Documentation

01. Getting Started

01.01. Basic Usage

This guide will walk you through the process of setting up and customizing the INI Configurator to create your own configuration editor application. The configurator allows you to design a custom graphical interface for managing .ini files by modifying fields and constants.

Step 1: Fork the Repository

1. Navigate to the GitHub repository LaswitchTech/ini-configurator.
2. Click the Fork button in the top-right corner to create your own copy of the repository under your GitHub account.

Step 2: Set Up a Development Branch

1. Go to your forked repository.
2. Create a new branch named dev:
    - In GitHub, go to the Code tab.
    - Click the branch dropdown, type dev, and select Create branch: dev.
3. This branch will be used to compile your .exe file and manage development.

Step 3: Clone the Repository Locally

1. Clone your forked repository using Git:

```
git clone -b dev https://github.com/<your-username>/ini-configurator.git
cd ini-configurator
```

2. Verify that you are on the dev branch:

```
git branch
```

Step 4: Customize the Application

Modify the fields.json File

The `src/lib/fields.json` file defines the sections and fields displayed in the configurator. Update this file to specify the `.ini` configuration options you want to manage.

Example: Here's an example of a `fields.json` entry:

fields.json

```
{
    "General": {
        "appName": {
            "label": "Application Name",
            "tooltip": "The name of your application",
            "type": "text",
            "default": "MyApp",
            "required": true
        },
        "debugMode": {
            "label": "Enable Debugging",
            "tooltip": "Enable or disable debug mode",
            "type": "checkbox",
            "default": "false",
            "required": false
        }
    }
}
```

Update Constants in configurator.py

At the start of the `src/configurator.py` script, modify the constants to set your application name, `.ini` file name, and debugging preferences:

```
# Declare Constants
APP_NAME = "INI Configurator"
INI_FILE = "conf.ini"
LOG_FILE = "log.log"
```

```
ENCODING = "mbcs" if sys.platform == "win32" else "utf-8"
DEBUG = False
```

- **APP_NAME**: Name of your application.
- **INI_FILE**: Name of the `.ini` file that will be managed.
- **LOG_FILE**: Name of the log file.
- **ENCODING**: This constant sets the default encoding method for the `.ini` file.
- **DEBUG**: Set to True for debugging during development.

Step 5: Use Utility Scripts

iconset.sh

This script converts your `src/icons/icon.png` file into icon formats suitable for Windows and macOS.

Usage:

1. Place your custom icon in `src/icons/icon.png`.
2. Run the script:

   ```
   ./iconset.sh
   ```

3. The script generates the required icon files for your application.

build.sh

This script compiles the configurator for macOS. **Usage:** Run the script:

```
./build.sh
```

The compiled macOS `.app` file will be created in the `dist/macos` directory.

Step 6: Compile for Windows

To compile the `.exe` file for Windows:

1. Push your changes to the dev branch:

   ```
   git add .
   ```

```
git commit -m "Customize fields.json and constants"
git push origin dev
```

2. GitHub Actions will automatically build the `.exe` file. It uses the `build.yml` workflow in the `.github/workflows/` directory.

Step 7: Test and Refine

1. Run the application locally or test the `.exe` or `.app` files to ensure everything works as expected.
2. Refine the fields, constants, or icons as needed.
3. Repeat the build process after each update.

Optional: Enable Debugging

If you encounter issues during development, set `DEBUG = True` in `src/configurator.py`. This enables logging of additional details to help troubleshoot problems.

Notes

- **File Paths**: Ensure all resource paths in your code (e.g., icons, stylesheets) work for both development and compiled environments. Use the `self.resource_directory` constant in `src/configurator.py` to access files dynamically.
- **Branch Management**: Keep the main branch clean for releases. Use dev for development and testing.

Conclusion

By following this guide, you can easily set up, customize, and build your own version of the INI Configurator. Enjoy creating your custom configuration editor!

01.02. Creating a Release

Once you've finished customizing and testing your configurator, follow these steps to create a release version:

Step 1: Create a Pull Request to Merge dev into stable

1. Go to your forked repository on GitHub.
2. Navigate to the **Pull Requests** tab and click **New pull request**.
3. Set the base branch to stable and the compare branch to dev.
4. Review the changes in the pull request and click **Create pull request**.
5. Once the pull request is created, review and merge it into the stable branch.

Step 2: Tag the Stable Branch for a Release

1. After merging, ensure you are on the stable branch:

```
git checkout stable
git pull origin stable
```

2. Create a version tag for the release:

```
git tag v1.0.0
git push origin v1.0.0
```

    1. Replace `v1.0.0` with your desired version number following semantic versioning conventions (e.g., `v1.1.0`, `v2.0.0`).

Step 3: Trigger the Release Workflow

The push of a version tag (e.g., `v1.0.0`) to the stable branch will automatically trigger the GitHub Actions workflow for creating a release. This workflow will:

- Package the application for both Windows (.exe) and macOS (.app).
- Upload the compiled files as assets to the release in GitHub.

Step 4: Download the Release

1. Once the release workflow is complete, go to the **Releases** section in your repository.

2. Find the newly created release (e.g., `v1.0.0`).
3. Download the compiled assets (`Configurator.exe` for Windows and `Configurator.app` for macOS).

Notes for Releases

- **Branch Management**: Keep the stable branch clean and only merge fully tested changes from the dev branch.
- **Versioning**: Use semantic versioning for tags (vX.Y.Z) to indicate major, minor, and patch updates.
- **Automated Workflow**: Ensure your GitHub Actions workflows (build.yml and others) are correctly configured for release tagging.

Conclusion

By following these steps, you'll ensure a smooth release process and maintain a clear separation between development and production code.

02. Troubleshooting

02.01. Getting Help

General support

Having trouble getting **INI Configurator** working in your project? You can try emailing support@laswitchtech.com.

Reporting bugs

Found a problem with **INI Configurator**? Feel free to open a ticket on the **INI Configurator** repository on GitHub, but you should keep a few things in mind:

1. Use the GitHub issue search to check if your issue has already been reported.
2. Try to isolate your problem as much as possible. Try to create a minimal, verifiable, and complete example of the problem.

3. Enable debugging to try and retrieve as much data as possible.
4. Once you are sure the issue is with **INI Configurator**, open an issue with a description of the bug, steps required to reproduce, Debugging output, and the content of your `src/lib/fields.json`.

Requesting new features

New feature requests are usually requested by the INI Configurator community on GitHub, and are often fulfilled by fellow contributors.

1. Use the GitHub issue search to check if your feature has already been requested.
2. Please make sure you are only requesting a single feature, and not a collection of smaller features.
3. Once you are ready, open an issue with a description of the feature.

03. Configuration

03.01. Field Types

Type	Value	Options	Description	Example
`static`	None	None	This field type is only used to display static text such as instructions.	`"static": { "label": "Static", "tooltip": "This is a static field", "type": "static", "default": "Lorem Ipsum is simply dummy text.", "required": true }`

Type	Value	Options	Description	Example
`text`	String	None	This field type is a regular text input. This is also the default field type if the type used does not exist.	`"text": { "label": "Text", "tooltip": "This is a text field", "type": "text", "default": "My default value", "required": true }`
`password`	String	None	This field type is a regular text input that will hide the text by displaying *.	`"password": { "label": "Password", "tooltip": "This is a password field", "type": "password", "default": "MyPassword", "required": true }`
`number`	Integer	None or Object(Keys: **min**, **max**)	This field type is a number input.	`"number": { "label": "Number", "tooltip": "This is a number field", "type": "number", "default": 1234, "options": { "min": 0, "max": 2000 }, "required": true }`

Type	Value	Options	Description	Example

raw	String/None	None	This field type is used to output a particular string in the configuration file. When checked, the default value is added in the related section of the .ini file.	"raw": {
				"label": "Raw",
				"tooltip": "This is a raw field",
				"type": "raw",
				"default": "string",
				"required": false
				}

checkbox	Boolean	None	This field type is a regular checkbox.	"checkbox": {
				"label": "Checkbox",
				"tooltip": "This is a checkbox field",
				"type": "checkbox",
				"default": false,
				"required": true
				}

path	String	None	This field type can be used to retrieve paths. You may edit the text input or press the ... button to select a path.	"path": {
				"label": "Path",
				"tooltip": "This is a path field",
				"type": "path",
				"default": "path/",
				"required": true
				}

Type	Value	Options	Description	Example
`select`	String	Array(Possible Values)	This field type is a regular select input.	`"select": { "label": "Select", "tooltip": "This is a select field", "type": "select", "default": "Option 2", "options": [ "Option 1", "Option 2", "Option 3" ], "required": true }`
`multi-select`	String	Array(Possible Values)	This field type is a multi-select input. It is used to allow the selection of multiple preset values. The selected values will be converted to CSV	`"multi-select": { "label": "Multi-Select", "tooltip": "This is a multi-select field", "type": "multi-select", "default": "Option 1,Option 3", "options": [ "Option 1", "Option 2", "Option 3" ], "required": true }`

Type	Value	Options	Description	Example
`range`	Integer	None or Object(Keys: **min**, **max**)	This field type is a range input.	`"range": { "label": "Range", "tooltip": "This is a range field", "type": "range", "default": 50, "options": { "min": 0, "max": 100 }, "required": true }`
`filesize`	String	None or Object(Keys: **min**, **max**)	This field type is a range input. But it will convert the number of bytes to a human-readable format.	`"filesize": { "label": "FileSize", "tooltip": "This is a filesize field", "type": "filesize", "default": 67108864, "options": { "min": 1048576, "max": 134217728 }, "required": true }`

04. Changelog

- Merge pull request #4 from LaswitchTech/dev (2024/12/05 14:49)
- configurator.py: Build executable Configurator.exe (2024/12/05 14:49)
- Version Upped (2024/12/05 14:47)
- Re-Compiled the macOS app (2024/12/05 14:45)
- Fixing paths to be absolute instead of relative (2024/12/05 14:45)
- Merge pull request #3 from LaswitchTech/dev (2024/12/05 14:09)
- configurator.py: Build executable Configurator.exe (2024/12/05 14:07)

- Re-Compiled the macOS app (2024/12/05 14:04)