load instructor

Copies instructor files on inaccessable drives to the temporary directory and automates instructor file loading

1. Introduction

If you have been considering writing an instructor file for your tool, you are probably aware of Sketchup's getInstructorContentDirectory method. This is a method used by Sketchup to return a path to your 'instructor' folder. This is the folder that contains an index.html file to display in the 'Instructor' window (Window->Instructor). However, the main issue with getInstructorContentDirectory is that it works on a relative path from the 'helpcontent/tool/' directory. This means that if a user of your tool installs the tool in a volume that is different to the volume that Sketchup runs from (lets say 'F:'), getInstructorContentDirectory will not be able to access the instructor directory. For this reason, I've written **Load Instructor**.

Load Instructor is a ruby script that will copy your instructor folder to the system's temporary directory when the instructor folder can not be accessed by a relative path. **Load Instructor** will then use getInstructorContentDirectory to return a relative path to your instructor folder, regardless of whether your instructor folder was copied to the temporary directory or left in its original location.

What this means is that, with **Load Instructor,** your tool can be completely portable across platforms.

Load Instructor is written as a Mixin Module. If this is completely foreign to you, don't worry. You don't need to know much about Mixin Modules to use the script. In short, all you need to know is that a Mixin Module enables you to "mix-in" the module's instance method(s) (ie. **Load Instructor's** methods) to a custom class definition (ie. your Tool).

Below I've written two examples of use: a Basic Use, which assumes that you'll have access to Sketchup's 'Plugins' and 'Tools' folders; and an Advanced Use, which assumes that you don't but you still want to make your script portable.

2. Basic Use

These instructions show you how to setup the **Load Instructor** underneath one of Sketchup's $LOAD_PATH's (like 'Plugins' or 'Tools') and then include and run the **Load Instructor** from your tool.

2.1 Create a '**mixin**' subdirectory under one of the $LOAD_PATH paths. The 'Plugins' folder is fine. (If you don't know what $LOAD_PATH paths are available, open the Ruby Console and type: $LOAD_PATH, then hit enter or return)

2.2 Copy the file '**load_instructor.rb**' into that '**mixin**' directory.

2.3 Make sure you have an '**instructor**' folder beneath your plugin's subdirectory, and make sure this folder includes a file '**index.html**'. (for example: if your script kept all its extra files in the directory 'my_cool_script', your 'instructor' folder should be in this directory, like: 'my_cool_script/instructor/')

Chapters

On this page

Check for Updates

Visit [code] Load Instructor at forums.sketchucation.com

Module Author

Niall Campbell (bentleykfrog)

load instructor

Copies instructor files on inaccessable drives to the temporary directory and automates instructor file loading

2.4 Add the lines to require() and include() to include the **Load Instructor** code into your tool, like:

```
module MyModule
  class MyCoolScript
    # Include the Load Instructor code
    require('mixin/load_instructor.rb')
    include( Mixin::LoadInstructor )
    # The rest of your code
  end #class MyCoolScript
end #module MyModule
```

2.5 Call the lins_setup('your_tools_subdirectory') method from inside your Tool's initialize() method, like the example below:

```
module MyModule
  class MyCoolScript
    # Include the Load Instructor code
    require('mixin/load_instructor.rb')
    include( Mixin::LoadInstructor )

    def initialize
      # Call the Load Instructor with your tools
      # subdirectory as the first argument
      lins_setup('my_cool_script')

      # The rest of your tool initialize code goes here
    end #def initialize
    # The rest of your code
  end #class MyCoolScript
end #module MyModule
```

By adding that code to your Tool's initialize() method, you've now automated the copy of your instructor files to the temporary directory. You've also automated the getInstructorContentDirectory method, so there's no need to use it in your Tool.

As an additional note, the previous example assumes the following file structure:

```
[any $LOAD_PATH]/mixin/load_instructor.rb
[any $LOAD_PATH]/my_cool_script.rb
[any $LOAD_PATH]/my_cool_script/some_image.png
[any $LOAD_PATH]/my_cool_script/etc..
[any $LOAD_PATH]/my_cool_script/instructor/index.html
[any $LOAD_PATH]/my_cool_script/instructor/etc...
```

#NB: Note that the first argument passed to lins_setup() matches the folder name 'my_cool_script' and the folder 'my_cool_script' contains the 'instructor' folder with all your instructor files inside. If you store your scripts files in a different directory, pass this directory name as the first argument to lins_setup().

Chapters

On this page

Check for Updates

Visit [code] Load Instructor at forums.sketchucation.com

Module Author

Niall Campbell (bentleykfrog)

load instructor

Copies instructor files on inaccessable drives to the temporary directory and automates instructor file loading

3. Advanced Use

Now you're probably thinking that if your tool is in the 'Plugins' or 'Tools' folder that there's no need to copy the instructor files to the temporary directory, as those folders are on the same drive as Sketchup. Well you're right, but some script loading plugins add additional paths to the $LOAD_PATH array. These paths can be on any drive, so there's still a need to copy the instructor files to the temporary directory.

Some other script loading plugins attempt to load your plugin into Sketchup after Sketchup has started up, like Alex's excellent Plugin Loader.

These script loading plugins are a nice way to get around certain permission issues that may result when an admin grants access for a user to modify files inside the Program Files/Program Files (x86) directories (the directories that 'Plugins' and 'Tools' both lie in. Instead, a user can have a plugin folder on a separate part of the drive, or even a completely different drive to 'C:', so they can still add plugins to Sketchup, and not risk modifying any of the system's delicates inside the Program Files/Program Files (x86) directories.

Its also possible that a user wants to keep one common plugins folder for all versions of Sketchup they run. This means that the user only has to add a plugin to one directory, to add it to all versions of Sketchup. It also means (if they store their files on a portable storage device) that their plugins can be loaded on any computer they wish to use.

So keeping your script portable and accessible from outside the 'C:' drive is important. The instructions below will show you how to set up the **Load Instructor** in a file that's relative to your script, and then run **Load Instructor** with paths relative to your script, so that your script becomes entirely portable.

3.1 Create a '**mixin**' directory beneath your plugin's subdirectory. (For example: if your script keeps all its extra files in the directory 'my_cool_script', the 'mixin' folder should be in this directory, like: 'my_cool_script/mixin/')

3.2 Copy the file '**load_instructor.rb**' to that '**mixin**' directory

3.3 Make sure you have an '**instructor**' folder beneath your plugin's subdirectory, and make sure this folder includes a file '**index.html**'. (for example: if your script kept all its extra files in the directory 'my_cool_script', your 'instructor' folder should be in this directory, like: 'my_cool_script/instructor/')

Chapters

On this page

Check for Updates

Visit [code] Load Instructor at forums.sketchucation.com

Module Author

Niall Campbell (bentleykfrog)

load instructor

Copies instructor files on inaccessable drives to the temporary directory and automates instructor file loading

3.4 Add the lines to require() and include() to include the **Load Instructor** code into your tool, like:

```
module MyModule
  class MyCoolScript
    # Form a path from this script to load_instructor.rb
    script_directory = File.dirname( __FILE__ )
    lins_directory = 'my_cool_script/mixin/load_instructor.rb'
    # Include the Load Instructor code
    require(File.join( script_directory , lins_directory ))
    include( Mixin::LoadInstructor )
    # The rest of your code
  end #class MyCoolScript
end #module MyModule
```

This code first stores the directory that contains your ruby script in the script_directory variable, then forms a path to the load_instructor.rb file from this directory. The next line simply joins the two paths together to form an absolute path to load_instructor.rb, which is passed to require.

3.5 Now call the lins_setup() method from inside your tool's initialize() method, but this time pass it a second argument representing the absolute path to the instructor folder, like below:

```
module MyModule
  class MyCoolScript
    # Form a path from this script to load_instructor.rb
    script_directory = File.dirname( __FILE__ )
    lins_directory = 'my_cool_script/mixin/load_instructor.rb'
    # Include the Load Instructor code
    require(File.join( script_directory , lins_directory ))
    include( Mixin::LoadInstructor )

    def initialize
      script_dir = File.dirname( __FILE__ )
      ins_dir = 'my_cool_script/instructor'
      instructor_folder = File.join( sript_dir , ins_dir )
      # Call the Load Instructor with your tools
      # subdirectory as the first argument and the
      # absolute location of your instructor folder
      # as the second argument
      lins_setup( 'my_cool_script' , instructor_folder)
      # The rest of your tool initialize code goes here
    end #def initialize
    # The rest of your code
  end #class MyCoolScript
end #module MyModule
```

Notice that we still pass the tools subdirectory as the first argument. In this case its not affecting the absolute location of the instructor folder, rather its used to create a folder in the temporary directory to store the copy of your 'instructor' folder beneath so it doesn't overwrite any other plugin's copies of their 'instructor' folder.

Chapters

On this page

Check for Updates

Visit [code] Load Instructor at forums.sketchucation.com

Module Author

Niall Campbell

(bentleykfrog)

load instructor

Copies instructor files on inaccessable drives to the temporary directory and automates instructor file loading

Now there's a difference in the file structure in this example, so please note: this example assumed the following file structure:

```
[ANY PATH]/my_cool_script/mixin/load_instructor.rb
[ANY PATH]/my_cool_script.rb
[ANY PATH]/my_cool_script/some_image.png
[ANY PATH]/my_cool_script/etc..
[ANY PATH]/my_cool_script/instructor/index.html
[ANY PATH]/my_cool_script/instructor/etc...
```

The subtle difference is that the 'load_instructor.rb' file is inside the folder '/my_cool_script/mixin' instead of just '/mixin'. This means that when you copy the '/my_cool_script' folder and the file 'my_cool_script.rb' to a new location, the **Load Instructor** will be copied with it, ensuring that your tool doesn't break when someone copies or moves it.

So now you have a completely portable version of your script, that can be loaded from any drive, without breaking the link to your 'instructor' folder.

4. Additional Code Snippets

There are a few code snippets that could be helpful for you if you want to do more complex things with **Load Instructor**. I've documented them here for your reference.

4.1 You can retrieve the instructor folder and temp folder location paths just after your call to lins_setup(), like:

```
# Returning the folder variables
instructor_folder = @instructor_folder.dup
temp_folder = @temp_folder.dup

# Setting the folder variables
@instructor_folder = instructor_folder
@temp_folder = temp_folder
```

Retrieving and setting these values might be helpful to you if you intend to use these values later in your script. It may also be helpful if you intend to modify the default values of the instructor and temp folders, or if you intend to check whether or not these files exist.

#NB: @temp_folder will return @instructor_folder when your script is running on a Mac (all volumes can be accessed relatively on a Mac so there's no need to copy files to the temporary directory)

#NB: On pc, @temp_folder will return the path to where it **WILL** store the instructor folder, regardless of whether it needs to copy the instructor folder to this location or not. See (**4.3**) for the method to check if the instructor folder needs to be copied to the temporary directory.

Chapters

On this page

Check for Updates

Visit [code] Load Instructor at forums.sketchucation.com

Module Author

Niall Campbell (bentleykfrog)

load instructor

Copies instructor files on inaccessable drives to the temporary directory and automates instructor file loading

#NB: Note that when you call lins_setup() with one argument like lins_setup('my_cool_script'), **Load Instructor** checks in what $LOAD_PATH this file exists. If it can't find the file it throws a script error. Also note that when you call lins_setup() with a second argument like lins_setup('my_cool_script', instructor_folder), **Load Instructor** trusts that there is a folder at the path represented by the variable instructor_loader. In other words, when you set the second variable, **Load Instructor** doesn't check whether or not this file exists. Also note, that when **Load Instructor** creates the temporary directory to store the instructor folder, first it checks if a directory already exists, and if it does (to preserve consistency) it deletes this directory and the files in it, then loads the instructor folder into this directory.

4.2 lins_setup() has a third argument that represents the path to the temporary directory if you want to use a custom temporary directory location. This third argument must be an absolute path to the directory, like in the example below:

```
script_dir = File.dirname( __FILE__ )
ins_dir = 'my_cool_script/instructor'
instructor_folder = File.join( sript_dir , ins_dir )
# Store a custom path to the temporary directory
# by first determining the operating system and
# then Sketchup's temporary directory variable
if (not RUBY_PLATFORM =~ /(mswin|mingw)/i) # on a Mac
  tmp_dir = ENV["TMPDIR"]
else
  tmp_dir = ENV["TEMP"]
end #if
custom_subdir = 'my_cool_script/instructor_wazoo'
temp_folder = File.join( tmp_dir , custom_subdir )
# Call the Load Instructor with your tools
# subdirectory as the first argument, the
# absolute location of your instructor folder
# as the second argument and the custom temporary
# directory as the third argument
lins_setup( 'my_cool_script' ,instructor_folder,temp_folder)
```

4.3 You can return the path to the instructor temporary folder from within and outside your script by calling the respective methods: lins_return_temp_instructor_folder_name() and temp_instructor_folder(). These methods return the @instructor_folder if the script is run on a Mac, the instructor temporary folder if the instructor folder and the temporary folder exist on different volumes on PC, and false if the instructor folder and the temporary folder exist on the same directory on PC.

```
# Calling from within your class
temp_folder = lins_return_temp_instructor_folder_name()
# Calling from outside your class
my_script_object = MyModule::MyCoolScript.new
temp_folder = my_script_object.temp_instructor_folder()
```

Chapters

On this page

Check for Updates

Visit [code] Load Instructor at forums.sketchucation.com

Module Author

Niall Campbell (bentleykfrog)

load instructor

Copies instructor files on inaccessable drives to the temporary directory and automates instructor file loading

4.4 Finally, you can get the relative path from the Sketchup 'helpcontent/tool' folder to the tools instructor folder from inside and outside your class by calling the respective methods: lins_return_relative_path() or relative_instructor_folder(). Both of these methods will return the relative path to the instructor folder, regardless of whether it was copied to the temporary folder or not.

```
# Calling from within your class
temp_folder = lins_return_relative_path()
# Calling from outside your class
my_script_object = MyModule::MyCoolScript.new
temp_folder = my_script_object.relative_instructor_folder()
```

5. Credits

I need to thank Dan Rathbun from the Sketchucation forum for his generous help and guidance with this script. He should be credited for (to say the least):

- Writing the Mixin Module container for the script
- Defining getInstructorContentDirectory from within the Module
- The script's platform checking
- Public methods to retrieve the instructor folder & temp folder
- Very helpful advice with .rbs integration
- $LOAD_PATH checking for the plugin's instructor folder
- Init method rename to reduce multiple init method clashes
- And lots of other helpful advice and guidance

Also, credit should go to thomthom (Thomas Thomassen) and TIG from the Sketchucation forum as lins_return_relative_path() is based on thomthom's get_instructor_path() method and TIG's getInstructorContentDirectory method.

I hope you get the best use out of this script

-niall campbell (bentleykfrog)

6. Changelog

= 2011-03-14: Version 1.0
- Official First Release (from beta)

Chapters

On this page

Check for Updates

Visit [code] Load Instructor at forums.sketchucation.com

Module Author

Niall Campbell
(bentleykfrog)