master 2 branches 13 tags Go to file Code

montegoulding Merge pull request #6 from ange... fa7d263 on Jun 15, 2017 132 commits

lcVCS.livecode.vcs	Widget support improvements	4 years ago
lcVCSPlugins	Widget support improvements	4 years ago
.gitignore	Implemented DeployRelease to create the archive and...	7 years ago
DCO.txt	Added documentation for updating lcVCS and contrib...	7 years ago
LICENSE.txt	LICENSE & README	8 years ago
LICENSE_HEADER.txt	Delete mainstacks that weren't in memory before an e...	7 years ago
README.md	Corrected haeders	3 years ago
lcVCSProject.json	Added cRevGeneralAll plugin, fixed issue with filenam...	7 years ago

About

Stackfile export/import for VCS support in LiveCode

goulding.ws

Readme

View license

Releases

13 tags

Contributors 3

montegoulding montegou...

angerangel angerangel

gitter-badger gitter-badger

README.md

lcVCS

Stackfile export/import for VCS support in LiveCode

lcVCS exports and imports stack files to a structured folder of json, script and image files.

Author

Monte Goulding - monte@goulding.ws

M E R Goulding Software Development http://goulding.ws

mergExt LiveCode Extensions http://mergext.com

Project File

A lcVCSProject.json file is used to order the stackFiles in a project and manage the build path of the stackFiles. It is posible to use multiple project files in a single code repository so the projects can be imported and exported independently.

Folder structure

Each object is given a UUID during the stackFile export. A stackFile is exported to a directory with the name of the stackFile and a .vcs extension. Inside this directory the data for each object can be found at a relative path created from the first two characters of the UUID as a parent directory and the rest of the UUID as a child directory.

In each folder is a properties.json file containing the class, uuid, properties and custom properties of the object. The script of the object (if there is one) is saved as a separate file script.utf8 file. If the object is an image then the image is saved as a file named image with an extention of the paintCompression of the image object.

An example path to an object properties file is therefore: stack.livecode.vcs/f7/8b57e1-8aed-4f01-9022-c3d47e231a2e/properties.json

lcVCS also uses lists of UUIDs in files to reference child objects. The root directory has a mainstack file with a single UUID. With that lcVCS can find the object path of the mainstack. Stack directories will have a cards file and may have a substacks file and a sharedGroups file. Cards and groups may have a layers file. Only the direct children of the control are listed in these files and they are in layer order. Because of the complexity of dealing with merge conflicts on these files lcVCS handles them in a specific way while importing. If it finds conflict markers it will strip them from the file. This may result in UUIDs being listed multiple times so the first occurance is the one that will be used. It may also result in UUIDs from deleted or moved objects being listed so if the object directory doesn't exist these are ignored.

All data with the exception of custom properties is exported as utf8 so that it's possible for cross platform developers to work on the files without encoding issues creating conflicts. Any custom property that contains non ASCII characters is base64Encoded and wrapped in `base64Decode()`.

This structure has been designed to resolve many of the issues there are with merging branches of stack files. Most of these issues relate to the fact that object ID conflicts are nigh on unresolvable. The project introduces a *uVersion* custom property set with the object UUID and its current ID as keys. When exporting if the ID of the object has changed a new UUID is generated. The *uVersion* custom property set is then redundant and not exported. The export includes object IDs, however, object IDs should not be used as constants to refer to objects because when the stack is regenerated the ID may be different if there was merge of two branches that created objects. Properties that record IDs (icons, patterns, behaviors) are exported instead as UUIDs so they can be resolved correctly during the import. You may think this is crazy but just tell me what should happen when two branches create a new control and it's assigned the next available IDs (as LiveCode does) which therefore creates an id conflict... which one should be given the correct id?

A plugin system supports the export and import of object references found in custom property sets.

Reducing false positive conflicts

Because LiveCode stackfiles retain session changes between saves it's important to clear anything that's likey to cause a false positive conflict between two versions. A good example of this is a resizable stack with a resizeStack handler that moves objects. This could cause a conflict on most of the controls on the stack. What we want to do is reset the stack to it's default state during the export. To help with this each object is dispatched a message `lcVCSExport`. lcVCS unlocks messages to dispatch the command so you can handle the message in the stack script and resize the stack triggering your resizeStack handler. Because the message is sent to every object the objects in the lower parts of the message heirarchy will recieve it multiple times. In any control that has a child control you will want to script your lcVCSExport handler like this:

```
on lcVCSExport
    if the target is not me then exit lcVCSExport
end lcVCSExport
```

Rules for successful use of lcVCS

- If you use IDs in scripts anywhere (such as setting the icon of a button) then replace it with a name based reference.
- Implement lcVCSExport handlers to ensure everything is set back to defaults during the export.
- If you have any custom objects or libraries that maintain custom property sets that store IDs then implement a plugin to support it. The plugin api is quite simple and there's a number of examples demonstrating their use. If possible contribute the plugin back to the project so we can maintain them centrally.
- Limit object references between stackFiles where possible. Where not possible ensure that there are no circular references. For example, stackFile A has an object reference to an object in stackFile B while stackFile B has an object reference to an object in stackFile A.
- Order your stackFiles so that the stackFiles that have object references to objects in other stackFiles are imported and exported after the stackFiles they refer to. The stackFiles list in the lcVCSProjects stack can be re-ordered by drag and drop.
- If you have object references to objects that no longer exist (such as icons referencing deleted images) clear the property.

Dependencies

- mergJSON which is a dual licensed (GPL/Commercial) JSON external for implementing JSONToArray and ArrayToJSON.
- LiveCode 6.5.1

Installation

You will need mergJSON and the lcVCS stackFiles. Compiled versions can be downloaded from here. If you don't have a mergExt account you can sign up for a free one.

mergJSON

- If you don't have a user extensions/Externals folder then create one. To find your user extensions folder go to LiveCode prefs.
- If you don't have an Externals.txt file in the user extensions/Externals folder then create one
- OS X
  - Copy mergJSON.bundle into your user extensions/Externals directory
  - Add mergJSON,mergJSON.bundle to your Externals.txt file in you user extensions/External directory
- Linux
  - Copy mergJSON.so into your user extensions/Externals directory
  - Add mergJSON,mergJSON.so to your Externals.txt file in you user extensions/External directory
- Windows
  - Copy mergJSON.dll into your user extensions/Externals directory
  - Add mergJSON,mergJSON.dll to your Externals.txt file in you user extensions/External directory

lcVCS

- Copy lcVCS.livecode into your user extensions/Plugins directory
- Copy the lcVCSPlugins folder into your user extensions/Plugins directory

Keeping lcVCS updated via git

Get the repo if you think it's unlikely you will contribute

- Clone the repo using `git clone https://github.com/montegoulding/lcVCS.git`
- Open lcVCS and click the + icon on the bottom left of the projects stack.
- Browse to and choose the cloned repo
- You should now see lcVCS in the projects list

Get the repo if you think you will probably contribute

- Go to GitHub
- Create an account if you don't have one
- Click on the button to fork lcVCS
- Clone the repo using `git clone https://github.com/<your username>/lcVCS.git` or preferably setup SSH keys on GitHub and use `git clone git@github.com:<your username>/lcVCS.git`
- Enter the repo `cd lcVCS`
- Add my repo as upstream with `git remote add upstream https://github.com/montegoulding/lcVCS.git`
- Make the `master` branch track `upstream/master` rather than `origin/master` with `git branch --set-upstream upstream/master master`

Updating the repo

```
cd lcVCS
git pull
```

- Open lcVCS by choosing it from the Development > Plugins menu
- Choose lcVCS from the projects list on the left of the projects stack
- Click the Import Project button
- When the import is complete the projects list will appear blank so just click the stack to trigger resumeStack and it will show the projects again

Contributing to lcVCS

- Follow the instructions above to fork lcVCS
- Branch from master before committing so you will always be able to pull master from the upstream remote
- Commit your changes using the `--sign-off` or `-s` option
- Send a pull request from your branch against the upstream remote
- By sending a pull request you are agreeing to the *Developer's Certificate of Origin* found in `DCO.txt`

GitHub Search

lcVCS includes a GitHub search for lcVCS based projects. If you want your project to appear in the list you just need to include lcVCS and LiveCode somewhere in the README of the repo. Just write something like `This is project uses lcVCS to enable VCS support in a LiveCode project`.