Project-JAAB = Just Another Addin Builder

A Github Repository can use this action to package the repository and create a JMP Addin to be used with JMP Software.

This action was presented and added to the JMP User Community for the Discovery Summit Americas in 2023. The materials for this presentation, including a recorded clip of usage, can be found here.

Whatcha Lookin' For?

Mandatory Prerequisites

Before building an addin with the Project-JAAB action, the following prerequisities must be met:

GitHub Token:

A token is required as an input to Project-JAAB when accessing private repositories for the addin compilation. This will be passed into the action as an input under `token`. Github documentation related to managing and creating personal access tokens can be found here. The token should have read and write permissions. After an access token is created, it can be added to the repository utilizing the action by navigating to the repository -> Settings -> Secrets and variables -> Actions -> New repository secret

Text File in .github/workflows of the Repository to describe the JMP Addin Menu Format

This text file should be located in the .github/workflows and can be named any name. This name will be input in the `jmpcust_txt_file` input field and is a required input. This file allows for flexibility for how the JMP Menu looks and where the addin is added and shows to the user.

Basic File format WITHOUT edits:

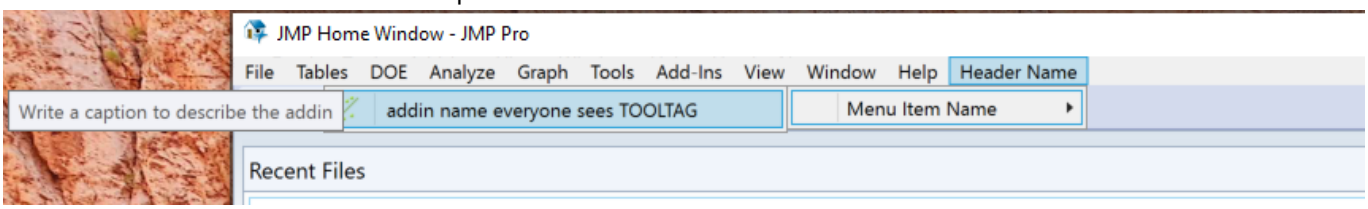
```
<!-- JMP Add-In Builder created --><jm:menu_and_toolbar_customizations
xmlns:jm="http://www.jmp.com/ns/menu" version="3">
<jm:insert_in_main_menu>
    <jm:insert_after>
        <jm:name></jm:name>
        <jm:menu>

            <jm:name>[Header Name]</jm:name>
                <jm:caption>[Header Name]</jm:caption>
            <jm:menu>
                <jm:name>[Menu Item Name]</jm:name>
                    <jm:caption>[Menu Item Name]</jm:caption>
                <jm:command>
                    <jm:name>[addin name everyone sees]</jm:name>
```

```
                        <jm:caption>[addin name everyone sees]
TOOLTAG</jm:caption>
                        <jm:action type="path">$ADDIN_HOME(AdDinIDDoNotTouCHY)\
[name of jsl file in addin].jsl</jm:action>
                        <jm:tip>[Write a caption to describe the addin]</jm:tip>
                        <jm:icon type="builtin">[Choose a symbol]</jm:icon>
                    </jm:command>
                </jm:menu>

            </jm:menu>
        </jm:insert_after>
</jm:insert_in_main_menu>
</jm:menu_and_toolbar_customizations>
```

What does this file look like when implemented?:



How to make the necessary edits: `[Header Name]`: edit this area of the template to say the naming you would like in the overarching menu location. This is the first section individuals will see and is the header item. In the case above, this is `Header Item`.

`[Menu Item Name]`: edit this area to be the next menu item that will be after the person selects whatever the `[Header Name]` is. In the case of above, this is `Menu Item Name`.

`[addin name everyone sees]`: edit this to be the name you want everyone to see for this addin. It doesn't need to match anything anywhere else but this is how the person will recognize this addin to use. In the case of above, this is `addin name everyone sees`. **IMPORTANT NOTE: Do not edit the TOOLTAG part of this line in the template. Only edit the [addin name everyone sees] part. TOOLTAG automatically adds the version info to the naming.**

`AdDinIDDoNotTouCHY`: This key says Do Not Touchy for a reason. Therefore, do not touchy. This value is pulled from input `addin_id` from the action. Typically it is something along the lines of `com.companyname.uniquetoolname`. In the event you modify this file and have multiple lines that need to pull this value, Project JAAB will replace the `AdDinIDDoNotTouCHY` key with the `addin_id` input whereever it is used. Feel free to use it in another line, should you choose.

`[name of jsl file in addin]`: edit this to be the name of the .jsl you are packaging as the addin. This is the .jsl code you have written and want JMP to recognize and execute when a person clicks this button. Because of the path information above, you only need the jsl name here. Make sure to keep the `.jsl` at the end so that JMP will recognize the code to execute. If this does not match your jsl filename, the button becomes effectively useless.

`[Write a caption to describe the addin]`: edit this to say whatever you want! When a person hovers in JMP over the `[addin name everyone sees]` this caption will appear to describe what the addin does. Try to keep this as a relatively short description. 1 sentence tops. The above example shows "Write a caption to describe the addin"

`[Choose a symbol]`: Here you can get creative about what you want your symbol to look like! This is the symbol that exists next to `addin name everyone sees`. JMP has many `"builtin"` ones you can leverage by using the name that JMP recognizes. If you are unsure, an Addin exists here that can help you pick what this is. Find your symbol from this addin, click it, find the Icon Name, and paste that into the `[Choose a symbol]` location. PRO TIP: You can also set this to be your own image, should you have an image you just can't live without. Change `<jm:icon type="builtin">[Choose a symbol]</jm:icon>` to `<jm:icon type="path">$ADDIN_HOME(AdDinIDDoNotTouCHY)\[name of JPG inside the addin]</jm:icon>`. Replace `[name of icon file inside the addin plus extension]` with the name of the image and it's extension that is saved inside the repository. Viola! You will have your own image as the icon for your addin.

Now your `jmpcust_txt_file` input is complete! ✨ ✨

Optional Prerequisites

A .ini file

The .ini file will be required to include files from other repositories for packaging but it is overall optional. To use, the file needs to be added to the .github/workflows area of the repo using this action. The name of this file is in an input for `external_files`.

This is the format of the .ini file when importing 1 file:

```
[external_files]
; ***how to write to get the file you want from other repositories***
; arbitary number = name of owner of repostiory, name of repository, name of the
file you want exactly as it's written in the repo, name to call the file in the
addin, folder to add it into the addin, repository release version number
; arbitary number means to start with 1 and continue by adding 1 to the number for
every new file you'd like to include.
; if the folder to place the file in is in the Main folder, write Main in the
foldername location. Otherwise, a new folder will be made inside the addin by the
name written here.
; repository release version is optional and defaults to latest release.
; inputs are separated by commas.
1 = owner, repo, file-to-include, name-to-call-it, foldername, version-number
```

This is the format of the .ini file when importing more than 1 file:

```
[external_files]
; ***how to write to get the file you want from other repositories***
; arbitary number = name of owner of repostiory, name of repository, name of the
file you want exactly as it's written in the repo, name to call the file in the
addin, folder to add it into the addin, repository release version number
; arbitary number means to start with 1 and continue by adding 1 to the number for
every new file you'd like to include.
; if the folder to place the file in is in the Main folder, write Main in the
foldername location. Otherwise, a new folder will be made inside the addin by the
name written here.
; repository release version is optional and defaults to latest release.
```

```
; inputs are separated by commas.
1 = owner, repo, file-to-include, name-to-call-it, foldername, version-number
2 = owner, repo, file-to-include, name-to-call-it, foldername, version-number
```

All inputs are case sensitive. Start with the file number starting with `1 =`. Continue to increment up for the number of files you'd like to add. Each line can be a separate owner, repo, file, etc. of your choosing. Replace where it says `owner` with the owner of the github repository for the file you wish to include. Next, replace where it says `repo` with the name of the repository that houses the file. After, write the name of the script that's located in the remote repository that you would like to include in the `file-to-include` location. Don't forget the extension type (ex. .jsl). In a similar format, write the name to call the script in your addin in the `name-to-call-it` location. Don't forget the extension type here as well. Next write the folder name in the `foldername` location. If you would like this in the main folder with your .jsl script, write "main" here. Otherwise, put whatever name you wish! Lastly, replace `version-number` with the version number of the file you'd like from the remote repository. This defaults to latest if it does not exist.

Now your `external_files` input is complete! ✨ ✨

Inputs

Name	Description	Default	Required
token	security token	N/A	false but needed for private repos
addin_id	the addin id. usually in the format com.company.addin	N/A	true
addin_name	the name of the addin	N/A	true
jmpcust_txt_file	the filename for the text file in Mandatory Prerequisites to create the addin menu	N/A	true
tag_suffix	whether the final addin includes version tag in the filename. true to include. false to exclude.	true	N/A
author	the author of the addin.	""	N/A
owner_repo	repo owner and name using the addin	${{github.repository}}	N/A
run_id	run reference id created from publishing	${{github.event.release.id}}	N/A
make_meta_file	boolean to make the meta data file for auto updates. true to make it. false to not make it.	false	N/A
pub_name	the name of the publishedaddins.jsl (added to metadata file and used for auto updates/deployment)	publishedaddins.jsl	N/A

Name	Description	Default	Required
final_pub_path	the pathway where the publishedaddins.jsl is saved (added to metadata file and used for auto updates/deployment)	""	N/A
external_files	the .ini file in the Optional Prerequisites for including external files	N/A	false

N/A is set as there's a default included in the .yml and thus a `with` is not required in the usage for this input. These defaults can be reset with a `with` (see usage).

Usage

Versioning for the release tag for use with Project-JAAB: Project-JAAB requires semi-specific versioning when publishing a release in order to accurately record the version information inside the JMP addin files.

The general rules are that numbers can be as long as you want and have as many places as you want, as long as they are separated by periods and no other punctuation with the exception of an RC, Beta, or Alpha designation. Additionally, `v` for version can be included at the beginning if you would like, but it is not required.

Project-JAAB allows the versioning to include a `RC` (standing for Release Candidate), `Beta`, or `Alpha` designation at the end of the version, should this fit your workflow. Just remember that `RC`, `Beta`, and `Alpha` also require a number designation to denote it's own version traceability. An example of this is that version `1.0.0` may be the version that will eventually be the production version, but the `RC` we are releasing is the 1st one. Thus, this version would be `1.0.0-RC1`. `RC`, `Beta`, and `Alpha` are taken into account in the versioning numbering inside both the addin.def file as well as the meta file when used.

Examples of acceptable versioning: `v1.0.0`, `1.0.0`, `1.0`, `v1.0`, `1.0-RC1`, `v1.0-Beta1`

Unacceptable versioning: `1-0-0`, `1.0-0`, `v1/0/0`, `1.0-RC`, `v1.0-Beta 1_0`

With all inputs default and only true required:

```
on:
  release:
    types:
      - published
jobs:
  build-addin:
    runs-on: ubuntu-latest
    steps:
      - name: Example Addin Build with Project JAAB
        id: Project-JAAB
        uses: sage-darling/Project-JAAB@v1.0
        with:
          addin_id: com.company.addin_name
          addin_name: addin_name
          jmpcust_txt_file: myfile.txt
```

With true required inputs, with autodeployment, a .ini file to include and not wanting the tag suffix excluded from the filename.

```
on:
  release:
    types:
      - published
jobs:
  build-addin:
    runs-on: ubuntu-latest
    steps:
      - name: Example Addin Build with Project JAAB
        id: Project-JAAB
        uses: sage-darling/Project-JAAB@v1.0
        with:
          addin_id: com.company.addin_name
          addin_name: addin_name
          jmpcust_txt_file: myfile.txt
          make_meta_file: true
          tag_suffix: false
          final_pub_path: D:/Users/Rando/SomeFolder/ProdDeploymentFolder/MetaData/
          external_files: config.ini
```