WSPS Automated Documentation Generation Process and Integration

This document is designed to give an overview of the automation process and how it is designed in order to facilitate documentation automation. The current state is in its prototype concept now, in this repository, but it is fully functional and capable of generating structured documentaion based on local or remote markdown files.

Process Overview

As described in the top levelREADME, the intent is to provide a process that is consistent and re-usable but tailorable to each program. The process is based on utilization of the Gitlab CI and a multi-project based solution integrated with CI scripting to gather remote markdown content and combine into one single artifact. In this case, it supports Word (docx) files and is capable of generating PDF's with additional tools. The common project setup is essentially one main project where the documentation templates and variables a program would desire and a remote project containing markdown files of content to be collected (This can be any number of projects based on program need.) The main project currently named doc_template is the current project that maintains all of the different documentation templates, that are able to be re-used or modified to a programs need. The remote project or data source named doc_stub is an example project to show how to retrieve markdown files from projects outside of the main template project.

The easiest way to describe the process flow is that it primarily controlled by the CI environment, in this case, the Gitlab CI/CD functions provide the mechanisms for driving this document generation process. This concept is depicted below between the WSPS devops projects which is the enabler for programs.

The flow shows the utilization of the ci_templates project. This project houses common CI templates for use in Gitlab, and has templated stages for the entire build process of the baseline. The documentation generation is also captured here and those templates can be integrated and used in the individual program CI pipelines. Then based on variables defined in the defaults.yaml and the documents identified in thedocuments.yaml provides the context of what documents are to be generated. Based on what documents are defined they are organized by folders with various sets of markdown that setup the constructs of the document. Each document can be seperated into as many markdown files (.md) as needed. The doc_template repository has a defined pattern that is common across the look and feel of the documents. This is driven by the use of a docx template file, that provide document formatting and structure and becomes the basis of each document generated. With that template, then it comes to organizing the content inside that document templated structure. Inside each document template folder there will be a index.md file. This file controls the orientation of the Document navigation (headings and ordering of content). This will list both local and remote markdown files. Below is the different annotations used by the CI to find all of the content.

The first content table shows accessing local markdown files (local to the doc_templates project).

```
 {% include 'api/01.scope.md' %}
{% include 'api/02.referenced.docs.md' %}
{% include 'api/03.overview.md' %}
{% include 'api/04.api.introduction.md' %}
```

The second content table shows accessing markdown remote files (outside or remote of the doc_templates project)

```
 {% include 'doc/remote_documentation.md' %}
```

These annotations can be utilized to format any document and provide a order or flow to the document.***Note:*** `The configuration of the CI template job is the search for all markdown files in a specific folder. It is programed to look in the /doc folder in the root of the remote project.`

Both of these projects maintain a relationship through the .gitlab-ci.yaml. The doc_stub project generates the artifacts (any markdown in the /doc folder). This then triggers the doc_templates to start a pipeline and start creating a new set of documents based on the content provided in the local and remote content areas.

Program Usage

As a program, the most logicial solution is the fork the ci_templates and doc_templates projects into your development sub group. Then once the metadata (variables) and content is manipulated to the program's liking they can engage the CI process on there end to integrate this capability into their pipelines.

Program Doc Builder

There are some pre-requisites for completely running this function in various sub-groups or in an isolated environment.

- Requires Gitlab CE or EE (depending on program funding) with enough capability to host artifacts in Gitlab
- The CI templates for document generation require the use of a gitlab-rununer configured as a docker runner on a host.
- If programs require a modified docx template, there is additional work needed to generate the proper Jinja2 defined template. Otherwise, the defaults can be used.

Versioning/tagging of component repositories is recommended for tracking and controlling documentation changes in sync with software changes. The gitlab-ci.yaml templates will have placeholders to accomodate for programs semantic versioning needs.

Documentation

The intention is that the majority of the document content will be generated by third party tools and driven by the a CI/CD pattern(as indicated earlier in the document).

Each program entity can utilize various tools to accomplish documentaiton automation.

For API's to be used, they must be discoverable and documented. Concise, clear, and complete documentation is one of the single most important qualities of API's.

Key Points for Documenting:

- Publish API documentation, and provide links to the documentation from the API endpoint. Use Automation where applicable.
- Provide feedback and support channels for API users.
- Consider using some sort of literate system that generates the API documentation from API code (or vice versa), to ensure there always precise alignment between the API and it's documentation.
- Consider using open source API specification tools to simplify and standardize the means at which API's are documented.

Api Documentation Tooling Examples:

- Swagger
- Slate
- FlatDoc
- ReadmeIO
- APIary
- Doxygen
- iodocs