Electric Cloud

PluginLite Introduction

Upon downloading the PluginLite folder in github, you will notice there are a number of contents inside. In this document, we will dive into some of the files in order for you to understand what is needed to create a custom plugin of your own. A prerequisite to understanding this document is a solid understanding of XML, Groovy and ElectricFlow.

The first file we will look at is *plugin.xml* located in *META-INF*

```xml
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <key>PluginLite</key>
  <version>1.0</version>
  <label>PluginLite</label>
  <description>Ultra light plugin customized with DSL</description>
  <category>Utility</category>
  <author>Electric Cloud</author>
  <help>help.xml</help>
  <authorUrl />
  <commander-version min="5.0" />
</plugin>
```

The node `key` is the actual name of the plugin. The `version` node is the version of your plugin – plugins can have multiple versions. The `label` node is the label of the plugin in the Electric Flow GUI. Best practice would be to keep the key and label the same, but they can be different. The `description` node is a brief description of what the plugin does. The `commander-version min` node is the minimum version the plugin can run on. Since DSL was introduced in version 5.0, this will always be the minimum version. When creating your own plugin, *plugin.xml* is the file you use to create to contents of the plugin.

The next file we are going to look at is *projct.xml. Project.xml* contains content related to the project definition file containing procedures and properties to be set automatically during plugin installation. If *project.xml* does not exist, the installation creates an empty ElectricFlow project for the plugin.

Lets take a look at *project.xml*

```
<exportedData version="39" buildLabel="build_3.5_30434_OPT_2010.01.13_07:32:22"
buildVersion="3.5.1.30434">
  <exportPath>/projects/PluginLite-1.0</exportPath>
  <project>
    <projectName>PluginLite-1.0</projectName>
    <propertySheet>
      <property>
        <propertyName>ec_setup</propertyName>
        <expandable>0</expandable>
        <value>use Cwd;
use File::Spec;

my $dir = getcwd;
my $logfile ="";
if(defined $ENV{'QUERY_STRING'}) { # Promotion through UI
       $logfile = "../../$pluginName/ec_setup.log";
       $pluginDir = File::Spec-&gt;rel2abs( "../../$pluginName" );
} else {
       $logfile = "$ENV{'TEMP'}/ec_setup.log";
       $pluginDir = $dir;
}
$commander-
&gt;setProperty("/plugins/$pluginName/project/pluginDir",{value=&gt;$pluginDir});
open(my $fh, '&gt;', $logfile) or die "Could not open file '$logfile' $!";
print $fh "Plugin Name: $pluginName\n";
print $fh "Current directory: $dir\n";

# Evaluate promote.groovy or demote.groovy based on whether plugin is being promoted or
demoted ($promoteAction)
local $/ = undef;
# If env variable QUERY_STRING exists:
if(defined $ENV{'QUERY_STRING'}) { # Promotion through UI
       open FILE, "../../$pluginName/dsl/$promoteAction.groovy" or die "Couldn't open
file: $!";
} else {  # Promotion from the command line
       open FILE, "dsl/$promoteAction.groovy" or die "Couldn't open file: $!";
}
my $dsl = &lt;FILE&gt;;
close FILE;
my $dslReponse = $commander-&gt;evalDsl($dsl,
               {parameters=&gt;qq(
                       {
                               "pluginName":"$pluginName"
                       }
               )}
)-&gt;findnodes_as_string("/");
print $fh $dslReponse;

close $fh;

# Create log file output property
open LOGFILE, $logfile or die "Couldn't open file: $!";
my $logFileContent = &lt;LOGFILE&gt;;
my $propertyResponse = $commander-
&gt;setProperty("/plugins/$pluginName/project/ec_setup.log",
                      {value=&gt;$logFileContent}
       );
close LOGFILE;</value>
      </property>
    </propertySheet>
  </project>
</exportedData>
```

The two nodes to pay attention to here are `project name` and `property name.` When `project name` is referenced the string inside the node will reference the actual name of the project being created in ElectricFlow. When creating your own plugin, modify `project name` to create your own unique project. The node `property name` is responsible for the installation, promotion and demotion of the plugin you create. The node calls a perl script `ec_setup.pl` to do this.

The next files we will look at are `promote.groovy` and `demote.groovy`. Lets take a look at `promote.groovy` first

```
def pluginName = args.pluginName
def pluginKey = getProject("/plugins/$pluginName/project").pluginKey
def pluginDir = getProperty("/server/settings/pluginsDirectory").value + "/" +
pluginName
// END Variables

// Sample plugin project content.  pluginName can be replaced by a name
// to create a non-plugin project
project pluginName,{

        // Make this plugin visible in all contexts
        property "ec_visibility", value: "all" // Legal values: pickListOnly, hidden,
all

        procedure "Sample Procedure",{

                // Server level property that exposes this procedure for use in pipeline
tasks,
                // application processes and component processes (pick lists). Note that
                // procedureName an intrinsic variable for the procedure closure
                property "/server/ec_customEditors/pickerStep/$pluginKey -
$procedureName",
                        value:
                                """\
                                        <step>

        <project>/plugins/$pluginKey/project</project>
                                                <procedure>$procedureName</procedure>
                                                <category>Utility</category>
                                        </step>
                                """.stripIndent(),
                        description: "A sample procedure"
                // TIP: Remove this property in demote.groovy
                // END property

                step "Hello", shell: "ec-perl",
                        // Get step content from a file in this plugin directory
                        command: new File(pluginDir + "/dsl/steps/Hello.pl").text
        }
}
```

At a high level, `promote.groovy` is creating a procedure within a project in ElectricFlow called Sample Procedure. Then, a step is created called `Hello` which calls a shell script `Hello.pl`. The contents of `Hello.pl` is as follows:

```
print "Hello from ec-perl\n";
```

This script will print “`Hello from ec-perl`” in a step.

Lets take a look at `demote.groovy`.

```
def pluginName = args.pluginName
def pluginKey = getProject("/plugins/$pluginName/project").pluginKey
def pluginDir = getProperty("/server/settings/pluginsDirectory").value + "/" +
pluginName

deleteProperty propertyName: "/server/ec_customEditors/pickerStep/$pluginKey - Sample
Procedure"

return "Demoting plugin"
```

The purpose of `demote.groovy` is to demote the plugin when needed.

The next and last file we will look at is `createPlugin.ps1`. The powerhsell script, `createPlugin.ps1` automates the entire process behind creating a plugin. *Note*: you **must** provide the Groovy syntax of what you’d like the plugin to do in `promote.groovy`. The file `createPlugin.ps1` will merely create, install and promote the plugin automatically in ElectricFlow instead of doing it manually.

```
param (
    [string]$pluginKey = "PluginLite",
        [string]$version = "1.0",
        [string]$description = "Ultra light plugin customized with DSL"
)

$pluginName = "${pluginKey}-${version}"

function Add-Zip
{
    param([string]$zipfilename)
    if(-not (test-path($zipfilename)))
    {
        set-content $zipfilename ("PK" + [char]5 + [char]6 + ("$([char]0)" * 18))
        (dir $zipfilename).IsReadOnly = $false
    }

    $shellApplication = new-object -com shell.application
        $zipfilenameFull = get-childitem $zipfilename
    $zipPackage = $shellApplication.NameSpace($zipfilenameFull.FullName)

        #$files = Get-ChildItem -Path $srcdir | where{! $_.PSIsContainer}

       write-host "Zipping up directories"
    foreach($file in $input)
    {
               write-host $file.FullName
                      $zipPackage.CopyHere($file.FullName)
                      while($zipPackage.Items().Item($file.name) -eq $null){
                                    Start-sleep -seconds 1
                      }
    }
}

# Update project.xml with ec_setup.pl
write-host "Updating project.xml with ec_setup.pl"
$ec_setup = Get-Content ec_setup.pl
$a = Select-Xml -Path .\META-INF\project.xml -XPath '//value[../propertyName/text() =
"ec_setup"]'
$a.Node.'#text'=$ec_setup -join "`n"
$a.Node.OwnerDocument.Save($a.Path)

write-host "Updating plugin.xml with key, version, label, description"
# Update plugin.xml with key, version, label, description
$b = Select-Xml -Path .\META-INF\plugin.xml -XPath '//key'
$b.Node.'#text' = $pluginKey
$b.Node.OwnerDocument.Save($b.Path)

$b = Select-Xml -Path .\META-INF\plugin.xml -XPath '//version'
$b.Node.'#text' = $version
$b.Node.OwnerDocument.Save($b.Path)

$b = Select-Xml -Path .\META-INF\plugin.xml -XPath '//label'
```

```
$b.Node.'#text' = $pluginKey
$b.Node.OwnerDocument.Save($b.Path)

$b = Select-Xml -Path .\META-INF\plugin.xml -XPath '//description'
$b.Node.'#text' = $description
$b.Node.OwnerDocument.Save($b.Path)

write-host "Removing old zip and jar files"
del "${pluginKey}.zip" -ErrorAction SilentlyContinue
del "${pluginKey}.jar" -ErrorAction SilentlyContinue
# Add all but .git directories to zip file
dir -exclude .git | ? {$_.mode -match "d"} | Add-Zip "${pluginKey}.zip"
Move-Item "${pluginKey}.zip" "${pluginKey}.jar"

write-host "Demoting old plugin"
ectool promotePlugin "$pluginName" --promoted false
write-host "Uninstalling old plugin"
ectool uninstallPlugin "$pluginName"
write-host "Installing new plugin"
ectool installPlugin "${pluginKey}.jar"
write-host "Promoting new plugin"
ectool promotePlugin "$pluginName"

write-host "Output from ec_setup.pl promoting plugin:"
type $Env:TEMP/ec_setup.log
```

The only part of this script you should modify is the `[string]$pluginKey`, `[string]$version` and `[string]$description` variables. These variables are unique to your plugin and if you're operating on a windows server you have the option of running this powershell script to automatically generate, install and promote your plugin. To modify the name of your plugin, simply change `[string]$pluginKey` to the desired plugin name. Apply the same logic for the version and description.

Plugin Lite “Hello World” Linux Tutorial

1.) SSH into your Linux box

2.) Run **git clone https://github.com/electric-cloud/PluginLite**

  i. This will clone the git repo to your server

3.) Run **cd PluginLite**

4.) Run **sudo apt-get install default-jdk** (skip this step if you have JDK installed already. Agree to any overrides.)

  i. This installs as jdk for you to create jar files

5.) Run **jar cvf PluginLiteUPDATE.jar dsl/ META-INF/ pages**

  i. This will create your jar file

  ii. PluginLiteUPDATE will be the name of your jar file. dsl/, META-INF and pages/ are the directories you are jarring. Note: when creating a jar file only jar the directories

6.) Run **ls** to make sure the jar file is in your working directory

7.) Login to ectool (run **ectool login username password**)

  i. If ectool is not working properly, make sure to export the path by running: **export PATH=$PATH:/opt/electriccloud/electriccommander/bin/**

  ii. For a more permanent solution add the following to /etc/profile (**sudo vi etc/profile/)** to the end of the file

  iii. **export PATH=$PATH:/opt/electriccloud/electriccommander/bin/**

  iv. **export PATH**

    a. *You must log out and log back in for this to take effect*

8.) Run **export TEMP=/tmp**

9.) Run **ectool installPlugin PluginLiteUPDATE.jar**

10.) Run **ectool promotePlugin PluginLite-1.0**

Testing Plugin Lite

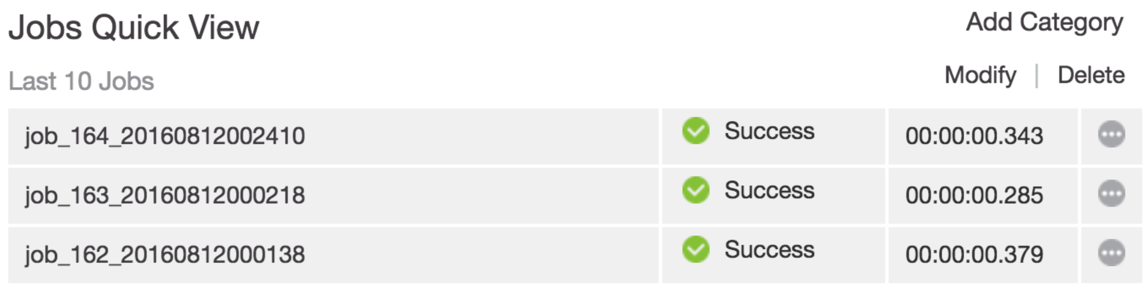
1.) Run **ectool runProcedure "PluginLite-1.0" --procedureName "Sample Procedure"**

  i. This ectool command will run the procedure within Flow, which will trigger PluginLite. See the docs for more info

    1. You can do the same thing within the flow GUI by navigating to https://flow/commander . Click on Projects. Click on PluginLite-1.0. Click ▶

2.) Navigate to https://flow/commander

3.) Click on the last ran job



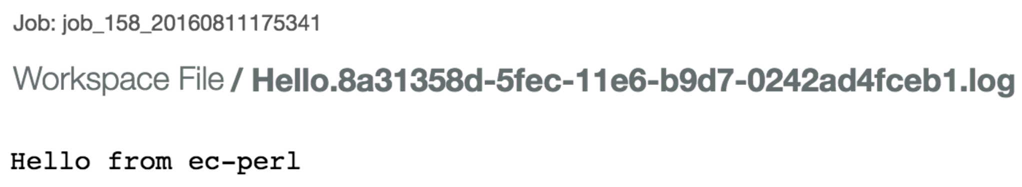
Jobs Quick View

Last 10 Jobs

job_164_20160812002410	✓ Success	00:00:00.343	●
job_163_20160812000218	✓ Success	00:00:00.285	●
job_162_20160812000138	✓ Success	00:00:00.379	●

4.) Click to view the log. This contains output from the plugin

5.) Expected output



Job: job_158_20160811175341

Workspace File / **Hello.8a31358d-5fec-11e6-b9d7-0242ad4fceb1.log**

```
Hello from ec-perl
```