WSPS Automated Documentation Generation

This repository contains source material to be used in the creation and maintenance of program plans.

Purpose

Project plans often suffer from a lack of reliability due to inconsistencies within the plans themselves. Changes to the project often fail to be reflected throughout all documents and it is difficult to keep the plans unified across all parties.

By utilizing a distinct set of project parameters along leveraging the capabilities of version control software, this repository provides projects the ability to create their documents in a shared environment where changes can be reliably implemented.

Approach

The source material is written in Markdown and is divided into separate files in this repository. The separate files are combined using a Python script (which incorporates a template language with variable substitution) and then passed to Pandoc, a tool for generating Word or PDF documents from Markdown source.

This approach to document generation was chosen for multiple reasons:

- Multiple documents can be created sharing "snippets" of content. This avoids the issue where different program plans are inconsistent, while still enabling a program plan to be used as a standalone document.
- Users of these program plans can easily customize the content by providing a set of parameters (e.g. program name, tool selections) and can quickly generate a set of program plans for a specific program.
- Updates to these program plans can be reviewed using standard peer review tools and tracked in a version control system. Updates to common snippets can be immediately reflected in all places where they are used.

Editing

Editing can be done with any text editor or directly in this Git repository management tool. To edit in this tool:

1. Click the "Fork" button at the top of the overview page. This will create a separate repository with write permission. You can select an existing group or select your own name to fork the project into a personal space.
2. Navigate to the file of interest and click "Edit" at the top of the page.
3. Make the changes, fill out the "Commit message" box with a description, and click "Commit changes".
4. Use the "Merge Requests" feature on the left side to create a merge request back to this repository. Once accepted, this will merge your changes into the common version.
5. For more complex changes, you can use the "Branches" feature to create a separate branch within your repository to hold your changes. This allows you to make changes to multiple files and submit them all

together in a merge request.

For more information, see the Gitlab User Documentation.

Building

The documents are built automatically by the Gitlab CI server. To see the latest build, click the badge at the top of this README, or the "CI / CD" item in the left hand menu.

To build the program plans from this repository requires a system with the following tools:

- Python 3
- Python packages: `pandoc-fignos`, `Jinja2`, `PyYAML`, `docbuilder`
- Pandoc

The included `Dockerfile` can be used to build a docker image from which to build the documents.

Provided that all of the dependencies have been installed, the documents can also be built by simply running the `build_documents.py` script.

A web application can be used to build the documents either from scratch, through the `metadata.json` file, or by using a specified Gitlab repository.

Ideally, the easiest use case is to utilize the Docker approach, and utilizing the dockerfile to either build the docker image manually to run in your environment. Or utilize a pre-existing built container in this project's registry. Following this link: Docbuilder Image

```
docker pull registry.gitlab.us.lmco.com:443/ws/wsps/wsps-devops/doc_templates/docbuilder
```

Customizing

There are three ways to customize the documents generated in this repository:

1. Find / replace. The default parameters have been built to simplify find and replace in the generated documents. However, this approach loses the benefits of the document build process, as any updates would then have to be made manually across the documents.
2. Web application. Use the web application to provide parameter values. The build process also generates a `metadata.json` file that includes all of the parameters provided; this `metadata.json` file will be used in the web application to re-generate the documents.
3. Repository fork (preferred). By forking the repository in the version control tool, it is possible to update the `defaults.yaml` file directly and to use a continuous integration server to generate a program-specific set of documents. This is the preferred approach, because it allows automatic generation of updated documents as changes are made, changes are captured in the version control tool, and any improvements to the documents can be submitted upstream to benefit all users of these plans.