. They provide examples for users to follow and modify, and allow users to compare outputs and verify results.

Highlighting code samples

When working with any kind of code sample, make sure that you are familiar with your markup language or documentation standard that you are using. For example, your standard might have specific tags that you have to use for specific types of code samples. It is up to you to correctly classify what type of code sample you are trying to integrate and label it accordingly.

For example, if you are using AsciiDoc and have a YAML sample that you want to integrate, you must use the `[source,yaml]` source code block. Using the correct tags is important because syntax in code samples can be highlighted differently depending on what type of code sample it is. A YAML configuration file has different syntax highlights than JSON, for example.

If tagging does not highlight the code sample automatically, or if there is no appropriate tag available, you can still use a general highlighting tag. Your language or standard might have a universal code block to accomplish this. For example, Markdown uses the following syntax to create an undefined code block:

```
```
{
  "firstName": "John",
  "lastName": "Doe",
}
```
```

If you are not using any markup language or other standard, you need to use manual highlighting. It is generally accepted that code samples should be monospaced, if no other option exists. On many platforms, you can achieve this by enclosing the code sample in backticks, as seen in the following example: `code-sample`

Types of code samples

See the following list of common code sample types:

Commands

Commands are prompts that users can run in command lines. Shell refers to any program that offers a command line interface. You might also encounter the term Bash, which is a specific type of shell that is often used in Unix/Linux.'

A good practice is to present commands line by line and one by one. Developers sometimes combine and stack commands to save space, but if the goal is to present them to the user clearly, they should be separated to make sure that the user understands what each command does.

Often, you will also see a `$` before a command to indicate where the command starts. Make sure that you are following whatever standard or style your organization uses.

See the following example of a command:

```
sudo pacman -Syyu
```

Configuration files

Configuration files are files with lines of code that set specific parameters. For example, YAML files. Configuration files presented in software documentation can save time for users by offering pre-configured settings that the user can copy and paste. However, these examples often need to be altered to suit specific user needs, as every user has a unique environment.

Make sure to offer enough information for users to understand what changes they need to make to the configuration sample, if any. Common variables are names, keys, or settings such as system architecture.

See the following example where the user needs to change one variable and one parameter is explained:

```
doe: "a deer, a female deer"
 ray: "a drop of golden sun"
 xmas: true
 french-hens: 3
 calling-birds:
   - huey
   - dewey
 xmas-fifth-day:
   calling-birds: four
   french-hens: 3
   partridges:
     count: 1
     location: "a pear tree" <1>
   turtle-doves: two
```

<1> Add the location where you saw the the partridges.

Field Table

Field	Optional or Required	Description
`xmas`	Required	Boolean that is set to `false` by default. Determines if the day of sightings is on xmas or not.

Outputs

Outputs are lines of code that the user might receive after running certain commands, when viewing logs, or if requested by the user. Outputs can be used to verify if the software is running correctly. Outputs can also be used for troubleshooting.

Generally, outputs can be long and disruptive to users reading documentation. Always ask yourself if an output sample is necessary, or if you can simply summarize what the user should look for in the output they receive. If you decide to include an output sample, check if you can

shorten it to only the parts that are relevant to the user in that specific section of the documentation. Enclosing output samples in generic code blocks is usually sufficient.

Even when including output samples, always tell the user what they should look for.

Integrating code sample into procedures

Generally, code samples should only appear in procedures. To ensure accessibility, always write prompts before providing a code sample. This allows users who might rely on screen readers to be prepared when samples appear in the documentation. See the following example prompts for different types of code samples:

Commands

Run the following command to update your operating system:

Configuration files

Create a file called `settings.yaml` *and add content that resembles the following example:*

Outputs

See the following example output. Make sure that the `test-services` *pod is healthy and running.*

Replaceable variables

We already touched on replaceable variables in the code sample types section. Often, users need to make additional changes to code samples before they can use them. Items in code samples that need to be changed are generally considered variables, but you might also encounter fields, parameters, or values.

If the user needs to replace several variables, code samples can quickly become long and difficult to understand. To simplify the process in commands, it is good practice to only call out a

replaceable variable if there are only one or two that need to be replaced. You can also add a note or sentence below the command for additional information. It is common to indicate replaceable values by enclosing them in `<>`. See the following example:

Run the following command to create a new text file. Replace <name> *with the name of your file and replace* <type> *with the file extension:*

```
touch <textfile>.<type>
```

Note: Possible values for `<type>` are `txt` or `rtf`.

Additional resources

- https://docs.asciidoctor.org/asciidoc/latest/verbatim/source-blocks/
- https://www.markdownguide.org/extended-syntax/
- https://www.freecodecamp.org/news/bash-scripting-tutorial-linux-shell-script-and-command-line-for-beginners/
- https://www.cloudbees.com/blog/yaml-tutorial-everything-you-need-get-started

Credit

Thank You!

See the Committee on Structured Authoring and Content Management page of the ACM SIGDOC website to learn more about committee activities, available resources, and volunteer opportunities.

See https://acm-sigdoc-structured.org to learn more about committee activities, available resources, and volunteer opportunities.