SHOWCASE DOCUMENTATION

PHILIPP HELSPER & MARCEL HEINZ & TOBIAS SCHMIDT

CONTENTS

LIST OF FIGURES

ABSTRACT

This documentation shows how to create a LDS document out of an existing document. Our showcase, which can be found in the LDS github repository, shows hot to set up the Chef-Repo located under `https://github.com/intercity/chef-repo`. For this need we took the original html file and recreated all parts which can be assigned to the LDS language.

ChangeFileLines	ChangeFileRegex	CodeReference
0	3	0
ContinueFile	DeclareReference	DeclareVariable
5	0	7
DeleteFileLines	InsertIntoFile	RootFile
0	0	9

Figure 1: Table of commands used in the file

1 INTRODUCTION

As this documentation shows how to transform a given HTML tutorial to a LDS version which is useable by our tool we decided to present this based on the annotated Chef-Repo file. The main reason for this is that this GitHub file contains a high number of LDS instructions so we are able to present lots of different transformations to the user. Another aspect we want to show is what keywords are interpretable so that we know the following part needed to be transformed in the given way. Possiby another user can transform the file in another way, but the main structure is not interchangeable in their basis.

In the following sections first of all we want to show why this showcase is a good one to demonstrate the procedure of creating an LDS file based on a tutorial previously written down as HTML file. In section 3 we then demonstrate the recreation process before we finally close this dosumentation with a short discussion.

2 DETECTED LDS COMMANDS IN THE CHEF-REPO FILE

Based on the table from our paper[**?**] showing which command has been used how often in which paper we decided to transform the Chef-Repo file based on two reasons. First of all the GitHub file is of a short size so the reader of this documentation is able to directly link the commands mentioned to th position in the GitHub file they belong to. Secondly the tag frequency is very high, so we can demonstrate a high amount of our commands and how they are integrated in the LDS file.

The tag frequency is displayed in the table(1) to show which commands are used for the transformation.

3 TRANSFORMED A HTML FILE FOR USE AS LDS

The Chef-Repo document starts with some constraints and dependencies which must be fulfilled for the tutorial to work. Few Constraints cannot be dealt with such as the need of a newer Linux Distribution such as Ubuntu 14.04. Other dependencies could be handled but are not described well in the document so we cannot transform the file. MySQL for example can be installed on a Linux distribution very easyly. Instead no instructions for this are given in this file. So we do not need to transform this and deal with it as given.

The tutorial itself starts at the Getting started section. The first subsection describes how to set the chef-repo up. The tutorial starts with the instruction to "Clone the repository". This is the first instruction to be transformed to a RootFile. As shown in the codefragment below this first rootfile gets named cloneRepo.sh and is the first section shown in the LDS.

```
<!--""LDS BeginRootFile section="1" filename="cloneRepo.sh"
id="clone repository" platform="UNIX" executor="sh" -->
<code><pre>
git clone git://github.com/intercity/chef-repo.git ~/Code
</code></pre>
<!--""LDS EndRootFile -->
```

As it can be seen easyly all code given in any RootFile needs to be executed from a given path. For this purpose we can declare a Code Reference Block which we are going to add at every beginning of every RootFile. For this only an id needs to be given. This is done by the LDS command below.

```
<!--""LDS BeginDeclareReference id="change to working folder" -->
<pre><code>
cd ~/Code
</code></pre>
<!--""LDS EndDeclareReference -->
```

The following command consists of two parts one RootFile and one BeginCodeReference part. The code reference is declared above and can be accessed using the id it is declared by.

```
<!--""LDS BeginRootFile section="1" filename="runBundle.sh"
id="run bundle" platform="UNIX" executor="sh" -->
<!--""LDS BeginCodeReference id="change to working folder"-->
<!--""LDS EndCodeReference -->
<pre><code>
bundle install
</code></pre>
<!--""LDS EndRootFile -->
```

As the command above the command below is formed in the same way.

```
<!--""LDS BeginRootFile section="1" filename="runLibrarian.sh"
id="run librarian chef" platform="UNIX" executor="sh" -->
<!--""LDS BeginCodeReference id="change to working folder"-->
<!--""LDS EndCodeReference -->
<pre><code>
librarian-chef install
</code></pre>
<!--""LDS EndRootFile -->
```

For the execution of same CodeBlocks sometimes variables need to be declared. Fur this purpose the DeclareVariables command can be used. It is defined with with one or multiple variable names which open InputDialogs so the user can define the value in the way he wants.

```
<!--""LDS DeclareVariables variables="username,host" -->
```

The next RootFile basically is defined as any other. The main difference is that a variable defined before is called. This is implied by the structure $.$-name-$.$. In this special case the defined variables for username and host are used.

```
<!--""LDS BeginRootFile section="1" filename="installServer.sh"
id="install server" platform="UNIX" executor="sh" -->
<!--""LDS BeginCodeReference id="change to working folder"-->
<!--""LDS EndCodeReference -->
<pre><code>
bundle exec knife solo prepare $.$-username-$.$@$.$-host-$.$
</code></pre>
<!--""LDS EndRootFile -->
```

Declared Variables can be used multiple times in one RootFile. This is showed in the example below.

```
<!--""LDS BeginRootFile section="1" filename="editconfiguration.sh"
id="edit server configuration" platform="UNIX" executor="sh"
cp ~/Code/nodes/sample_host.json ~/Code/nodes/$.$-host-$.$.json
xdg-open ~/Code/nodes/$.$-host-$.$.json
<!--""LDS EndRootFile -->
```

The following command installs the chef repo on a server. The command basis has been used in the previous commands so far.

```
<!--""LDS BeginRootFile section="1" filename="installServerFull.sh"
id="install server Step2" platform="UNIX" executor="sh" -->
<!--""LDS BeginCodeReference id="change to working folder"-->
<!--""LDS EndCodeReference -->
<pre><code>
bundle exec knife solo cook $.$-username-$.$@$.$-host-$.$
</code></pre>
<!--""LDS EndRootFile -->
```

In the next Command a file needs to be changed. For this purpose the BeginInsertIntoFile command is used. To know which file needs to be changed a filename needs to be given. The other arguments needed are a linenumber to insert the lines to and an id.

```
<!--""LDS BeginInsertIntoFile filename="~/Code/Gemfile" line="13"
id="Add capistrano to the Gemfile"--><pre><code>
# your other gems..
gem 'capistrano', '~> 3.2.1'
gem 'capistrano-rails', '~> 1.1'
</code></pre>
<!--""LDS EndInsertIntoFile-->
```

The next RootFile installes the bundle but has no extra attributes.

```
<!--""LDS BeginRootFile section="1" filename="bundleCapistrano.sh"
id="Bundle Capistrano" platform="UNIX" executor="sh" -->
<!--""LDS BeginCodeReference id="change to working folder"-->
<!--""LDS EndCodeReference -->
<pre><code>
bundle
</code></pre>
<!--""LDS EndRootFile -->
```

Additionally the capistrano configuration files need to be generated.

```
<!--""LDS BeginRootFile section="1" filename="configureCapistrano.sh"
id="Configure Capistrano" platform="UNIX" executor="sh" -->
<!--""LDS BeginCodeReference id="change to working folder"-->
```

```
<!--""LDS EndCodeReference -->
<pre><code>
bundle exec cap install
</code></pre>
<!--""LDS EndRootFile -->
```

The ChangeFileLines command works as BeginInsertFile but has a start and an endpoint. This way files can be changed instead of just expanded.

```
<!--""LDS BeginChangeFileLines filename="~/Code/Capfile"
id="Edit Capistrano configuration" startline="0" endline="25" -->
<pre><code>
# Load DSL and Setup Up Stages
require 'capistrano/setup'
# Includes default deployment tasks
require 'capistrano/deploy'
require 'capistrano/rails'
# Loads custom tasks from `lib/capistrano/tasks' if you have any defined.
Dir.glob('lib/capistrano/tasks/*.cap').each { |r| import r }
</code></pre>
<!--""LDS EndChangeFileLines -->
```

A new variable needs to bedeclared before changing the deploy file.

```
<!--""LDS DeclareVariables variables="git_repo_url,application_name" -->

<!--""LDS BeginChangeFileLines filename="~/Code/config/deploy.rb"
id="Edit Capistrano deployment configuration" startline="0" endline="59" -->
<pre><code>ruby
# config valid only for Capistrano 3.2.1
lock '3.2.1'
set :application, '$.$-application_name-$.$'
set :repo_url, '$.$-git_repo_url-$.$'
# Default branch is :master
# Uncomment the following line to have Capistrano ask which branch to deploy.
# ask :branch, proc { `git rev-parse --abbrev-ref HEAD`.chomp }
# Replace the sample value with the name of your application here:
set :deploy_to, '/u/apps/>> your_application_name <<_production'
# Use agent forwarding for SSH so you can deploy with the SSH key on your wo
set :ssh_options, {
forward_agent: true
}
# Default value for :pty is false
set :pty, true
set :linked_files, %w{config/database.yml .rbenv-vars .ruby-version}
set :linked_dirs, %w{log tmp/pids tmp/cache
                tmp/sockets vendor/bundle public/system}
set :default_env, { path: "/opt/rbenv/shims:$PATH" }
set :keep_releases, 5
namespace :deploy do
desc 'Restart application'
task :restart do
on roles(:app), in: :sequence, wait: 5 do
execute :touch, release_path.join('tmp/restart.txt')
end
end
```

```
after :publishing, :restart
end
</code></pre>
<!--""LDS EndChangeFileLines -->
```

A few other files need to be changed for the deployment. So no new commands and definitions are taking place we just leave these commands to be read by whoever wants to.

```
<!--""LDS DeclareVariables variables="server_address" -->

<!--""LDS BeginChangeFileLines filename="~/Code/config/deploy/production.rb"
id="Edit Capistrano server configuration" startline="0" endline="46" -->
<pre><code>ruby
server '$.$-server_address-$.$', user: 'deploy', roles: %w{web app db}
</code></pre>
<!--""LDS EndChangeFileLines -->

<!--""LDS BeginRootFile section="1" filename="checkCapistranoConfiguration.sh
id="Check Capistrano Configuration" platform="UNIX" executor="sh" -->
<!--""LDS BeginCodeReference id="change to working folder"-->
<!--""LDS EndCodeReference -->
<pre><code>sh
bundle exec cap production deploy:check
</code></pre>
<!--""LDS EndRootFile -->

<!--""LDS BeginRootFile section="1" filename="deployCapistrano.sh"
id="Deploy Capistrano" platform="UNIX" executor="sh" -->
<!--""LDS BeginCodeReference id="change to working folder"-->
<!--""LDS EndCodeReference --><pre><code>sh
bundle exec cap production deploy
</code></pre>
<!--""LDS EndRootFile -->

<!--""LDS BeginRootFile section="1" filename="installVagrantPlugins.sh"
id="Install Vagrant plugins" platform="UNIX" executor="sh" -->
<pre><code>
vagrant plugin install vagrant-librarian-chef
vagrant plugin install vagrant-omnibus
</code></pre>
<!--""LDS EndRootFile -->

<!--""LDS BeginRootFile section="1" filename="startVagrant.sh"
id="Start Vagrant" platform="UNIX" executor="xterm -e sh $.$-f-$.$ &" -->
<pre><code>
vagrant up mysql
</code></pre>
<!--""LDS EndRootFile -->

<!--""LDS DeclareVariables variables="Your_Digital_Ocean_Access_Token" -->

<!--""LDS BeginRootFile section="1" filename="getDigitalOceanIDs.sh"
id="Get DigitalOcean IDs" platform="UNIX" executor="sh" -->
<!--""LDS BeginCodeReference id="change to working folder"-->
```

```
<!--""LDS EndCodeReference -->
<pre><code>
curl -X GET https://api.digitalocean.com/v2/account/keys -H "Authorization: B
</code></pre>
<!--""LDS EndRootFile -->

<!--""LDS DeclareVariables variables="Your_Digital_Ocean_SSH_Key_ID" -->

<!--""LDS BeginRootFile section="1" filename="runDigitalOceanTest.sh"
id="Run Digital Ocean Tests" platform="UNIX" executor="sh" -->
<!--""LDS BeginCodeReference id="change to working folder"-->
<!--""LDS EndCodeReference -->
<pre><code>
export DIGITALOCEAN_ACCESS_TOKEN=$.$-Your_Digital_Ocean_Access_Token-$.$
export DIGITALOCEAN_SSH_KEY_IDS=$.$-Your_Digital_Ocean_SSH_Key_ID-$.$
bin/kitchen test
</code></pre>
<!--""LDS EndRootFile -->

<!--""LDS BeginRootFile section="1" filename="runDigitalOceanVerify.sh"
id="Run Digital Ocean Verify" platform="UNIX" executor="sh" -->
<!--""LDS BeginCodeReference id="change to working folder"-->
<!--""LDS EndCodeReference -->
<pre><code>
$ bin/kitchen verify
</code></pre>
<!--""LDS EndRootFile -->
```

4 RESULTS AND DISCUSSION

As a result it can be said that transforming an already given tutorial is relatively easy. The only problem is that there can be constraints or user interactions which cannot be transformed to LDS that easy. In this tutorial MySQL and Ubuntu needed to be installed for MySql a simple installation can be done with few commands. Ubuntu instead is a operating system and can therefore not be installed by an LDS. This is just one example written down in a tutorial which is not transformable. The Program itself can be installed by LDS in an very simple way.