CentOS Artwork Repository

Manual

Alain Reguera Delgado

This manuals documents relevant information regarding the deployment, organization, and administration of CentOS Artwork Repository.

Copyright © 2009-2011 Alain Reguera Delgado

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

Table of Contents

1 Introduction

Welcome to CentOS Artwork Repository Manual.

The CentOS Artwork Repository Manual describes how The CentOS Project Corporate Visual Identity is organized and produced inside the CentOS Artwork Repository (https://projects.centos.org/svn/artwork/). If you are looking for a comprehensive, task-oriented guide for understanding how The CentOS Project Corporate Visual Identity is produced, this is the manual for you.

This manual discusses the following intermedite topics:

- The CentOS Brand
- The CentOS Corporate Visual Structure
- The CentOS Corporate Visual Style

This manual is organized in the same way of CentOS Artwork Repository directory structure. In the CentOS Artwork Repository we use directories to store conceptual ideas of The CentOS Project Corporate Visual Identity. For this manual sake, each directory in CentOS Artwork Repository has its own documentation section to describe the related conceptual ideas it is based on.

This manual uses Texinfo as base documentation system. The Texinfo documentation structure is controlled by the `help` functionality of `centos-art.sh` script. Through this functionality you can create documentation sections for each directory structure inside the repository and edit those already created, as well. See Section 2.33 [Directories trunk Scripts Functions Help], page 52, for more information on how to use the `help` functionality of `centos-art.sh` script.

This guide assumes you have a basic understanding of your CentOS system. If you need help with CentOS, refer to the help page on the CentOS Wiki (http://wiki.centos.org/Help) for a list of different places you can find help.

1.1 History

This section records noteworthy changes of CentOS Artwork Repository through years.

2008

The CentOS Artwork Repository started at CentOS Developers mailing list (centos-devel@centos.org) during a discussion about how to automate the slide images of Anaconda. In such discussion, Ralph Angenendt rose up his hand to ask: Do you have something to show?

To answer the question, Alain Reguera Delgado posted a bash script to produce slide images in different languages —together with the proposition of creating a Subversion centralized repository where translations and image production could be distributed inside The CentOS Community—.

Karanbirn Sighn considered the idea intresting and provides the infrastructure necessary to support the effort. This way the CentOS Artwork SIG and the CentOS Artwork Repository are officially created and made available in the following urls:

- https://projects.centos.org/trac/artwork/
- https://projects.centos.org/svn/artwork/

Once the CentOS Artwork Repository was available, Alain Reguera Delagdo uploaded the bash script for rendering Anaconda slides; Ralph Angenendt documented it very well and The CentOS Translators started to download working copies of CentOS Artwork Repository to produce slide images in their own languages.

2009

The rendition script is at a very rustic state where only slide images can be produced.

The rendition script is improved to produce not only slide images, but PNG images using one SVG file as input. In this configuration one translated SVG instance was created from the SVG provided as input in order to produce one translated PNG image as output. The translation of SVG files is made through SED replacement commands and the rendition of PNG images is realized through Inkscape command line internface.

The rendition script is named `render.sh`. The directory structures are prepared to receive the rendition script so images could be produced inside them. Each directory structure has design templates (.svg), translation files (.sed), and translated images (.png).

The rendition script is unified in a common place and linked from different directory structures. There is no need to have the same code in different directory structures if it can be in just one place and then be linked from different locations.

The concepts of corporate identity started to be considered. As referece, it is used Wikipedia (http://en.wikipedia.org/Corporate_identity) and the book *Corporate Identity* by Wally Olins (1989).

The rendition script main's goal becomes to: automate production of a monolithic corporate visual identity structure based on The CentOS Mission and The CentOS Release Schema.

The documentation of CentOS Artwork Repository starts to take form in LaTeX format.

2010

The rendition script `render.sh` is no longer a rendition script, but a collection of functionalities grouped into the `centos-art.sh` script where rendition is one functionality among others. The `centos-art.sh` is created to automate most frequent tasks inside the repository. There is no need to have links all around the repository if a command-line interface can be created and called anywhere inside the repository as it is usually done with regular commands.

Inside `centos-art.sh`, functionalities started to get identified and separated one another. For example, when images are rendered, there is no need to load functionalities related to documentation manual. There is now common functionalities and specific functionalities. Common functionalities are loaded when the script is initiated and are available to specific functionalities.

The `centos-art.sh` script is updated to handle options trough `getopt` option parser.

The repository directory structure is updated to improve the implementation of corporate visual identity concepts.

2011

The `centos-art.sh` script is updated to translate SVG and other XML-based files (e.g., XHTML and Docbook) through `xml2po` program and shell scripts files through `xgettext` command. In this configuration there is no need to use '.sed' translation files as they previously were used.

The `centos-art.sh` script is updated to improve option parsing through `getopt` program. All arguments are parsed by `getopt` now. Once all option arguments have been parsed, only non-option arguments remain for processing.

The `centos-art.sh` script is updated to organize functionalities in two groups: "the administrative functionalities" and "the productive functionalities". The administrative functionalities cover actions like: copying, deleting and renaming directory structures inside the repository. Also, preparing your workstation for using `centos-art.sh` script, making backups of the distribution theme currently installed, installing themes created inside repository and restoring themes from backup. On the other hand, the productive functionalities cover actions like: content rendition, content localization, content documentation and content maintainance.

1.2 Authors

This section records authoring information of CentOS Artwork Repository:

Karanbirn Singh <karan@centos.org>
Infrastructure, Packaging.

Ralph Angenendt <ralph@centos.org>
Infrastructure, Packaging.

Alain Reguera Delgado
Implementation of a monolithic corporate visual identity for The CentOS Project that can be maintained by The CentOS Community through the CentOS Artwork Repository and the `centos-art.sh` script.

Marcus Moeller <marcus@moeller.org>
Theme Design.

Guideon de Kok
Theme Design.

1.3 Copying Conditions

Inside the CentOS Artwork Repository you can find content branded by The CentOS Project and content not branded at all. Contents branded by The CentOS Project contain either The CentOS Trademark, The CentOS Logo or The CentOS Symbol. Content branded by The CentOS Project cannot be redistributed without previous conversation with The CentOS Project. However, you can study and modify both content branded by The CentOS Project and content not branded at all in the sake of proposing improvements to The CentOS Project corporate visual identity.

If you are using the CentOS Artwork Repository for producing your own corporate visual identity, you should remove all The CentOS Trademarks from your contents and rename the repository to something other than CentOS Artwork Repository.

The CentOS Artwork Repository organizes files in a very specific way to implement The CentOS Project corporate visual identity. This very specific organization of files is part of `centos-art.sh` script, a bash script that automates most of the frequent tasks inside the repository.

The `centos-art.sh` script

The `centos-art.sh` script and the organization of files it needs to work are not in the public domain; they are copyrighted and there are restrictions on their distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of this program that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of `centos-art.sh` script, that you receive source code or else can get it if you want it, that you can change this program or use pieces of it in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of the `centos-art.sh` script, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for the `centos-art.sh` script. If this program is modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the license for the `centos-art.sh` script are found in the General Public Licenses that accompany it (see file trunk/Scripts/COPYING). This manual specifically is covered by the GNU Free Documentation License.

1.4 Document Convenctions

In this manual the personal pronoun *we* is used to repesent *The CentOS Artwork SIG*. This is, the group of persons building the CentOS Artwork Repository.

In this manual, certain words are represented in different fonts, typefaces, sizes, and weights. This highlighting is systematic; different words are represented in the same style to indicate their inclusion in a specific category. The types of words that are represented this way include the following:

`command`

Linux commands (and other operating system commands, when used) are represented this way. This style should indicate to you that you can type the word or phrase on the command line and press Enter to invoke a command. Sometimes a command contains words that would be displayed in a different style on their own (such as file names). In these cases, they are considered to be part of the command, so the entire phrase is displayed as a command. For example:

Use the `centos-art identity --render='path/to/dir'` command to produce contents inside the '`trunk/Identity`' directory structure.

'`file name`'

File names, directory names, paths, and RPM package names are represented this way. This style indicates that a particular file or directory exists with that name on your system. Examples:

The '`init.sh`' file in '`trunk/Scripts/Bash/Cli/`' directory is the initialization script, written in Bash, used to automate most of tasks in the repository.

The `centos-art` command uses the '`ImageMagick`' RPM package to convert images from PNG format to other formats.

⟨key⟩

A key on the keyboard is shown in this style. For example:

To use ⟨TAB⟩ completion to list particular files in a directory, type `ls`, then a character, and finally the Tab key. Your terminal displays the list of files in the working directory that begin with that character.

⟨key-combination⟩

A combination of keystrokes is represented in this way. For example:

The ⟨Ctrl-Alt-Backspace⟩ key combination exits your graphical session and returns you to the graphical login screen or the console.

`computer output`

Text in this style indicates text displayed to a shell prompt such as error messages and responses to commands. For example:

The `ls` command displays the contents of a directory. For example:

```
Config                              manual_renameEntry.sh
manual_copyEntry.sh                 manual_restoreCrossReferences.sh
manual_deleteCrossReferences.sh     manual_searchIndex.sh
```

The output returned in response to the command (in this case, the contents of the directory) is shown in this style.

Additionally, we use several different strategies to draw your attention to certain pieces of information. In order of urgency, these items are marked as a note, tip, important, caution, or warning. For example:

Note Remember that Linux is case sensitive. In other words, a rose is not a ROSE is not a rOsE.

Tip The directory '`/usr/share/doc/`' contains additional documentation for packages installed on your system.

Important If you modify the DHCP configuration file, the changes do not take effect until you restart the DHCP daemon.

Caution Do not perform routine tasks as root — use a regular user account unless you need to use the root account for system administration tasks.

Warning Be careful to remove only the necessary partitions. Removing other partitions could result in data loss or a corrupted system environment.

1.5 Repository Layout

"CentOS like trees, has roots, trunk, branches, leaves and flowers. Day by day they work together in freedom, ruled by the laws of nature and open standards, to show the beauty of its existence."

This section describes the organization of files and directories inside the CentOS Artwork Repository. The repository layout provides the standard backend required for automation scripts to work correctly. If such layout changes unexpectedly, automation scripts may confuse themselves and stop doing what you may expect from them to do.

As convenction, inside CentOS Artwork Repository, files and directories related to CentOS corporate visual identity are organized under three top level directories named: '`trunk/`', '`branches/`', and '`tags/`'.

The '`trunk/`' directory (see Section 2.3 [Directories trunk], page 12) organizes the main development line of CentOS corporate visual identity. Inside '`trunk/`' directory structure, the CentOS corporate visual identity concepts are implemented using directories. There is one directory level for each relevant concept inside the repository. The '`trunk/`' directory structure is mainly used to perform development tasks related to CentOS corporate visual identity.

The '`branches/`' directory (see Section 2.1 [Directories branches], page 11) oranizes parallel development lines to '`trunk/`' directory. The '`branches/`' directory is used to set points in time where develpment lines are devided one from another taking separte and idependent lives that share a common past from the point they were devided on. The '`branches/`' directory is mainly used to perform quality assurance tasks related to CentOS corporate visual identity.

The '`tags/`' directory (see Section 2.2 [Directories tags], page 11) organizes parallel frozen lines to '`branches/`' directory. The parallel frozen lines are immutable, nothing change inside them once they has been created. The '`tags/`' directory is mainly used to publish final releases of CentOS corporate visual identity.

The CentOS Artwork Repository layout is firmly grounded on a Subversion base. Subversion (`http://subversion.tigris.org`) is a version control system, which allows you to keep old versions of files and directories (usually source code), keep a log of who, when, and why changes occurred, etc., like CVS, RCS or SCCS. Subversion keeps a single copy of the master sources. This copy is called the source "repository"; it contains all the information to permit extracting previous versions of those files at any time.

1.5.1 Repository file names

Repository name convenctions are applied to files and directories inside the repository layout. As convenction, file names are all written in lowercase ('`01-welcome.png`', '`splash.png`',

'anaconda_header.png', etc.) and directory names are all written capitalized (e.g., 'Identity', 'Themes', 'Motifs', 'TreeFlower', etc.).

Repository name convenctions are implemented inside the cli_getRepoName function of 'centos-art.sh' script. With cli_getRepoName function the amount of commands and convenctions to remember are reduced and concentrated in just one single place to look for fixes and improvements.

1.5.2 Repository work flow

As repository work flow we understand a serie of normalized steps used to produce contents inside CentOS Artwork Repository. There are three remarkable actions inside the repository that we need to provide a normalized work flow for, they are: *Rendition*, *Localization* and *Programming*.

1.5.2.1 Rendition

See Section 2.4 [Directories trunk Identity], page 12.

Rendition is the action through which The CentOS Project Corporate Identity is produced inside the repository. The CentOS Project Corporate Identityis is made, in part, of several visual manifestations. These visual manifestations implement the Corporate Identity by mean of images placed in key areas of each manifestation visual space.

As work flow to produce these images, we decompose them in design models and artistic motifs. Design models provide the image structure (i.e., dimension, translation markers, common designs, etc.) and artistic motifs the visual style (i.e., the background information provide the look and feel).

Design models are created for each visual manifestation the CentOS Project is made of. This way we describe the visual manifestation and provide a template system for it where several artistic motifs can be applied. Artistic motifs are created with design models in mind, not the visual manifestation such design models is built for.

The combination of one design models with one artistic motifs is what we know as one *theme*. Inside themes directory structure, you can find several design models and several artistic motifs, apart one another, that can be albitrarily combined one another through *theme rendition*, a flexible way to produce images for different visual manifestations inside CentOS Artwork Repository. Theme rendition takes place in 'trunk/Identity/Themes/Models' and 'trunk/Identity/Themes/Motifs' directory structures.

In addition to theme rendition, you can find another way of image production, a more direct way of image production where there is no artistic motif at all, but design models only. Such configuration is named *direct rendition* and is very useful to produce simple content that doesn't need specific background information like icons and illustrations used in documentation. Direct rendition takes place in 'trunk/Identity/Models' and 'trunk/Identity/Images' directory structures.

1.5.2.2 Localization

See ⟨undefined⟩ [Directories trunk Locales], page ⟨undefined⟩.

Whatever content rendition you perform, sometimes you need to produce content in different languages. Production of content in different languages is known as localization or l10n for short. Inside CentOS Artwork Repository, content localization is performed using the processed provided by gettext internationalization standard.

Basically, the gettext internationalization standard consists on retriving translatable strings from source files and creating Portable Objects and Machine Objects. Portable Objects contain the information used by translators to do their work, they are files that can be edited by any

convenctional text editor like Vim, Emacs or Nano. On the other hand, Machine Objects are produced to be machine-redable as its name implies, and are produced from Portable Objects.

Since `gettext` needs to extract translatable strings form source files in order to let translators to localize them, the types of source files we use inside the repository are limitted to the file types supported by `gettext` program. Most of source files supported by `gettext` are those from programming languages like C, C++, Bash, Python and Perl just to mention a few ones from the long list of supported formats. However, formats like SVG, XHTML and Docbook don't figure as supported in the long list of `gettext` supported source files. For these formats we use the `xml2po` program that come in the '`gnome-doc-utils`' package instead. The `xml2po` program reads one XML based file and extracts translatable strings from it and creates one Portable Object that is use to create a translated XML based file with the same definition of the source file where translatable strings were taken from (e.g., if we extract translatable strings from a SVG file, as result we get the same SVG file but with translatable strings already localized — obviously, for this to happen translators need to localize translatable strings inside the Portable Object first, localization won't appear as art of magic—). When using `xml2po`, the Machine Object file is used just temporally to produce the final translated XML based file.

> **Tip** If you want to have your content localized inside CentOS Artwork Repository be sure to use source files supported by `gettext` and `xml2po` programs.

1.5.2.3 Programming

See Section 2.31 [Directories trunk Scripts], page 34.

So far, we've seen a conceptual view of how image production inside CentOS Artwork Repository is performed, but how all this is implemented, what is the practical view?

The practical view can be found through `centos-art.sh` script, a bash script specially designed to automate most frequent tasks inside CentOS Artwork Repository. The `centos-art.sh` script is stored in '`trunk/Scripts`' directory structure and is there where the main development for `centos-art.sh` script takes place. Basically, the `centos-art.sh` script is divided in several functionalities that perform specific tasks inside the repository. Such functionalities relay on repository organization to work as expected.

Maiking the `centos-art.sh` script to automate tasks is the only possible work flow that I can think about right now. If you are a programmer, take a functionality from `centos-art.sh` script, study it, look how to improve it and share your ideas at `centos-devel@centos.org` mailing list.

> **Note** As default configuration, everyone is denied from committing changes up to CentOS Artwork Repository. In order to gain commit rights, you need to prove your interest in contributing and maintaining that contribution. Otherwise, if you only want to comment your ideas, the `centos-devel@centos.org` mailing list exists for you to manifest yourself. This restrictions are necessary to protect our work from spammers.

1.5.3 Parallel directories

Inside CentOS Artwork Repository, parallel directories are simple directory entries built from a common parent directory and placed in a location different to that, the common parent directory is placed on. Parallel directories are useful to create branches, tags, translations, documentation, pre-rendering configuration script, and similar directory structures.

Parallel directories take their structure from one unique parent directory. Inside CentOS Artwork Repository, this unique parent directory is under '`trunk/Identity`' location. The '`trunk/Identity`' location must be considered the reference for whatever information you plan to create inside the repository.

In some circumstances, parallel directories may be created removing uncommon information from their paths. Uncommon path information refers to those directory levels in the path which are not common for other parallel directories. For example, when rendering '`trunk/Identity/Themes/Motifs/TreeFlower/Distro`' directory structure, the '`centos-art.sh`' script removes the '`Motifs/TreeFlower/`' directory levels from path, in order to build the parallel directory used to retrived translations, and pre-rendering configuration scripts required by `render` functionality.

Another example of parallel directory is the documentation structure created by `manual` functionality. This time, '`centos-art.sh`' script uses parallel directory information with uncommon directory levels to build the documentation entry required by Texinfo documentation system, inside the repository.

Othertimes, parallel directories may add uncommon information to their paths. This is the case we use to create branches and tags. When we create branches and tags, a numerical identifier is added to parallel directory structure path. The place where the numerical identifier is set on is relevant to corporate visual identity structure and should be carefully considered where it will be.

When one parent directory changes, all their related parallel directories need to be changed too. This is required in order for parallel directories to retain their relation with the parent directory structure. In the other hand, parallel directories should never be modified under no reason but to satisfy the relation to their parent directory structure. Liberal change of parallel directories may suppresses the conceptual idea they were initially created for; and certainly, things may stop working the way they should do.

1.5.4 Syncronizing path information

Parallel directories are very useful to keep repository organized but introduce some complications. For instance, consider what would happen to functionalities like `manual` ('`trunk Scripts Bash Functions Manual`') that rely on parent directory structures to create documentation entries (using parallel directory structures) if one of those parent directory structures suddenly changes after the documentation entry has been already created for it?

In such cases, functionalities like `manual` may confuse themselves if path information is not updated to reflect the relation with its parent directory. Such functionalities work with parent directory structure as reference; if a parent directory changes, the functionalities dont't even note it because they work with the last parent directory structure available in the repository, no matter what it is.

In the specific case of documentation (the `manual` functionality), the problem mentioned above provokes that older parent directories, already documented, remain inside documentation directory structures as long as you get your hands into the documentation directory structure ('`trunk/Manuals`') and change what must be changed to match the new parent directory structure.

There is no immediate way for `manual`, and similar functionalities that use parent directories as reference, to know when and how directory movements take place inside the repository. Such information is available only when the file movement itself takes place inside the repository. So, is there, at the moment of moving files, when we need to syncronize parallel directories with their unique parent directory structure.

> **Warning** There is not support for URL reference inside '`centos-art.sh`' script. The '`centos-art.sh`' script is designed to work with local files inside the working copy only.

As CentOS Artwork Repository is built over a version control system, file movements inside the repository are considered repository changes. In order for these repository changes to be versioned, we need to, firstly, add changes into the version control system, commit them, and

later, perform movement actions using version control system commands. This configuration makes possible for everyone to know about changes details inside the repository; and if needed, revert or update them back to a previous revision.

Finally, once all path information has been corrected, it is time to take care of information inside the files. For instance, considere what would happen if you make a reference to a documentation node, and later the documentation node you refere to is deleted. That would make Texinfo to produce error messages at export time. So, the '`centos-art.sh`' script needs to know when such changes happen, in a way they could be noted and handled without producing errors.

1.5.5 What is the right place to store it?

Occasionly, you may find that new corporate visual identity components need to be added to the repository. If that is your case, the first question you need to ask yourself, before start to create directories blindly all over, is: What is the right place to store it?

The CentOS Community different free support vains (see: http://wiki.centos.org/GettingHelp) are the best place to find answers to your question, but going there with hands empty is not good idea. It may give the impression you don't really care about. Instead, consider the following suggestions to find your own comprehension and so, make your propositions based on it.

When we are looking for the correct place to store new files, to bear in mind the corporate visual identity structure used inside the CentOS Artwork Repository (see Section 2.4 [Directories trunk Identity], page 12) would be probaly the best advice we could offer, the rest is just matter of choosing appropriate names. To illustrate this desition process let's consider the '`trunk/Identity/Themes/Motifs/TreeFlower`' directory as example. It is the trunk development line of *TreeFlower* artistic motif. Artistic motifs are considered part of themes, which in turn are considered part of CentOS corporate visual identity.

When building parent directory structures, you may find that reaching an acceptable location may take some time, and as it uses to happen most of time; once you've find it, that may be not a definite solution. There are many concepts that you need to play with, in order to find a result that match the conceptual idea you try to implement in the new directory location. To know which these concepts are, split the location in words and read its documentation entry from less specific to more specific.

For example, the '`trunk/Identity/Themes/Motifs/TreeFlower`' location evolved through several months of contant work and there is no certain it won't change in the future, even it fixes quite well the concept we are trying to implement. The concepts used in '`trunk/Identity/Themes/Distro/Motifs/TreeFlower`' location are described in the following commands, respectively:

```
centos-art manual --read=turnk/
centos-art manual --read=turnk/Identity/
centos-art manual --read=turnk/Identity/Themes/
centos-art manual --read=turnk/Identity/Themes/Motifs/
centos-art manual --read=turnk/Identity/Themes/Motifs/TreeFlower/
```

Other location concepts can be found similary as we did above, just change the location we used above by the one you are trying to know concepts for.

1.6 Send in Your Feedback

If you find an error in the *CentOS Artwork Repository Manual*, or if you have thought of a way to make this manual better, we would like to hear from you! Create a new ticket in The CentOS Artwork SIG web site (https://projects.centos.org/trac/artwork/).

If you have a suggestion for improving the documentation, try to be as specific as possible. If you have found an error, include the section number and some of the surrounding text so we can find it easily.

2 The Repository Directories

The CentOS Artwork Repository uses directories to organize files and describe conceptual idea about corporate identity. Such conceptual ideas are explained in each directory related documentation entry.

In this chapter you'll learn what each directory inside The CentOS Artwork Repository is for and so, how you can make use of them. For that purpose, the following list of directories is available for you to explore:

2.1 The 'branches' Directory

2.1.1 Goals

This directory implements the Subversion's branches concept in a trunk, branches, tags repository structure.

2.1.2 Description

The '`branches/`' directory structre provides the intermediate space for creating several instances of '`trunk/`' directory structure for parallel development and later merging changes back to '`trunk/`' in the same parallel basis.

2.1.3 Usage

The '`branches/`' directory structure is unused, so far.

2.1.4 See also

- See Section 2.2 [Directories tags], page 11.
- See Section 2.3 [Directories trunk], page 12.
- Subversion's book (`http://svnbook.red-bean.com/`).

2.2 The 'tags' Directory

2.2.1 Goals

This directory implements the Subversion's tags concept in a trunk, branches, tags repository structure.

2.2.2 Description

The '`tags/`' directory structre provides frozen branches. Generally, we use frozen branches to make check-points in time for development lines under '`branches/`' or '`trunk/`' directory structure.

2.2.3 Usage

The '`tags/`' directory structure is unused, so far.

2.2.4 See also

- See Section 2.1 [Directories branches], page 11.
- See Section 2.3 [Directories trunk], page 12.
- Subversion's book (`http://svnbook.red-bean.com/`).

2.3 The 'trunk' Directory

2.3.1 Goals

This directory implements the Subversion's trunk concept in a trunk, branches, tags repository structure.

2.3.2 Description

The '`trunk/`' directory structure is the main development line inside the CentOS Artwork Repository and organizes the following sections:

Identity

This section describes the implementation of The CentOS Project Corporate Identity. This place is perfect to consolidate *The CentOS Artwork SIG*. If you are interested in producing art works for The CentOS Project, this place is for you.

See Section 2.4 [Directories trunk Identity], page 12, for more information.

Scripts

This section describes the implementation of `centos-art.sh` script, a bash scripts specially designed to automate most frequent tasks in the repository (e.g., image rendition, documenting directory structures, translating content, etc.). If you can't resist the idea of automating repeatable tasks, then take a look here.

See Section 2.31 [Directories trunk Scripts], page 34, for more information.

2.3.3 Usage

It seems to be no other use for this directory but to organize the sections described above.

2.3.4 See also

- See Section 2.1 [Directories branches], page 11.
- See Section 2.2 [Directories tags], page 11.
- Subversion's book (`http://svnbook.red-bean.com/`).

2.4 The 'trunk/Identity' Directory

2.4.1 Goals

This section descirbes the implementation of *The CentOS Project Corporate Identity*.

2.4.2 Description

The CentOS Project Corporate Identity is the "persona" of the organization known as The CentOS Project. The CentOS Project Corporate Identity plays a significant role in the way the CentOS Project, as organization, presents itself to both internal and external stakeholders. In general terms, the CentOS Project Corporate Identity expresses the values and ambitions of the CentOS Project organization, its business, and its characteristics.

The CentOS Project Corporate Identity provides visibility, recognizability, reputation, structure and identification to The CentOS Project organization by means of *Corporate Design*, *Corporate Communication*, and *Corporate Behaviour*.

2.4.2.1 Corporate Design

The CentOS Project Corporate Design is applied to every single visual manifestations The CentOS Project as organization wants to express its existence. Examples of the most relevant

visual manifestations inside The CentOS Project are *The CentOS Distribution*, *The CentOS Web* and *The CentOS Stationery*.

The CentOS Project Corporate Design is organized in the following work-lines:

Brands

The CentOS Brand provides the one unique name or trademark that connects the producer with their products. In this case, the producer is The CentOS Project and the products are The CentOS Project visual manifestations.

See Section 2.5 [Directories trunk Identity Brands], page 16, for more information.

Palettes

The CentOS Palettes provide the *Corporate Color* information used along The CentOS Project visual manifestations.

See Section 2.9 [Directories trunk Identity Palettes], page 18, for more information.

Fonts

The CentOS Fonts provide the *Corporate Typography* information used along The CentOS Project visual manifestations.

See Section 2.6 [Directories trunk Identity Fonts], page 16, for more information.

Themes

The CentOS Themes provide the *Corporate Structure* and the *Corporate Visual Style* used along The CentOS Project visual manifestations.

See Section 2.10 [Directories trunk Identity Themes], page 18, for more information.

Manual

This section organizes the *CentOS Artwork Repository Manual* (i.e., the documentation manual you're reading right now). If you are interested on improving The CentOS Artwork Repository Manual, in this place you'll find the Texinfo documentation structure you need to work with.

See Section 2.8 [Directories trunk Identity Manual], page 18, for more information.

Locales

This section organizes production of translation messages for *Identity*, *Documentation* and *Scripts*. This place is perfect to consolidate *The CentOS Translation SIG*. If you love translating, you'll find lot of messages waiting for you to translate here.

See Section 2.7 [Directories trunk Identity Locales], page 17, for more information.

2.4.2.2 Corporate Communication

The CentOS Project Corporate Communication is based on *Community Communication*. In that sake, the following media are available:

- The CentOS Chat (`#centos`, `#centos-social`, `#centos-devel` on irc.freenode.net)
- The CentOS Mailing Lists (`http://lists.centos.org/`).
- The CentOS Forums (`http://forums.centos.org/`).

2.4.2.3 Corporate Behaviour

The CentOS Project Corporate Behaviour is based on *Community Behaviour*.

2.4.2.4 Corporate Structure

The CentOS Project Corporate Structure is based on a *Monolithic Corporate Visual Identity Structure*. In this structure, one unique name and one unique visual style is used in all visual manifestation of The CentOS Project.

In a monolithic corporate visual identity structure, internal and external stakeholders use to feel a strong sensation of uniformity, orientation, and identification with the organization. No matter if you are visiting web sites, using the distribution, or acting on social events, the one unique name and one unique visual style connects them all to say: *Hey! we are all part of The CentOS Project*.

Other corporate structures for The CentOS Project have been considered as well. Such is the case of producing one different visual style for each major releasae of CentOS Distribution. This structure isn't inconvenient at all, but some visual contradictions could be introduced if it isn't applied correctly and we need to be aware of it. To apply it correctly, we need to know what The CentOS Project and which are the visual manifestations it is made of.

The CentOS Project, as organization, is mainly made of (but not limited to) three visual manifestions: Distribution, Web and Stationery. Inside the Distribution visual manifestations, The CentOS Project maintains near to four different major releases of CentOS Distribution, parallely in time. Inside Web and Stationery visual manifestations content is visually produced to fit non-release-specifc content but treat it as a visual manifestation properly. For example, consider that there is no a complete web site for each major release of CentOS distribution, but one web site to cover the information related to all release-specific visual manifestations like CentOS distribution.

In order to produce the correct corporate structure for The CentOS Project we need to concider all the visual manifestations The CentOS Project is made of, not just one of them. If one different visual style is used for each major release of The CentOS Distribution, which one of those different visual styles would be used to cover the remaining visual manifestations The CentOS Project is made of (e.g., web sites and stationery)?

Probably you are thinking, that's right, but The CentOS Brand connects them all already, why would we need to join them up into the same visual style too, isn't it more work to do, and harder to maintain?

Harder to maintain, more work to do, probably. Specially when you consider that The CentOS Project has proven stability and consistency through time and that, certainly, didn't come through swinging magical wangs or something but hardly working out to automate tasks and providing maintainance through time. Said that, we consider that The CentOS Project Visual Structure should be consequent with such stability and consistency tradition. It is true that The CentOS Brand does connect all the visual manifestations it is present on, but that connection would be stronger if one unique visual style backups it. In fact, whatever thing you do to strength the visual connection among The CentOS Project visual manifestations would be very good in favor of The CentOS Project recognition.

Obviously, having just one visual style in all visual manifestations for eternity would be a very boring thing and would give the idea of a visually dead project. So, there is no problem on creating a brand new visual style for each new major release of The CentOS Distribution, in order to refresh The CentOS Distribution visual style; the problem is in not propagating the brand new visual style created for the new release of CentOS Distribution to all other visual manifestations The CentOS Project is made of, in a way The CentOS Project could be recognized no matter what visual manifestation be in front of us. Such lack of uniformity is what introduces the visual contradition we are precisely trying to solve by mean of themes production in the CentOS Artwork Repository.

2.4.3 Usage

The '`trunk/Identity/`' directory structure is organized in *renderable* and *non-renderable* directories. Generally, renderable directories are stored under '`trunk/Identity/Images`' and '`trunk/Identity/Themes/Motifs`' directories. These directories contain the image files used to implemente The CentOS Project Corporate Identity.

2.4.3.1 Rendition

In order to produce content inside rendereble directories, you can use the following command:

```
centos-art render trunk/Identity/Path/To/Dir
```

> **Warning** If the `centos-art` command-line is not found in your workstation, it is probably because you haven't prepared your workstation for using The CentOS Artwork Repository yet. See Section 2.36 [Directories trunk Scripts Functions Prepare], page 58, for more information.

This command takes one design template (a.k.a., design model) from the template directory and creates an instance of it in order to apply translation messages, if any. Later, using the translated design template instance, the command renders the final content based on whether the design template instance is a SVG file or XHTML. If the design template instace is a SVG file, the final content produced is a PNG image. On the other hand, if the design template instance is a XHTML file, the final content produced is a XHTML file. The rendition flow described so far is known as the `centos-art.sh` script *base-rendition* flow.

Besides the base-rendition flow, the `centos-art` provides *post-rendition* and *last-rendition* flows. The post-rendition flow is applied to files produced as result of base-rendition flow under the same directory structure. For example, you can use post-rendition action to convert the PNG base output into different outputs formats (e.g., JPG, PDF, etc.) before passing to process the next file in the same directory structure. The last-rendition flow, on the other hand, is applied to all files produced as result of both base-rendition and post-rendition flows in the same directory structure, just before passing to process a different directory structure. For example, the '`Preview.png`' image from Ksplash component is made of three images. In order to build the '`Preview.png`' image through `centos-art.sh` we need to wait for all the three images the '`Preview.png`' image is made of to be rendered in order to combine them all together into just one image (i.e., the '`Preview.png`' image). This is something we can't do using post-rendition flow.

Inside '`trunk/Identity`' directory structure, you can find that base-rendition, post-rendition and last-rendition flows can be combined to build *directory-specific* rendition. The directory-specific rendition exists to automatically process specific renderable directories in very specific ways. Using directory-specific rendition speeds up production of different components like Syslinux, Grub, Gdm, Kdm and Ksplash that require intermediate formats or even several independent files, in order to reach the final content construction. Directory-specific rendition is a way to programmatically describe how specific art works are built in and organized inside The CentOS Artwork Repository. Such descriptions have been added to `centos-art.sh` command-line to let you produce them all with just one single command, as fast as your machine can be able to handle it.

See Section 2.37 [Directories trunk Scripts Functions Render], page 63, for more information about the `render` functionality of `centos-art.sh` script.

2.4.3.2 Documentation

2.4.3.3 Localization

2.4.4 See also

See http://en.wikipedia.org/Corporate_identity (and related links), for general information on Corporate Identity.

Specially useful has been, and still is, the book *Corporate Identity* by Wally Olins (1989). This book provides many conceptual ideas we've used as base to build The CentOS Artwork Repository.

2.5 The 'trunk/Identity/Brands' Directory

2.5.1 Goals

- ...

2.5.2 Description

2.5.3 Usage

2.5.4 See also

2.6 The 'trunk/Identity/Fonts' Directory

2.6.1 Goals

This section exists to organize digital typographies used by the CentOS project.

2.6.2 Description

2.6.3 Usage

The CentOS corporate identity is attached to '`DejaVu LGC`' font-family. Whatever artwork you design for CentOS project, that requires typography usage, must be done using '`DejaVu LGC`' font-family.

Recommendation-1:

For screen desings (e.g., anything that final destination will never be printed on paper or any medium outside computer screens) use '`DejaVu LGC Sans`' font-family.

Recommendation-2:

For non-screen designs (e.g., anything that final desition will be printed on paper or any other medium outside computer screens) use '`DejaVu LGC Serif`' font-family. As convenction files described in this rule are stored under '`Stationery`' directories.

The only execption for the two recommendations above is the typography used inside CentOS logo. The CentOS logo is the main visual representation of the CentOS project so the typography used in it must be the same always, no matter where it be shown. It also has to be clear enough to dismiss any confussion between similar typefaces (e.g., the number one (1) sometimes is confuesed with the letter '`el`' (l) or letter '`ai`' (i)).

As CentOS logo typography convenction, the word '`CentOS`' uses '`Denmark`' typography as base, both for the word '`CentOS`' and the phrase '`Community Enterprise Operating System`'. The phrase size of CentOS logo is half the size in poits the word '`CentOS`' has and it below '`CentOS`' word and aligned with it on the left. The distance between '`CentOS`' word and phrase '`Community Enterprise Operating System`' have the size in points the phrase has.

When the CentOS release brand is built, use '`Denmark`' typography for the release number. The release number size is two times larger (in height) than default '`CentOS`' word. The separation between release number and '`CentOS`' word is twice the size in points of separation between '`CentOS`' word and phrase '`Community Enterprise Operating System`'.

Another component inside CentOS logo is the trademark symbol (TM). This symbol specifies that the CentOS logo must be consider a product brand, even it is not a registered one. The trademark symbol uses DejaVu LGC Sans Regular typography. The trademark symbol is aligned right-top on the outter side of '`CentOS`' word. The trademark symbol must not exceed haf the distance, in points, between '`CentOS`' word and the release number on its right.

It would be very convenient for the CentOS Project and its community to to make a registered trademark () of CentOS logo. To make a register trademark of CentOS Logo prevents legal complications in the market place of brands. It grants the consistency, through time, of CentOS project corporate visual identity.

> **Note** The information about trademarks and corporate identity is my personal interpretation of http://en.wikipedia.org/Corporate_identity and http://en.wikipedia.org/Trademark description. If you have practical experiences with these affairs, please serve yourself to improve this section with your reasons.

2.6.4 See also

2.7 The '`trunk/Identity/Locales`' Directory

The '`trunk/Locales`' directory exists to store the translation messages used to produce content in different languages.

Translation messages are organized using the directory structure of the component being translated. For example, if we want to provide translation messages for '`trunk/Manuals/Repository`', then the '`trunk/Locales/Manuals/Repository`' directory needs to be created.

Once the locale directory exists for the component we want to provide translation messages for, it is necessary to create the translation files where translation messages are. The translation files follows the concepts of `xml2po` and GNU `gettext` tools.

The basic translation process is as follow: first, translatable strings are extracted from files and a portable object template (.pot) is created or updated with the information. Using the portable object template, a portable object (.po) is created or updated for translator to locale the messages retrived. Finally, a machine object (.mo) is created from portable object to sotore the translated messages.

Inside the repository there are two ways to retrive translatable strings from files. The first one is through `xml2po` command and the second through `xgettext` command. The `xml2po` is used to retrive translatable strings from XML files (e.g., Scalable Vector Graphics, DocBook, etc.) and the `xgettext` command is used to retrive translatable strings from shell scripts files (e.g., the files that make the `centos-art.sh` command-line interface).

When translatable strings are retrived from XML files, using the `xml2po` command, there is no need to create the machine object as we do when translatable strings ar retrived from shell files, using the `xgettext` command. The `xml2po` produces a temporal machine object in order to create a translated XML file. Once the translated XML file has been created the machine object is no longer needed. On the other hand, the machine object produced by the `xgettext` command is required by the system in order for the show shell script localized messages.

Another difference between `xml2po` and `xgettext` we need to be aware of is the directory structure used to store machine objects. In `xml2po`, the machine object is created in the current working directory as '`.xml2po.mo`' and can be safetly removed once the translated XML file has been created. In the case of `xgettext`, the machine object needs to be stored in the '`$TEXTDOMAIN/$LOCALE/LL_MESSAGES/$TEXTDOMAIN.mo`' file in order for the system to interpret it and should not be removed since it is the file that contain the translation messages themselves.

Automation of localization tasks is achived through the `locale` functionality of command-line interface.

2.8 The '`trunk/Identity/Manual`' Directory

2.8.1 Goals

- ...

2.8.2 Description

- ...

2.8.3 Usage

- ...

2.8.4 See also

2.9 The '`trunk/Identity/Palettes`' Directory

2.9.1 Goals

- ...

2.9.2 Description

- ...

2.9.3 Usage

- ...

2.9.4 See also

2.10 The '`trunk/Identity/Themes`' Directory

2.10.1 Goals

The '`trunk/Identity/Themes/`' directory exists to organize production of CentOS themes.

2.10.2 Description

2.10.2.1 Work Flow

Initially, we start working themes on their trunk development line (e.g., '`trunk/Identity/Themes/Motifs/TreeFlower/`'), here we organize information that cannot be produced automatically (i.e., background images, concepts, color information, screenshots, etc.).

Later, when theme trunk development line is considered "ready" for implementation (e.g., all required backgrounds have been designed), we create a branch for it (e.g., '`branches/Identity/Themes/Motifs/TreeFlower/1/`'). Once the branch has been created, we forget that branch and continue working the trunk development line while others (e.g., an artwork quality assurance team) test the new branch for tunning it up.

Once the branch has been tunned up, and considered "ready" for release, it is freezed under '`tags/`' directory (e.g., '`tags/Identity/Themes/Motifs/TreeFower/1.0/`') for packagers, webmasters, promoters, and anyone who needs images from that CentOS theme the tag was created for.

Both branches and tags, inside CentOS Artwork Repository, use numerical values to identify themselves under the same location. Branches start at one (i.e., '1') and increment one unit for each branch created from the same trunk development line. Tags start at zero (i.e., '0') and increment one unit for each tag created from the same branch development line.

> **Convenction** Do not freeze trunk development lines using tags directly. If you think you need to freeze a trunk development line, create a branch for it and then freeze that branch instead.

The trunk development line may introduce problems we cannot see immediatly. Certainly, the high changable nature of trunk development line complicates finding and fixing such problems. On the other hand, the branched development lines provide a more predictable area where only fixes/corrections to current content are commited up to repository.

If others find and fix bugs inside the branched development line, we could merge such changes/experiences back to trunk development line (not visversa) in order for future branches, created from trunk, to benefit.

Time intervals used to create branches and tags may vary, just as different needs may arrive. For example, consider the release schema of CentOS distribution: one major release every 2 years, security updates every 6 months, support for 7 years long. Each time a CentOS distribution is released, specially if it is a major release, there is a theme need in order to cover CentOS distribution artwork requirements. At this point, is where CentOS Artwork Repository comes up to scene.

Before releasing a new major release of CentOS distribution we create a branch for one of several theme development lines available inside the CentOS Artwork Repository, perform quality assurance on it, and later, freeze that branch using tags. Once a the theme branch has been frozen (under '`tags/`' directory), CentOS Packagers (the persons whom build CentOS distribution) can use that frozen branch as source location to fulfill CentOS distribution artwork needs. The same applies to CentOS Webmasters (the persons whom build CentOS websites), and any other visual manifestation required by the project.

2.10.3 Usage

In this location themes are organized in "Models" —to store common information— and "Motifs"—to store unique information. At rendering time, both motifs and models are combined to produce the final CentOS themes. CentOS themes can be tagged as "Default" or "Alternative". CentOS themes are maintained by CentOS community.

2.10.4 See also

2.11 The '`trunk/Identity/Themes/Models`' Directory

2.11.1 Goals

- Organize theme models.

2.11.2 Description

Theme models let you modeling characteristics (e.g., dimensions, translation markers, position of each element on the display area, etc.) common to all themes. Theme models let you reduce the time needed when propagating artistic motifs to different visual manifestations.

Theme models serves as a central pool of design templates for themes to use. This way you can produce themes with different artistic motifs but same characteristics.

2.11.3 Usage

Default Design Model
: Default Design Models for CentOS Themes provide the common structural information (e.g., image dimensions, translation markers, trademark position, etc.) the `centos-art` script uses to produce images when no other design model is specified.

Alternative Design Models
: CentOS alternative theme models exist for people how want to use a different visual style on their installations of CentOS distribution. As the visual style is needed for a system already installed components like Anaconda are not required inside alternative themes. Inside alternative themes you find post-installation visual style only (i.e. Backgrounds, Display Managers, Grub, etc.). CentOS alternative themes are maintained by CentOS Community.

2.11.4 See also

2.12 The 'trunk/Identity/Themes/Models/Default' Directory

2.12.1 Goals

Default Design Models for CentOS Themes provide design models for the following components:

Distribution
: Design models for CentOS Distribution (e.g., Anaconda, Firstboot, Gdm, Grub, Gsplash, Kdm, Ksplash, Rhgb and Syslinux, etc.). See Section 2.14 [Directories trunk Identity Themes Models Default Distro], page 21, for more information.

Concept Design models to illustrate Artistic Motifs Concepts. See Section 2.13 [Directories trunk Identity Themes Models Default Concept], page 21, for more information.

Promotion
: Design models for CentOS Promotion stuff (e.g., installation media, posters, etc.). — **Removed**(xref:Directories trunk Identity Themes Models Default Promo) —, for more information.

2.12.2 Description

This directory implements the concept of *Default Design Models for CentOS Themes*. Default Design Models for CentOS Themes provide the common structural information (e.g., image dimensions, translation markers, trademark position, etc.) the `centos-art` script uses to produce images when no other design model is specified.

Deisgn models in this directory do use the *CentOS Release Brand*. The CentOS Release Brand is a combination of both The CentOS Type and The CentOS Release Schema used to illustrate the major release of CentOS Distribution the image produced belongs to. — **Removed**(xref:Directories trunk Identity Models Tpl Brands) —, for more information.

2.12.3 Usage

The CentOS Project maintains near to four different major releases of CentOS Distribution. Each major release of CentOS Distribution has internal differences that make them unique and, at the same time, each CentOS Distribution individually is tagged into the one unique visual manifestation (i.e., Distribution). So, how could we implement the monolithic visual structure in one visual manifestation that has internal difference?

To answer this question we broke the question in two parts and later combined the resultant answers to build a possible solution.

How to remark the internal differences visually?
Merge both The CentOS Project Release Schema into The CentOS Project Trademark to build The CentOS Project Release Trademark. The CentOS Project Release Trademark remarks two things: first, it remarks the image is from The CentOS Project and second, it remarks which major release of CentOS Distribution does the image belongs to. — **Removed**(xref:Directories trunk Identity Models Tpl Brands) —, for more information on how to develop and improve The CentOS Project Brand.

How to remark the visual resemblance?
Use a common artistic motifs as background for all CentOS Distribution images. See Section 2.25 [Directories trunk Identity Themes Motifs], page 25, for more information.

So, combining answers above, we could conclude that:
In order to implement the CentOS Monolithic Visual Structure on CentOS Distribution visual manifestations, a CentOS Release Trademark and a background information based on one unique artistic motif should be used in all remarkable images The CentOS Distribution visual manifestation is made of.

Important Remarking the CentOS Release Schema inside each major release of CentOS Distribution —or similar visual manifestations— takes *high attention* inside The CentOS Project corporate visual identity. It should be very clear for everyone which major release of CentOS Distribution is being used.

2.12.4 See also

- Section 2.10 [Directories trunk Identity Themes], page 18
- Section 2.11 [Directories trunk Identity Themes Models], page 19
- Section 2.25 [Directories trunk Identity Themes Motifs], page 25

2.13 The ‘trunk/Identity/Themes/Models/Default/Concept’ Directory

2.13.1 Goals

- ...

2.13.2 Description

- ...

2.13.3 Usage

- ...

2.13.4 See also

2.14 The ‘trunk/Identity/Themes/Models/Default/Distro’ Directory

2.14.1 Goals

This directory provides design models to produce image files for the following CentOS Distribution components:

Syslinux Contains design models for syslinux, the program used to boot the CentOS Distribution installation media. See Section 2.23 [Directories trunk Identity Themes Models Default Distro Syslinux], page 25, for more information.

Anaconda Contains design models for Anaconda, the program used to install CentOS Distribution. See Section 2.15 [Directories trunk Identity Themes Models Default Distro Anaconda], page 23, for more information.

Firstboot Contains design models for the first boot program used to configure the maching onece installed. See Section 2.16 [Directories trunk Identity Themes Models Default Distro Firstboot], page 23, for more information.

Rhgb Contains design models for CentOS Graphical Boot, the program used to show the boot process from Grub to Display Manager. See Section 2.22 [Directories trunk Identity Themes Models Default Distro Rhgb], page 25, for more information.

Gdm Contains design models for GNOME Display Manager, the program used to log into the manchine once installed and configured. See Section 2.17 [Directories trunk Identity Themes Models Default Distro Gdm], page 23, for more information.

Kdm Contains design models for KDE Display Manager, the program used to log into the manchine once installed and configured. See Section 2.20 [Directories trunk Identity Themes Models Default Distro Kdm], page 24, for more information.

Grub Contains design models for GRUB (Grand Unified Boot Loader), the program used to boot the machine into an operating system. See Section 2.20 [Directories trunk Identity Themes Models Default Distro Kdm], page 24, for more information.

Gsplash Contains design models for GNOME splash, the program used to show the progress information while user's graphical session is loading. See Section 2.19 [Directories trunk Identity Themes Models Default Distro Gsplash], page 24, for more information.

Ksplash Contains design models for KDE splash, the program used to show the progress information while user's graphical session is loading. See Section 2.21 [Directories trunk Identity Themes Models Default Distro Ksplash], page 24, for more information.

2.14.2 Description

The CentOS Distribution visual style is controlled by image files. These image files are packaged inside The CentOS Distribution and made visible once such packages are installed and executed. The way to go for changing The CentOS Distribution visual style is changing all those image files to add the desired visual style first and later, repackage them to make them available inside the final iso files of CentOS Distribution.

2.14.3 Usage

This directory provides organizationl structure to store default design models for CentOS Themes of CentOS Distribution and so it should be considered to be used.

When a new component is added to CentOS Distribution, this is the directory you need to go for specifying design models for image files inside such component.

The procedure to follow is creatig a directory for each component using its very same name (e.g., the directory '`Anaconda`' stores image files for Anaconda component, the installer program). Inside the directory, you need to create one scalable vector graphic for each image file inside the component you want to produce images for. This, in order to set image dimensions, image file-name, position of trademarks in the final image, translation markers and whatever common information you need to have specified in them when rendered by `centos-art` script.

Sometimes, between major releases, image files inside packages can be added, removed or just change their names. In order to describe such image files variations, the design models directory structure is organized in the same way the file variations are introduced (i.e., through

The CentOS Project Release Schema). So, each major release of CentOS Distribution does have its own design model directory structure in this directory.

When a whole package is removed from one or all CentOS Distribution major releases, the design models directory structure releated to it is no longer used. However it could be very useful for historical reasons. Also, someone could feel motivated enough to keep himself documenting it or supporting it for whatever reason.

2.14.4 See also

2.15 The 'trunk/Identity/Themes/Models/Default/Distro/Anaconda' Directory

2.15.1 Goals

- ...

2.15.2 Description

2.15.3 Usage

2.15.4 See also

2.16 The 'trunk/Identity/Themes/Models/Default/Distro/Firstboot' Directory

2.16.1 Goals

- ...

2.16.2 Description

- ...

2.16.3 Usage

- ...

2.16.4 See also

2.17 The 'trunk/Identity/Themes/Models/Default/Distro/Gdm' Directory

2.17.1 Goals

- ...

2.17.2 Description

- ...

2.17.3 Usage

- ...

2.17.4 See also

2.18 The 'trunk/Identity/Themes/Models/Default/Distro/Grub' Directory

2.18.1 Goals

- ...

2.18.2 Description

- ...

2.18.3 Usage

- ...

2.18.4 See also

2.19 The 'trunk/Identity/Themes/Models/Default/Distro/Gsplash' Directory

2.19.1 Goals

- ...

2.19.2 Description

- ...

2.19.3 Usage

- ...

2.19.4 See also

2.20 The 'trunk/Identity/Themes/Models/Default/Distro/Kdm' Directory

2.20.1 Goals

- ...

2.20.2 Description

- ...

2.20.3 Usage

- ...

2.20.4 See also

2.21 The 'trunk/Identity/Themes/Models/Default/Distro/Ksplash' Directory

2.21.1 Goals

- ...

2.21.2 Description

- ...

2.21.3 Usage

- ...

2.21.4 See also

2.22 The ‘trunk/Identity/Themes/Models/Default/Distro/Rhgb’ Directory

2.22.1 Goals

- ...

2.22.2 Description

- ...

2.22.3 Usage

- ...

2.22.4 See also

2.23 The ‘trunk/Identity/Themes/Models/Default/Distro/Syslinux’ Directory

2.23.1 Goals

- ...

2.23.2 Description

- ...

2.23.3 Usage

- ...

2.23.4 See also

2.24 The ‘trunk/Identity/Themes/Models/Default/Posters’ Directory

2.24.1 Goals

- ...

2.24.2 Description

- ...

2.24.3 Usage

- ...

2.24.4 See also

2.25 The ‘trunk/Identity/Themes/Motifs’ Directory

2.25.1 Goals

The '`trunk/Identity/Themes/Motifs`' directory exists to:

- Organize CentOS themes' artistic motifs.

2.25.2 Description

The artistic motif of theme is a graphic design component that provides the visual style of themes, it is used as pattern to connect all visual manifestations inside one unique theme.

Artistic motifs are based on conceptual ideas. Conceptual ideas bring the motivation, they are fuel for the engines of human imagination. Good conceptual ideas may produce good motivation to produce almost anything, and art works don't escape from it.

'`TreeFlower`'
: CentOS like trees, has roots, trunk, branches, leaves and flowers. Day by day they work together in freedom, ruled by the laws of nature and open standards, to show the beauty of its existence.

'`Modern`' Modern, squares and circles flowing up.

If you have new conceptual ideas for CentOS, then you can say that you want to create a new artistic motif for CentOS. To create a new artistic motif you need to create a directory under '`Identity/Themes/Motifs/`' using a name coherent with your conceptual idea. That name will be the name of your artistic motif. If possible, when creating new conceptual ideas for CentOS, think about what CentOS means for you, what does it makes you feel, take your time, think deep, and share; you can improve the idea as time goes on.

Once you have defined a name for your theme, you need to create the motif structure of your theme. The motif structure is the basic direcotry structure you'll use to work your ideas. Here is where you organize your graphic design projects.

To add a new motif structure to CentOS Artwork Repository, you need to use the `centos-art` command line in the '`Identity/Themes/Motifs/`' directory as described below:

```
centos-art add --motif=ThemeName
```

The previous command will create the basic structure of themes for you. The basic structure produced by `centos-art` command is illustrated in the following figure:

```
trunk/Identity/Themes/Motifs/$ThemeName/
|-- Backgrounds
|   |-- Img
|   `-- Tpl
|-- Info
|   |-- Img
|   `-- Tpl
|-- Palettes
`-- Screenshots
```

2.25.3 Usage

When designing artistic motifs for CentOS, consider the following recommendations:

- Give a unique (case-sensitive) name to your Motif. This name is used as value wherever theme variable (**$THEME**) or translation marker (**=THEME=**) is. Optionally, you can add a description about inspiration and concepts behind your work.
- Use the location '`trunk/Identity/Themes/Motifs/$THEME/`' to store your work. If it doesn't exist create it. Note that this require you to have previous commit access in CentOS Artwork Repository.

- The CentOS Project is using the blue color (**#204c8d**) as base color for its corporate visual identity. Use such base corporate color information as much as possible in your artistic motif designs.
- Try to make your design fit one of the theme models.
- Feel free to make your art enterprise-level and beautiful.
- Add the following information on your artwork (both in a visible design area and document metadata):
  - The name (or logo) of your artistic motif.
  - The copyright sentence: **Copyright (C) YEAR YOURNAME**
  - The license under which the work is released. All CentOS Art works are released under Creative Common Share-Alike License 3.0 (`http://creativecommons.org/licenses/by-sa/3.0/`).

2.25.4 See also

The '`Backgrounds/`' directory is used to organize artistic motif background images and the projects used to build those images.

Background images are linked (using the **import** feature of Inkscape) inside almost all theme art works. This structure let you make centralized changes on the visual identity and propagate them quickly to other areas.

In this configuration you design background images for different screen resolutions based on the theme artistic motif.

You may create different artistic motifs propositions based on the same conceptual idea. The conceptual idea is what defines a theme. Artistic motifs are interpretations of that idea.

Inside this directory artistic motifs are organized by name (e.g., TreeFlower, Modern, etc.).

Each artistic motif directory represents just one unique artistic motif.

The artistic motif is graphic design used as common pattern to connect all visual manifestations inside one unique theme. The artistic motif is based on a conceptual idea. Artistic motifs provide visual style to themes.

Designing artistic motifs is for anyone interested in creating beautiful themes for CentOS. When building a theme for CentOS, the first design you need to define is the artistic motif.

Inside CentOS Artwork Repository, theme visual styles (a.k.a., artistic motifs) and theme visual structures (a.k.a., design models) are two different working lines. When you design an artistic motif for CentOS you concentrate on its visual style, and eventualy, use the `centos-art` command line interface to render the visual style, you are currently producing, against an already-made theme model in order to produce the final result. Final images are stored under '`Motifs/`' directory using the model name, and the model directory structure as reference.

The artistic motif base structure is used by `centos-art` to produce images automatically. This section describes each directory of CentOS artistic motif base structure.

The '`Backgrounds/`' directory is probably the core component, inside '`Motifs/`' directory structure. Inside '`Backgrounds/`' directory you produce background images used by almost all theme models (e.g., Distribution, Websites, Promotion, etc.). The '`Backgrounds/`' directory can contain subdirectories to help you organize the design process.

2.26 The '`trunk/Identity/Themes/Motifs/Flame`' Directory

2.26.1 Goals

This section describes the *Flame* artistic motif. This section may be useful for anyone interested in reproducing the *Flame* artistic motif, or in creating new artistic motifs for The CentOS Project corporate visual identity.

2.26.2 Description

The *Flame* artistic motif was built using the flame filter of Gimp 2.2 in CentOS 5.5.

The flame filter of Gimp can produce stunning, randomly generated fractal patterns. The flame filter of Gimp gives us a great oportunity to reduce the time used to produce new artistic motifs, because of its "randomly generated" nature. Once the artistic motif be created, it is propagated through all visual manifestations of CentOS Project corporate visual identity using the '`centos-art.sh`' script (see Section 2.31 [Directories trunk Scripts], page 34) inside the CentOS Artwork Repository.

To set the time intervals between each new visual style production, we could reuse the CentOS distribution major release schema. I.e., we could produce a new visual style, every two years, based on a new "randomly generated" flame pattern, and publish the whole corporate visual identity (i.e., distribution stuff, promotion stuff, websites stuff, etc.) with the new major release of CentOS distribution all together at once.

Producing a new visual style is not one day's task. Once we have defined the artistic motif, we need to propagate it through all visual manifestations of The CentOS Project corporate visual identity. When we say that we could produce one new visual style every two years we really mean: to work two years long in order to propagate a new visual style to all visual manifestations of The CentOS Project corporate visual identity.

Obviously, in order to propagate one visual style to all different visual manifestations of The CentOS Project corporate visual identity, we need first to know which the visual manifestations are. To define which visual manifestations are inside The CentOS Project corporate visual identity is one of the goals the CentOS Artwork Repository and this documentation manual are both aimed to satisfy.

Once we define which the visual manifestation are, it is possible to define how to produce them, and this way, organize the automation process. Such automation process is one of the goals of '`centos-art.sh`' script.

With the combination of both CentOS Artwork Repository and '`centos-art.sh`' scripts we define work lines where translators, programmers, and graphic designers work together to distribute and reduce the amount of time employed to produce The CentOS Project monolithic corporate identity.

From a monolithic corporate visual identity point of view, notice that we are producing a new visual style for the same theme (i.e., *Flame*). It would be another flame design but still a flame design. This idea is very important to be aware of, because we are somehow "refreshing" the theme, not changing it at all.

This way, as we are "refreshing" the theme, we still keep oursleves inside the monolithic conception we are trying to be attached to (i.e., one unique name, and one unique visual style for all visual manifestations).

Producing artistic motifs is a creative process that may consume long time, specially for people without experienced knowledge on graphic design land. Using "randomly generated" conception to produce artistic motifs could be, practically, a way for anyone to follow in order to produce maintainable artistic motifs in few steps.

Due to the "randomly generated" nature of Flame filter, we find that *Flame* pattern is not always the same when we use *Flame* filter interface.

Using the same pattern design for each visual manifestation is essential in order to maintain the visual connection among all visual manifestations inside the same theme. Occasionally, we may introduce pattern variations in opacity, size, or even position but never change the pattern design itself, nor the color information used by images considered part of the same theme.

> **Important** When we design background images, which are considered part of the same theme, it is essential to use the same design pattern always. This is what

> makes theme images to be visually connected among themeselves, and so, the reason we use to define the word "theme" as: a set of images visually connected among themeselves.

In order for us to reproduce the same flame pattern always, *Flame* filter interface provides the '`Save`' and '`Open`' options. The '`Save`' option brings up a file save dialog that allows you to save the current Flame settings for the plug-in, so that you can recreate them later. The '`Open`' option brings up a file selector that allows you to open a previously saved Flame settings file.

The Flame settings we used in our example are saved in the file named '`800x600.xcf-flame.def`', inside the '`Backgrounds/Xcf`' directory structure.

2.26.3 See also

- See Section 2.25 [Directories trunk Identity Themes Motifs], page 25.
- See Section 2.10 [Directories trunk Identity Themes], page 18.
- See Section 2.4 [Directories trunk Identity], page 12.
- See Section 2.3 [Directories trunk], page 12.

2.27 The '`trunk/Identity/Themes/Motifs/Modern`' Directory

2.27.1 Goals

2.27.2 Description

- ...

2.27.3 Usage

- ...

2.27.4 See also

2.28 The '`trunk/Identity/Themes/Motifs/Pipes`' Directory

2.28.1 Goals

- ...

2.28.2 Description

- ...

2.28.3 Usage

- ...

2.28.4 See also

2.29 The '`trunk/Identity/Themes/Motifs/TreeFlower`' Directory

2.29.1 Goals

2.29.2 Description

2.29.3 Usage

2.29.4 See also

2.30 The 'trunk/Identity/Webenv' Directory

2.30.1 Goals

- ...

2.30.2 Description

The CentOS web environment is formed by a central web application —to cover base needs (e.g., per-major release information like release notes, lifetime, downloads, documentation, support, security advisories, bugs, etc.)— and many different free web applications —to cover specific needs (e.g., wiki, mailing lists, etc.)—.

The CentOS web environment is addressed to solve the following issues:

- One unique name and one unique visual style to all web applications used inside the web environment.
- One-step navigation to web applications inside the environment.
- High degree of customization to change the visual style of all web applications with few changes (e.g, updating just two or three images plus common style sheet [CSS] definitions).

The CentOS project is attached to a monolithic corporate visual identity (see Section 2.4 [Directories trunk Identity], page 12), where all visual manifestations have one unique name and one unique visual style. This way, the CentOS web environment has one unique name (the CentOS brand) and one unique visual style (the CentOS default theme) for all its visual manifestations, the web applications in this case.

Since a maintainance point of view, achiving the one unique visual style inside CentOS web environment is not a simple task. The CentOS web environment is built upon many different web applications which have different visual styles and different internal ways to customize their own visual styles. For example: MoinMoin, the web application used to support the CentOS wiki (http://wiki.centos.org/) is highly customizable but Mailman (in its 2.x.x serie), the web application used to support the CentOS mailing list, doesn't support[1] a customization system that separates presentation from logic, similar to that used by MoinMoin.

This visual style diversity complicates our goal of one unique visual style for all web applications. So, if we want one unique visual style for all web applications used, it is innevitable to modify the web applications in order to implement the CentOS one unique visual style customization in them. Direct modification of upstream applications is not convenient because upstream applications come with their one visual style and administrators take the risk of loosing all customization changes the next time the application be updated (since not all upstream web applications, used in CentOS web environment, separate presentation from logic).

To solve the "one unique visual style" issue, installation and actualization of web applications —used inside CentOS web environment— need to be independent from upstream web applications development line; in a way that CentOS web environment administrators can install and update web applications freely without risk of loosing the one unique visual style customization changes.

At the surface of this issue we can see the need of one specific yum repository to store CentOS web environment customized web applications.

2.30.2.1 Design model (without ads)

2.30.2.2 Design model (with ads)

2.30.2.3 HTML definitions

[1] The theme support of Mailman may be introduced in mailman-3.x.x release.

2.30.2.4 Controlling visual style

Inside CentOS web environment, the visual style is controlled by the following compenents:

Webenv header background
`trunk/Identity/Themes/Motifs/$THEME/Backgrounds/Img/1024x250.png`

CSS definitions
`trunk/Identity/Themes/Models/Default/Promo/Web/CSS/stylesheet.css`

2.30.2.5 Producing visual style

The visual style of CentOS web environment is defined in the following files:

```
trunk/Identity/Themes/Motifs/$THEME/Backgrounds/Xcf/1024x250.xcf
trunk/Identity/Themes/Motifs/$THEME/Backgrounds/Img/1024x250.png
trunk/Identity/Themes/Motifs/$THEME/Backgrounds/Img/1024x250-bg.png
trunk/Identity/Themes/Motifs/$THEME/Backgrounds/Tpl/1024x250.svg
```

As graphic designer you use '`1024x250.xcf`' file to produce '`1024x250-bg.png`' file. Later, inside '`1024x250.svg`' file, you use the '`1024x250-bg.png`' file as background layer to draw your vectorial design. When you consider you artwork ready, use the `centos-art.sh` script, as described below, to produce the visual style controller images of CentOS web environment.

```
centos-art render --entry=trunk/Identity/Themes/Motifs/$THEME/Backgrounds --filter='1024x
```

Once you have rendered required image files, changing the visual style of CentOS web environment is a matter of replacing old image files with new ones, inside webenv repository file system structure. The visual style changes will take effect the next time customization line of CentOS web applications be packaged, uploded, and installed from [webenv] or [webenv-test] repositories.

2.30.2.6 Navigation

Inside CentOS web environment, the one-step navegation between web applications is addressed using the web environment navigation bar. The web environment navigation bar contains links to main applications and is always visible no matter where you are inside the web environment.

2.30.2.7 Development and release cycle

The CentOS web environment development and relase cycle is described below:

Download

The first action is download the source code of web applications we want to use inside CentOS web environment.

> **Important** The source location from which web application are downloaded is very important. Use SRPMs from CentOS [**base**] and [**updates**] repositories as first choise, and third party repositories (e.g. RPMForge, EPEL, etc.) as last resource.

Prepare

Once web application source code has been downloaded, our duty is organize its files inside '`webenv`' version controlled repository.

When preparing the structure keep in mind that different web applications have different visual styles, and also different ways to implement it. A convenient way to organize the file system structure would be create one development line for each web application we use inside CentOS web environment. For example, consider the following file system structure:

```
https://projects.centos.org/svn/webenv/trunk/
|-- WebApp1/
|   |-- Sources/
|   |   `-- webapp1-0.0.1/
|   |-- Rpms/
|   |   `-- webapp1-0.0.1.rpm
|   |-- Srpms/
|   |   `-- webapp1-0.0.1.srpm
|   `-- Specs/
|       `-- webapp1-0.0.1.spec
|-- WebApp2/
`-- WebAppN/
```

Customize

Once web applications have been organized inside the version controlled repository file system, use subversion to create the CentOS customization development line of web applications source code. For example, using the above file system structure, you can create the customization development line of '`webapp1-0.0.1/`' with the following command:

```
svn cp trunk/WebApp1/Sources/webapp1-0.0.1 trunk/WebApp1/Sources/webapp1-0.0.1-
```

The command above creates the following structure:

```
https://projects.centos.org/svn/webenv/trunk/
|-- WebApp1/
|   |-- Sources/
|   |   |-- webapp1-0.0.1/
|   |   `-- webapp1-0.0.1-webenv/
|   |-- Rpms/
|   |   `-- webapp1-0.0.1.rpm
|   |-- Srpms/
|   |   `-- webapp1-0.0.1.srpm
|   `-- Specs/
|       `-- webapp1-0.0.1.spec
|-- WebApp2/
`-- WebAppN/
```

In the above structure, the '`webapp1-0.0.1-webenv/`' directory is the place where you customize the visual style of '`webapp1-0.0.1/`' web application.

> **Tip** Use the `diff` command of Subversion between CentOS customization and upstream development lines to know what you are changing exactly.

Build packages

When web application has been customized, build the web application RPM and SRPM using the source location with '`-webenv`' prefix.

```
https://projects.centos.org/svn/webenv/trunk/
|-- WebApp1/
|   |-- Sources/
|   |   |-- webapp1-0.0.1/
|   |   `-- webapp1-0.0.1-webenv/
|   |-- Rpms/
|   |   |-- webapp1-0.0.1.rpm
|   |   `-- webapp1-0.0.1-webenv.rpm
|   |-- Srpms/
```

```
|   |   |-- webapp1-0.0.1.srpm
|   |   `-- webapp1-0.0.1-webenv.srpm
|   `-- Specs/
|       |-- webapp1-0.0.1.spec
|       `-- webapp1-0.0.1-webenv.spec
|-- WebApp2/
`-- WebAppN/
```

Release for testing

When the customized web application has been packaged, make packages available for testing and quality assurance. This can be achives using a [webenv-test] yum repository.

> **Note** The [webenv-test] repository is not shipped inside CentOS distribution default yum configuraiton. In order to use [webenv-test] repository you need to configure it first.

If some problem is found to install/update/use the customized version of web application, the problem is notified somewhere (a bugtracker maybe) and the customization face is repated in order to fix the problem. To release the new package add a number after '`-webenv`' prefix. For example, if some problem is found in '`webapp1-0.0.1-webenv.rpm`', when it be fixed the new package will be named '`webapp1-0.0.1-webenv-1.rpm`'. If a problem is found in '`webapp1-0.0.1-webenv-1.rpm`', when it be fixed the new package will be named '`webapp1-0.0.1-webenv-2.rpm`', and so on.

The "customization — release for testing" process is repeated until CentOS quality assurance team considers the package is ready for production.

Release for production

When customized web application packages are considered ready for production they are moved from [webenv-test] to [webenv] repository. This action is commited by CentOS quality assurance team.

> **Note** The [webenv] repository is not shipped inside CentOS distribution default yum configuraiton. In order to use [webenv] repository you need to configure it first.

2.30.2.8 The [webenv-test] repository

`/etc/yum.repos.d/CentOS-Webenv-test.repo`

```
[webenv-test]
name=CentOS-$releasever - Webenv-test
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=webenv-t
#baseurl=http://mirror.centos.org/centos/$releasever/webenv-test/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-$releasever
enabled=1
priority=10
```

2.30.2.9 The [webenv] repository

`/etc/yum.repos.d/CentOS-Webenv.repo`

```
[webenv]
name=CentOS-$releasever - Webenv
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=webenv
#baseurl=http://mirror.centos.org/centos/$releasever/webenv/$basearch/
```

```
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-$releasever
enabled=1
priority=10
```

2.30.2.10 Priority configuration

Both [webenv] and [webenv-test] repositories update packages inside CentOS [base] and CentOS [updates] repositories.

2.30.3 Usage

- ...

2.30.4 See also

2.31 The 'trunk/Scripts' Directory

2.31.1 Goals

The '`trunk/Scripts`' directory exists to organize the trunk development line of '`centos-art.sh`' automation script. The '`centos-art.sh`' script standardizes tasks you need to do frequently inside CentOS Artwork Repository.

2.31.2 Description

The best way to understand '`centos-art.sh`' automation script is studying its source code. However, as start point, you may prefer to read an introductory resume before diving into the source code details.

The '`centos-art.sh`' script is written in Bash. Most tasks, inside '`centos-art.sh`' script, have been organized in many specific functionalities that you can invoke from the `centos-art` command-line interface.

When you type the `centos-art` command in your terminal, the operating system trys to execute that command. In order to execute the command, the operating system needs to know where it is, so the operating system uses the *PATH* environment variable to look for that command location. If your system was prepared to use CentOS Artwork Repository correctly (— **Removed**(pxref:trunk Scripts Bash Functions Verify) —), you should have a symbolic link inside '`~/bin/`' directory that points to the '`centos-art.sh`' script file. As '`~/bin/`' directory is, by default, inside *PATH* environment variable, the execution of `centos-art` command runs the '`centos-art.sh`' script.

When '`centos-art.sh`' script is executed, the first it does is executing the '`trunk/Scripts/Bash/initEnvironment.sh`' script to initialize global variables (e.g., `gettext` variables) and global function scripts. Global function scripts are located inside '`trunk/Scripts/Bash/Functions`' directory and their file names begin with '`cli`'. Global function scripts provide common functionalities that can be used anywhere inside '`centos-art.sh`' script execution environment.

Once global variables and function scripts have been loaded, '`centos-art.sh`' script executes the `cli` global function from '`cli.sh`' function script to retrive command-line arguments and define some default values that may be used later by specific function scripts (— **Removed**(pxref:trunk Scripts Bash Functions) —).

As convenction, the '`centos-art.sh`' command-line arguments have the following format:

```
centos-art function path/to/dir --options
```

In the above example, '`centos-art`' is the command you use to invoke '`centos-art.sh`' script. The '`arg1`' is required and represents the functionality you want to perform (e.g.,

‘verify’, ‘render’, ‘locale’, ‘manual’, etc.). The remaining arguments are modifiers to ‘arg1’. The ‘--arg2’ definition is required and represets, specifically, the action inside the functionality you want to perform. The ‘--arg3’ and on, are optional.

Once command-line arguments have been retrived, the ‘centos-art.sh’ script loads specific functionalities using the ‘cli_getFunctions.sh’ function script. Only one specific functionality can be loaded at one script execution I.e., you run centos-art.sh script to run just one functionality.

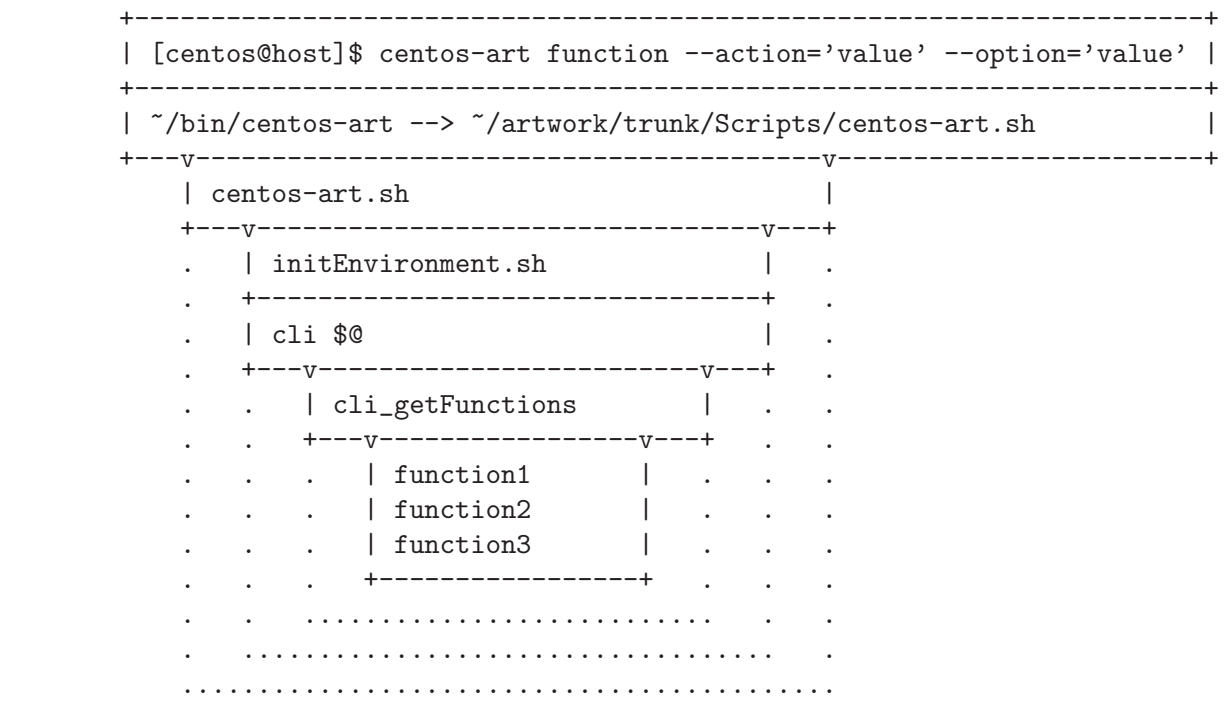

Figure 2.1: The functionalities initialization environment.

Functionalities are implemented by means of actions. Once the functionality has been initiazalized, actions initialization take place for that functionality. Actions initialization model is very similar to functions initialization model. But with the difference, that actions are loaded inside function environment, and so, share variables and functions defined inside function environment.

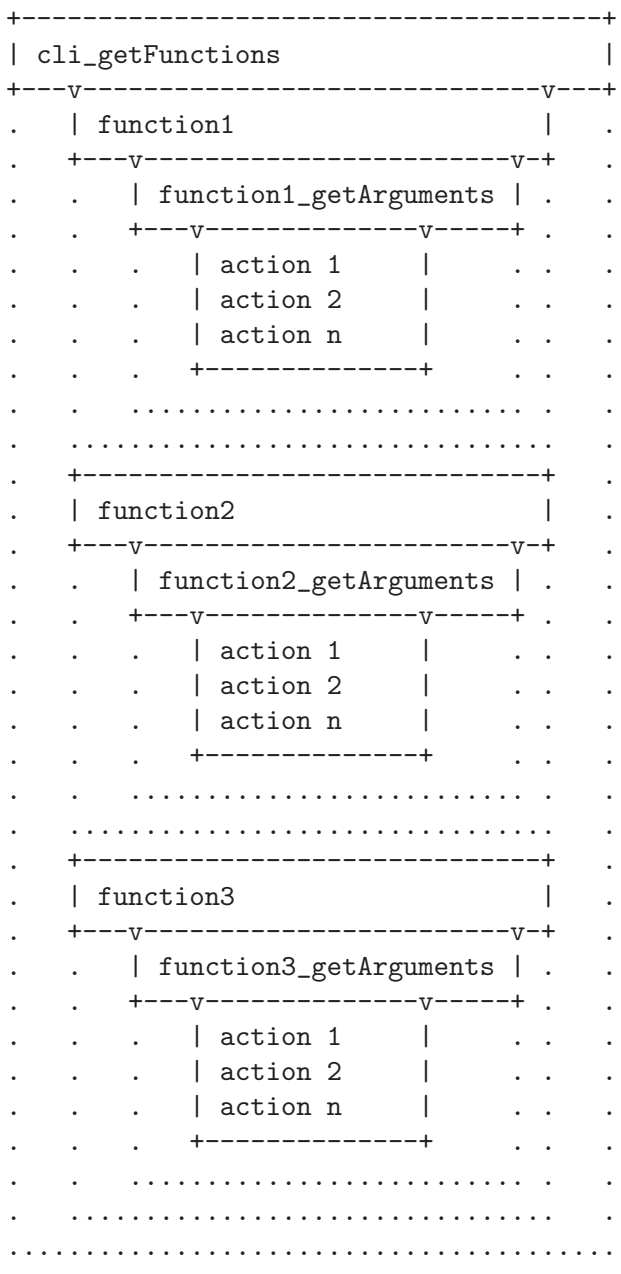

Figure 2.2: The actions initialization environment.

2.31.3 Usage

The '`centos-art.sh`' script usage information is described inside each specific function documentation (— **Removed**(pxref:trunk Scripts Bash Functions) —).

2.31.4 See also

2.32 The '`trunk/Scripts/Functions`' Directory

2.32.1 Goals

The '`trunk/Scripts/Bash/Functions`' directory exists to organize '`centos-art.sh`' specific functionalities.

2.32.2 Description

The specific functions of 'centos-art.sh' script are designed with "Software Toolbox" philosophy (see ⟨undefined⟩ [Toolbox introduction], page ⟨undefined⟩) in mind: each program "should do one thing well". Inside 'centos-art.sh' script, each specific functionality is considered a program that should do one thing well. Of course, if you find that they still don't do it, feel free to improve them in order for them to do so.

The specific functions of 'centos-art.sh' script are organized inside specific directories under 'trunk/Scripts/Bash/Functions' location. Each specific function directory should be named as the function it represents, with the first letter in uppercase. For example, if the function name is render, the specific function directory for it would be 'trunk/Scripts/Bash/Functions/Render'.

To better understand how specific functions of 'centos-art.sh' script are designed, lets create one function which only goal is to output different kind of greetings to your screen.

When we create specific functions for 'centos-art.sh' script it is crucial to know what these functions will do exactly and if there is any function that already does what we intend to do. If there is no one, it is good time to create them then. Otherwise, if functionalities already available don't do what you exactly expect, contact their authors and work together to improve them.

> **Tip** Join CentOS developers mailing list centos-art@centos.org to share your ideas.

It is also worth to know what global functions and variables do we have available inside 'centos-art.sh' script, so advantage can be taken from them. Global variables are defined inside global function scripts. Global functions scripts are stored immediatly under 'trunk/Scripts/Bash/Functions' directory, in files begining with 'cli' prefix.

OK, let's begin with our functionality example.

What function name do we use? Well, lets use greet. Note that 'hello' word is not a verb; but an expression, a kind of greeting, an interjection specifically. In contrast, 'greet' is a verb and describes what we do when we say 'Hello!', 'Hi!', and similar expressions.

So far, we've gathered the following function information:

```
Name: greet
Path: trunk/Scripts/Bash/Functions/Greet
File: trunk/Scripts/Bash/Functions/Greet/greet.sh
```

The 'greet.sh' function script is the first file 'centos-art.sh' script loads when the 'greet' functionality is called using commands like 'centos-art greet --hello='World''. The 'greet.sh' function script contains the greet function definition.

Inside 'centos-art.sh' script, as convenction, each function script has one top commentary, followed by one blank line, and then one function defintion below it only.

Inside 'centos-art.sh' script functions, top commentaries have the following components: the functionality description, one-line for copyright note with your personal information, the license under which the function source code is released —the 'centos-art.sh' script is released as GPL, so do all its functions—, the Id keyword of Subversion is later expanded by svn propset command.

In our greet function example, top commentary for 'greet.sh' function script would look like the following:

```
#!/bin/bash
#
# greet.sh -- This function outputs different kind of greetings to
# your screen. Use this function to understand how centos-art.sh
```

```
# script specific functionalities work.
#
# Copyright (C) YEAR YOURFULLNAME
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
# USA.
#
# ------------------------------------------------------------------
# $Id$
# ------------------------------------------------------------------
```

After top commentary, separated by one blank line, the `greet` function definition would look like the following:

```
function greet {

    # Define global variables.

    # Define command-line interface.
    greet_getActions

}
```

The first definition inside `greet` function, are global variables that will be available along `greet` function execution environment. This time we didn't use global variable definitions for `greet` function execution environment, so we left that section empty.

Later, we call `greet_getActions` function to define the command-line interface of `greet` functionality. The command-line interface of `greet` functionality defines what and how actions are performed, based on arguments combination passed to '`centos-art.sh`' script.

```
function greet_getActions {

    case "$ACTIONNAM" in

        --hello )
            greet_doHello
            ;;

        --bye )
            greet_doBye
            ;;

        * )
```

```
            cli_printMessage "‘gettext "The option provided is not valid."‘"
            cli_printMessage "$(caller)" 'AsToKnowMoreLine'

    esac

}
```

The *ACTIONNAM* global variable is defined in ‘`cli.sh`’ function script and contains the value passed before the equal sign (i.e., ‘`=`’) in the second command-line argument of ‘`centos-art.sh`’ script. For example, if the second command-line argument is ‘`--hello='World'`’, the value of *ACTIONNAM* variable would be ‘`--hello`’. Using this configuration let us deside which action to perform based on the action name passed to ‘`centos-art.sh`’ script as second argument.

The `greet` function definition makes available two valid greetings through ‘`--hello`’ and ‘`--bye`’ options. If no one of them is provided as second command-line argument, the ‘`*`’ case is evaluated instead.

The ‘`*`’ case and its two lines further on should always be present in ‘`_getActions.sh`’ function scripts, no matter what specific functionality you are creating. This convenction helps the user to find out documentation about current functionality in use, when no valid action is provided.

The `greet_doHello` and `greet_doBye` function definitions are the core of `greet` specific functionality. In such function definitions we set what our `greet` function really does: to output different kinds of greetings.

```
function greet_doHello {

    cli_printMessage "‘gettext "Hello"‘ $ACTIONVAL"

}
```

The `greet_doHello` function definition is stored in ‘`greet_doHello.sh`’ function script.

```
function greet_doBye {

    cli_printMessage "‘gettext "Goodbye"‘ $ACTIONVAL"

}
```

The `greet_doBye` function definition is stored in the ‘`greet_doBye.sh`’ function script.

Both ‘`greet_doHello.sh`’ and ‘`greet_doBye.sh`’ function scripts are stored inside `greet` function directory path (i.e. ‘`trunk/Scripts/Bash/Functions/Greet`’).

The *ACTIONVAL* global variable is defined in ‘`cli.sh`’ function script and contains the value passed after the equal sign (i.e., ‘`=`’) in the second command-line argument of ‘`centos-art.sh`’ script. For example, if the second command-line argument is ‘`--hello='World'`’, the value of *ACTIONVAL* variable would be ‘`World`’ without quotes.

Let's see how `greet` specific functionality files are organzied under `greet` function directory. To see file organization we use the `tree` command:

```
trunk/Scripts/Bash/Functions/Greet
|-- greet_doBye.sh
|-- greet_doHello.sh
|-- greet_getActions.sh
‘-- greet.sh
```

To try the `greet` specific functionality we've just created, pass the function name (i.e., ‘`greet`’) as first argument to ‘`centos-art.sh`’ script, and any of the valid options as second argument. Some examples are illustrated below:

```
[centos@projects ~]$ centos-art greet --hello='World'
Hello World
[centos@projects ~]$ centos-art greet --bye='World'
Goodbye World
[centos@projects ~]$
```

The word '`World`' in the examples above can be anything. In fact, change it to have a little fun.

Now that we have a specific function that works as we expect, it is time to document it. To document `greet` specific functionality, we use its directory path and the `manual` functionality (— **Removed**(pxref:trunk Scripts Bash Functions Manual) —) of '`centos-art.sh`' script, just as the following command illustrates:

```
centos-art manual --edit=trunk/Scripts/Bash/Functions/Greet
```

To have a well documented function helps user to understand how your function really works, and how it should be used. When no valid action is passed to a function, the '`centos-art.sh`' script uses the function documentation entry as vehicle to communicate which the valid functions are. When no documentation entry exists for a function, the '`centos-art.sh`' script informs that no documentation entry exists for such function and requests user to create it right at that time.

Now that we have documented our function, it is time to translate its output messages to different languages. To translate specific functionality output messages to different languages we use the `locale` functionality (— **Removed**(pxref:trunk Scripts Bash Functions Locale) —) of '`centos-art.sh`' script, just as the following command illustrates:

```
centos-art locale --edit
```

> **Warning** To translate output messages in different languages, your system locale information —as in `LANG` environment variable— must be set to that locale you want to produce translated messages for. For example, if you want to produce translated messages for Spanish language, your system locale information must be set to '`es_ES.UTF-8`', or similar, first.

Well, it seems that our example is rather complete by now.

In `greet` function example we've described so far, we only use `cli_printMessage` global function in action specific function definitions in order to print messages, but more interesting things can be achieved inside action specific function definitions. For example, if you pass a directory path as action value in second argument, you could retrive a list of files from therein, and process them. If the list of files turns too long or you just want to control which files to process, you could add the third argument in the form '`--filter='regex'`' and reduce the amount of files to process using a regular expression pattern.

The `greet` function described in this section may serve you as an introduction to understand how specific functionalities work inside '`centos-art.sh`' script. With some of luck this introduction will also serve you as motivation to create your own '`centos-art.sh`' script specific functionalities.

By the way, the `greet` functionality doesn't exist inside '`centos-art.sh`' script yet. Would you like to create it?

2.32.3 Usage

2.32.3.1 Global variables

The following global variables of '`centos-art.sh`' script, are available for you to use inside specific functions:

`TEXTDOMAIN` [Variable]
Default domain used to retrieve translated messages. This value is set in '`initFunctions.sh`' and shouldn't be changed.

`TEXTDOMAINDIR` [Variable]
Default directory used to retrieve translated messages. This value is set in '`initFunctions.sh`' and shouldn't be changed.

`FUNCNAM` [Variable]
Define function name.

Function names associate sets of actions. There is one set of actions for each unique function name inside '`centos-art.sh`' script.

Dunction names are passed as first argument in '`centos-art.sh`' command-line interface. For example, in the command '`centos-art render --entry=path/to/dir --filter=regex`', the *ACTION* passed to '`centos-art.sh`' script is '`render`'.

When first argument is not provided, the '`centos-art.sh`' script immediatly ends its execution.

`FUNCDIR` [Variable]

`FUNCDIRNAME` [Variable]

`FUNCSCRIPT` [Variable]

`FUNCCONFIG` [Variable]

`ACTIONNAM` [Variable]
Define action name.

Each action name identifies an specific action to perform, inside an specific function.

Action name names aare passed as second argument in '`centos-art.sh`' command-line interface. For example, in the command '`centos-art render --entry=path/to/dir --filter=regex`', the *ACTIONNAM* passed to '`centos-art.sh`' script is '`--entry`'.

When second argument is not provided, the '`centos-art.sh`' script immediatly ends its execution.

`ACTIONVAL` [Variable]
Define action value.

Action values are associated to just one action name. Action values contain the working copy entry over which its associated action will be performed in. Working copy entries can be files or directories inside the working copy.

`REGEX` [Variable]
Define regular expression used as pattern to build the list of files to process.

By default, *REGEX* variable is set to `.+` to match all files.

Functions that need to build a list of files to process use the option '`--filter`' to redefine *REGEX* variable default value, and so, control the amount of files to process.

`ARGUMENTS` [Variable]
Define optional arguments.

Optional arguments, inside '`centos-art.sh`' script, are considered as all command-line arguments passed to '`centos-art.sh`' script, from third argument position on. For example, in the command '`centos-art render --entry=path/to/dir --filter=regex`' , the optional arguments are from '`--filter=regex`' argument on.

Optional arguments are parsed using `getopt` command through the following base construction:

```
# Define short options we want to support.
local ARGSS=""

# Define long options we want to support.
local ARGSL="filter:,to:"

# Parse arguments using getopt(1) command parser.
cli_doParseArguments

# Reset positional parameters using output from (getopt) argument
# parser.
eval set -- "$ARGUMENTS"

# Define action to take for each option passed.
while true; do
    case "$1" in
        --filter )
            REGEX="$2"
            shift 2
            ;;
        --to )
            TARGET="$2"
            shift 2
            ;;
        * )
            break
    esac
done
```

Optional arguments provide support to command options inside '`centos-art.sh`' script. For instance, consider the Subversion (`svn`) command, where there are many options (e.g., '`copy`', '`delete`', '`move`', etc), and inside each option there are several modifiers (e.g., '`--revision`', '`--message`', '`--username`', etc.) that can be combined one another in their short or long variants.

The *ARGUMENTS* variable is used to store arguments passed from command-line for later use inside '`centos-art.sh`' script. Storing arguments is specially useful when we want to run a command with some specific options from them. Consider the following command:

```
centos-art path --copy=SOURCE --to=TARGET --message="The commit message goes here." --u
```

In the above command, the '`--message`', and '`--username`' options are specific to `svn copy` command. In such cases, options are not interpreted by '`centos-art.sh`' script itself. Instead, the '`centos-art.sh`' script uses `getopt` to retrive them and store them in the *ARGUMENTS* variable for later use, as described in the following command:

```
# Build subversion command to duplicate locations inside the
# workstation.
eval svn copy $SOURCE $TARGET --quiet $ARGUMENTS
```

When `getopt` parses *ARGUMENTS*, we may use short options (e.g., '`-m`') or long options (e.g., '`--message`'). When we use short options, arguments are separated by one space from the option (e.g., '`-m 'This is a commit message.'`'). When we use long options arguments are separated by an equal sign ('`=`') (e.g., '`--message='This is a commit message'`').

In order for `getopt` to parse *ARGUMENTS* correctly, it is required to provide the short and long definition of options that will be passed or at least supported by the command performing the final action the function script exists for.

As convenction, inside '`centos-art.sh`' script, short option definitions are set in the *ARGSS* variable; and long option definitions are set in the *ARGSL* variable.

When you define short and long options, it may be needed to define which of these option arguments are required and which not. To define an option argument as required, you need to set one colon '`:`' after the option definition (e.g., '`-o m: -l message:`'). On the other hand, to define an option argument as not required, you need to set two colons '`::`' after the option definition (e.g., '`-o m:: -l message::`').

`EDITOR` [Variable]

Default text editor.

The '`centos-art.sh`' script uses default text `EDITOR` to edit pre-commit subversion messages, translation files, configuration files, script files, and similar text-based files.

If `EDITOR` environment variable is not set, '`centos-art.sh`' script uses '`/usr/bin/vim`' as default text editor. Otherwise, the following values are recognized by '`centos-art.sh`' script:

- '`/usr/bin/vim`'
- '`/usr/bin/emacs`'
- '`/usr/bin/nano`'

If no one of these values is set in `EDITOR` environment variable, '`centos-art.sh`' uses '`/usr/bin/vim`' text editor by default.

2.32.3.2 Global functions

Function scripts stored directly under '`trunk/Scripts/Bash/Functions/`' directory are used to define global functions. Global functions can be used inside action specific functionalities and or even be reused inside themselves. This section provides introductory information to global functions you can use inside '`centos-art.sh`' script.

`cli_checkActionArguments` [Function]

Validate action value (*ACTIONVAL*) variable.

The action value variable can take one of the following values:

1. Path to one directory inside the local working copy,
2. Path to one file inside the local working copy,

If another value different from that specified above is passed to action value variable, the '`centos-art.sh`' script prints an error message and ends script execution.

`cli_checkFiles` *FILE* [*TYPE*] [Function]

Verify file existence.

`cli_checkFiles` receives a *FILE* absolute path and performs file verification as specified in *TYPE*. When *TYPE* is not specified, `cli_checkFiles` verifies *FILE* existence, no matter what kind of file it be. If *TYPE* is specified, use one of the following values:

'`d`'
'`directory`'
: Ends script execution if *FILE* is not a directory.

  When you verify directories with cli_checkFiles, if directory doesn't exist, '`centos-art.sh`' script asks you for confirmation in order to create that directory. If you answer positively, '`centos-art.sh`' script creates that directory and continues script flows normally. Otherwise, if you answer negatively,

'`centos-art.sh`' ends script execution with an error and documentation message.

'`f`'

'`regular-file`'
: Ends script execution if *FILE* is not a regular file.

'`h`'

'`symbolic-link`'
: Ends script execution if *FILE* is not a symbolic link.

'`x`'

'`execution`'
: Ends script execution if *FILE* is not executable.

'`fh`' Ends script execution if *FILE* is neither a regular file nor a symbolic link.

'`fd`' Ends script execution if *FILE* is neither a regular file nor a directory.

'`isInWorkingCopy`'
: Ends script execution if *FILE* is not inside the working copy.

As default behaviour, if *FILE* passes all verifications, '`centos-art.sh`' script continues with its normal flow.

`cli_commitRepoChanges` [*LOCATION*] [Function]

Syncronize changes between repository and working copy.

The `cli_commitRepoChanges` function brings changes from the central repository down to the working copy—using `svn update`—, checks the working copy changes—using `svn status` command—, prints status report—using both `svn update` and `svn status` commands output, and finally, commits recent changes from the working copy up to the repository—using `svn commit` command—.

Previous to commit the working copy changes up to the central repository, the `cli_commitRepoChanges` function asks you to verify changes—using `svn diff` command—, and later, another confirmation question is shown to be sure you really want to commit changes up to central repository.

If *LOCATION* argument is not specified, the value of *ACTIONVAL* variable is used as reference instead.

```
------------------------------------------------------------------------
--> Bringing changes from the repository into the working copy
--> Checking changes in the working copy
------------------------------------------------------------------------
Added           0 file from the repository.
Deleted         0 file from the repository.
Updated         0 file from the repository.
Conflicted      0 file from the repository.
Merged          0 file from the repository.
Modified        4 files from the working copy.
Unversioned     0 file from the working copy.
Deleted         0 file from the working copy.
Added           0 file from the working copy.
------------------------------------------------------------------------
```

Figure 2.3: The `cli_commitRepoChanges` function output.

Call the `cli_commitRepoChanges` function before or/and after calling functions that modify files or directories inside the working copy as you may need to.

`cli_doParseArguments` [Function]
Redefine arguments (*ARGUMENTS*) global variable using `getopt` command output. For more information about how to use `cli_doParseArguments` function, see *ARGUMENTS* variable description above.

`cli_doParseArgumentsReDef $@` [Function]
Initialize/reset arguments (*ARGUMENTS*) global variable using positional parameters variable (`$@`) as reference.

When we work inside function definitions, positional parameters are reset to the last function definition positional parameters. If you need to redefine positional parameters from one specific function, you need to call `cli_doParseArgumentsReDef` with the positional parameters variable (`$@`), set as first argument, to that specific function you want to redefine positional parameters at.

`cli_getArguments` [Function]
Initialize function name (*FUNCNAM*), action name (*ACTIONNAM*), and action value (*ACTIONVAL*) global variables, using positional parameters passed in `$@` variable.

The `cli_getArguments` function is called from `cli.sh` function script, using `cli` function positional parameters (i.e., the positional parameters passed as arguments in the command-line) as first function argument.

Once command-line positional parameters are accesible to '`centos-art.sh`' script execution evironment, `cli_getArguments` uses regular expression to retrive action variables from first and second argument. The first argument defines the value used as function name (*FUNCNAM*), and the second argument defines both values used as action name (*ACTIONNAM*) and action value (*ACTIONVAL*), respectively.

The first argument is a word in lower case. This word specifies the name of the functionality you want to use (e.g., '`render`' to render images, '`manual`' to work on documentation, and so on.)

The second argument has a long option style (e.g., '`--option=value`'). The '`--option`' represents the action name (*ACTIONNAM*), and the characters inbetween the equal sign ('`=`') and the first space character, are considered as the action value (*ACTIONVAL*). In order to provide action values with space characters inbetween you need to enclose action value with quotes like in '`--option='This is long value with spaces inbetween''`. Generally, action values are used to specify paths over which the action name acts on.

Once action related variables (i.e., *FUNCNAM*, *ACTIONNAM*, and *ACTIONVAL*) are defined and validated, `cli_getArguments` shifts the positional arguments to remove the first two arguments passed (i.e., those used to retrive action related variables) and redefine the arguments (*ARGUMENTS*) global variable with the new positional parameters information.

`cli_getFunctions` [Function]
Initialize funtionalities supported by '`centos-art.sh`' script.

Functionalities supported by '`centos-art.sh`' script are organized in functionality directories under '`trunk/Scripts/Bash/Functions/`' directory. Each functionality directory stores function scripts to the functionality such directory was created for. Function scripts contain function definitions. Function definitions contain several commands focused on achieving one specific task only (i.e., the one such functionality was created for).

In order for '`centos-art.sh`' script to recognize a functionality, such functionality needs to be stored under '`trunk/Scripts/Bash/Functions/`' in a directory written capitalized (i.e., the whole name is written in lowercase except the first character which is in uppercase). The directory where one specific functionality is stored is known as the '`functionality directory`'.

Inside each functionality directory, the functionalty itself is implemented through function scripts. Function scripts are organized in files independently one another and written in '`camelCase`' format with the function name as prefix. Separation between prefix and description is done using underscore ('`_`') character.

In order for '`centos-art.sh`' script to load functionalities correctly, function definition inside function scripts should be set using the '`function`' reserved word, just as in the following example:

```
function prefix_doSomething {

    # Do something here...

}
```

The above function definition is just a convenction we use, in order to make identification of function names easier read and automate by '`centos-art.sh`' script initialization commands, once '`centos-art.sh`' script determines which functionality directory to use. Specifically, in order to initialize and export functions, '`centos-art.sh`' script executes all function scripts inside the functionality directory, and later `grep` on them using a regular expression pattern, where the '`function`' reserved word is used as reference to retrive the function names and export them to '`centos-art.sh`' script execution environment, and so, make function definitions —from function scripts inside the functionality directory— available for further calls.

If the functionality specified in the command-line first argument doesn't have a functionality directory, '`centos-art.sh`' script considers the functionality provided in the command-line as invalid functionality and immediatly stops script execution with an error message.

In order to keep visual consistency among function scripts, please consider using the following function script design model as template for your own function scripts:

```
#!/bin/bash
#
# prefix_doSomething.sh -- This function illustrates function scripts
# design model you can use to create your own function scripts inside
# centos-art.sh script.
#
# Copyright (C) YEAR YOURFULLNAME
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
# USA.
#
# ------------------------------------------------------------------
```

```
# $Id$
# ----------------------------------------------------------------------

function prefix_doSomething {

    # Do something here...

}
```

`cli_getCountryCodes` [*FILTER*] [Function]
Output country codes supported by '`centos-art.sh`' script.

The `cli_getCountryCodes` function outputs a list with country codes as defined in ISO3166 standard. When *FILTER* is provided, `cli_getCountryCodes` outputs country codes that match *FILTER* regular expression pattern.

`cli_getCountryName` [*FILTER*] [Function]
Outputs country name supported by '`centos-art.sh`' script.

The `cli_getCountryName` function reads one language locale code in the format LL_CC and outputs the name of its related country as in ISO3166. If filter is specified, `cli_getCountryName` returns the country name that matches the locale code specified in *FILTER*, exactly.

`cli_getCurrentLocale` [Function]
Output current locale used by '`centos-art.sh`' script.

The `cli_getCurrentLocale` function uses `LANG` environment variable to build a locale pattern that is later applied to `cli_getLocales` function output in order to return the current locale that '`centos-art.sh`' script works with.

The current locale information, returned by `cli_getCurrentLocale`, is output from more specific to less specific. For example, if '`en_GB`' locale exists in `cli_getLocales` function output, the '`en_GB`' locale would take precedence before '`en`' locale.

Locale precedence selection is quite important in order to define the locale type we use for message translations. For example, if '`en_GB`' is used, we are also saying that the common language specification for English language (i.e., '`en`') is no longer used. Instead, we are using English non-common country-specific language specifications like '`en_AU`', '`en_BW`', '`en_GB`', '`en_US`', etc., for message translations.

Use `cli_getCurrentLocale` function to know what current locale information to use inside '`centos-art.sh`' script.

`cli_getFilesList` [*LOCATION*] [Function]
Output list of files to process.

The `cli_getFilesList` function uses *LOCATION* variable as source location to build a list of files just as specified by regular expression (*REGEX*) global variable. Essentially, what the `cli_getFilesList` function does is using `find` command to look for files in the location (*LOCATION*) just as posix-egrep regular expression (*REGEX*) specifies.

If *LOCATION* is not specified when `cli_getFilesList` function is called, the action value (*ACTIONVAL*) global variable is used as location value instead.

By default, if the regular expression (*REGEX*) global variable is not redefined after its first definition in the `cli` function, all files that match default regular expression value (i.e., '`.+`') will be added to the list of files to process. Otherwise, if you redefine the regular expression global variable after its first definition in the `cli` function and before calling `cli_getFilesList` function, the last value you specifed is used instead.

When you need to customize the regular expression (*REGEX*) global variable value inside a function, do not redefine the global variable (at least you be absolutly convinced you need to). Instead, set the regular expression global variable as '`local`' to the function you need a customized regular expression value for. If we don't redefine the regular expression global variable as local to the function, or use another name for the regular expression variable (which is not very convenient in order to keep the amount of names to remember low), you may experiment undesired concantenation issues that make your regular expression to be something different from that you expect them to be, specially if the function where you are doing the variable redefinition is called several times during the same script execution.

As result, the `cli_getFilesList` re-defines the value of *FILES* variable with the list of files the `find` command returned. As example, consider the following construction:

```
function prefix_doSomething {

    # Initialize the list of files to process.
    local FILES=''

    # Initialize location.
    local LOCATION=/home/centos/artwork/trunk/Identity/Themes/Models/Default

    # Re-define regular expression to match scalable vector graphic
    # files only. Note how we use the global value of REGEX to build a
    # new local REGEX value here.
    local REGEX="${REGEX}.*\.(svgz|svg)"

    # Redefine list of files to process.
    cli_getFilesList $LOCATION

    # Process list of files.
    for FILE in $FILES;do
        cli_printMessages "$FILE" 'AsResponseLine'
        # Do something else here on...
    done

}
```

`cli_getLangCodes` [*FILTER*] [Function]

Outputs language codes supported by '`centos-art.sh`' script.

`cli_getLangCodes` function outputs a list of language codes as defined in ISO639 standard. When *FILTER* is provided, `cli_getLangCodes` outputs language codes that match *FILTER* regular expression pattern.

`cli_getLangName` [*FILTER*] [Function]

Outputs language names supported by '`centos-art.sh`' script.

`cli_getLangName` function reads one language locale code in the format LL_CC and outputs the language related name as in ISO639. If filter is specified, `cli_getLangName` returns the language name that matches the locale code specified in *FILTER*, exactly.

`cli_getLocales` [Function]

Output locale codes supported by '`centos-art.sh`' script.

Occasionally, you use `cli_getLocales` function to add locale information in non-common country-specific language ('`LL_CC`') format for those languages (e.g., '`bn_IN`', '`pt_BR`', etc.)

which locale differences cannot be solved using common language specifications (‘LL’) into one unique common locale specification (e.g., ‘bn’, ‘pt’, etc.).

`cli_getRepoName` *NAME TYPE* [Function]

Sanitate file names.

Inside ‘centos-art.sh’ script, specific functionalities rely both in `cli_getRepoName` and repository file system organization to achieve their goals. Consider `cli_getRepoName` function as central place to manage file name convenctions for other functions inside ‘centos-art.sh’ script.

> **Important** `cli_getRepoName` function doesn't verify file or directory existence, for that purpose use `cli_checkFiles` function instead.

The *NAME* variable contains the file name or directory name you want to sanitate.

The *TYPE* variable specifies what type of sanitation you want to perform on *NAME*. The *TYPE* can be one of the following values:

‘d’
‘directory’
: Sanitate directory *NAME*s.

‘f’
‘regular-file’
: Sanitate regular file *NAME*s.

Use `cli_getRepoName` function to sanitate file names and directory names before their utilization.

Use `cli_getRepoName` when you need to change file name convenctions inside ‘centos-art.sh’ script.

When we change file name convenctions inside `cli_getRepoName` what we are really changing is the way functions interpret repository file system organization. Notice that when we change a file name (e.g., a function name), it is necessary to update all files where such file name is placed on. This may require a massive substitution inside the repository, each time we change name convenctions in the repository (— **Removed**(pxref:trunk Scripts Bash Functions Path) —, for more information).

`cli_getRepoStatus` [*LOCATION*] [Function]

Request repository status.

This function requests the status of a *LOCATION* inside the working copy using the `svn status` command and returns the first character in the output line, just as described in `svn help status`. If *LOCATION* is not a regular file or a directory, inside the working copy, the ‘centos-art.sh’ script prints a message and ends its execution.

Use this function to perform verifications based a repository *LOCATION* status.

`cli_getTemporalFile` *NAME* [Function]

Output absolute path to temporal file *NAME*.

The `cli_getTemporalFile` function uses ‘/tmp’ directory as source location to store temporal files, the ‘centos-art.sh’ script name, and a random identification string to let you run more than one ‘centos-art.sh’ script simultaneously on the same user session. For example, due the following temporal file defintion:

```
cli_getTemporalFile $FILE
```

If *FILE* name is ‘instance.svg’ and the unique random string is ‘f16f7b51-ac12-4b7f-9e66-72df847f12de’, the final temporal file, built from previous temporal file definition, would be:

```
/tmp/centos-art.sh-f16f7b51-ac12-4b7f-9e66-72df847f12de-instance.svg
```

When you use the `cli_getTemporalFile` function to create temporal files, be sure to remove temporal files created once you've ended up with them. For example, consider the following construction:

```
for FILE in $FILES;do

    # Initialize temporal instance of file.
    INSTANCE=$(cli_getTemporalFile $FILE)

    # Do something ...

    # Remove temporal instance of file.
    if [[ -f $INSTANCE ]];then
        rm $INSTANCE
    fi

done
```

Use the `cli_getTemporalFile` function whenever you need to create temporal files inside '`centos-art.sh`' script.

`cli_getThemeName` [Function]

Output theme name.

In order for `cli_getThemeName` function to extract theme name correctly, the *ACTIONVAL* variable must contain a directory path under '`trunk/Identity/Themes/Motifs/`' directory structure. Otherwise, `cli_getThemeName` returns an empty string.

`cli_printMessage` *MESSAGE* [*FORMAT*] [Function]

Define standard output message definition supported by '`centos-art.sh`' script.

When *FORMAT* is not specified, `cli_printMessage` outputs information just as it was passed in *MESSAGE* variable. Otherwise, *FORMAT* can take one of the following values:

'`AsHeadingLine`'
: To print heading messages.

```
------------------------------------------------------------------------
$MESSAGE
------------------------------------------------------------------------
```

'`AsWarningLine`'
: To print warning messages.

```
------------------------------------------------------------------------
WARNING: $MESSAGE
------------------------------------------------------------------------
```

'`AsNoteLine`'
: To print note messages.

```
------------------------------------------------------------------------
NOTE: $MESSAGE
------------------------------------------------------------------------
```

'`AsUpdatingLine`'
: To print '`Updating`' messages on two-columns format.

```
Updating        $MESSAGE
```

‘AsRemovingLine’
To print ‘Removing’ messages on two-columns format.

```
Removing        $MESSAGE
```

‘AsCheckingLine’
To print ‘Checking’ messages on two-columns format.

```
Checking        $MESSAGE
```

‘AsCreatingLine’
To print ‘Creating’ messages on two-columns format.

```
Creating        $MESSAGE
```

‘AsSavedAsLine’
To print ‘Saved as’ messages on two-columns format.

```
Saved as        $MESSAGE
```

‘AsLinkToLine’
To print ‘Linked to’ messages on two-columns format.

```
Linked to       $MESSAGE
```

‘AsMovedToLine’
To print ‘Moved to’ messages on two-columns format.

```
Moved to        $MESSAGE
```

‘AsTranslationLine’
To print ‘Translation’ messages on two-columns format.

```
Translation     $MESSAGE
```

‘AsConfigurationLine’
To print ‘Configuration’ messages on two-columns format.

```
Configuration   $MESSAGE
```

‘AsResponseLine’
To print response messages on one-column format.

```
--> $MESSAGE
```

‘AsRequestLine’
To print request messages on one-column format. Request messages output messages with one colon (‘:’) and without trailing newline (‘\n’) at message end.

```
$MESSAGE:
```

‘AsYesOrNoRequestLine’
To print ‘yes or no’ request messages on one-column format. If something different from ‘y’ is answered (when using en_US.UTF-8 locale), script execution ends immediatly.

```
$MESSAGE [y/N]:
```

When we use ‘centos-art.sh’ script in a locale different from en_US.UTF-8, confirmation answer may be different from ‘y’. For example, if you use es_ES.UTF-8 locale, the confirmation question would look like:

```
$MESSAGE [s/N]:
```

and the confirmation answer would be ‘s’, as it is on Spanish ‘s’ word.

Definition of which confirmation word to use is set on translation messages for your specific locale information. — **Removed**(xref:trunk Scripts Bash Functions Locale) —, for more information about locale-specific translation messages.

'AsToKnowMoreLine'
: To standardize 'to know more, run the following command:' messages. When the 'AsToKnowMoreLine' option is used, the *MESSAGE* value should be set to "$(caller)". caller is a Bash builtin that returns the context of the current subroutine call. 'AsToKnowMoreLine' option uses caller builtin output to build documentation entries dynamically.

```
------------------------------------------------------------------------
To know more, run the following command:
centos-art manual --read='path/to/dir'
------------------------------------------------------------------------
```

Use 'AsToKnowMoreLine' option after errors and for intentional script termination.

'AsRegularLine'
: To standardize regular messages on one-column format.

When *MESSAGE* contains a colon inside (e.g., 'description: message'), the cli_printMessage function outputs *MESSAGE* on two-columns format.

Use cli_printMessage function whenever you need to output information from 'centos-art.sh' script.

Tip To improve two-columns format, change the following file:

trunk/Scripts/Bash/Styles/output_forTwoColumns.awk

2.32.3.3 Specific functions

The following specific functions of 'centos-art.sh' script, are available for you to use:

2.32.4 See also

2.33 The 'trunk/Scripts/Functions/Help' Directory

2.33.1 Goals

- ...

2.33.2 Description

- ...

2.33.3 Usage

- ...

2.33.4 See also

2.34 The 'trunk/Scripts/Functions/Locale' Directory

2.34.1 Goals

- ...

2.34.2 Description

This command looks for '.sh' files inside Bash directory and extracts translatable strings from files, using xgettext command, in order to create a portable object template ('centos-art.sh.pot') file for them.

With the '`centos-art.sh.pot`' file up to date, the `centos-art` command removes the temporal list of files sotred inside '`/tmp`' directory and checks the current language of your user's session to create a portable object file for it, in the location '`$CLI_LANG/$CLI_LANG.po`'.

The *CLI_LANG* variable discribes the locale language used to output messages inside `centos-art` command. The locale language used inside `centos-art` command is taken from the `LANG` environment variable. The *CLI_LANG* variable has the '`LL_CC`' format, where '`LL`' is a language code from the ISO-639 standard, and '`CC`' a country code from the ISO-3166 standard.

The `LANG` environment variable is set when you do log in to your system. If you are using a graphical session, change language to your native language and do login. That would set and exoprt the `LANG` environment variable to the correct value. On the other side, if you are using a text session edit your '`~/.bash_profile`' file to set and export the `LANG` environment variable to your native locale as defines the `locale -a` command output; do logout, and do login again.

At this point, the `LANG` environment variable has the appropriate value you need, in order to translate `centos-art.sh` messages to your native language (the one set in `LANG` environment variable).

With the '`$CLI_LANG/$CLI_LANG.po`' file up to date, the `centos-art` opens it for you to update translation strings. The `centos-art` command uses the value of *EDITOR* environment variable to determine your favorite text editor. If no value is defined on *EDITOR*, the '`/usr/bin/vim`' text editor is used as default.

When you finishd PO file edition and quit text editor, the `centos-art` command creates the related machine object in the location '`$CLI_LANG/LC_MESSAGES/$TEXTDOMAIN.mo`'.

At this point, all translations you made in the PO file should be available to your language when runing `centos-art.sh` script.

In order to make the `centos-art.sh` internationalization, the `centos-art.sh` script was modified as described in the `gettext` info documentation (`info gettext`). You can find such modifications in the following files:

- '`trunk/Scripts/Bash/initFunctions.sh`'
- '`trunk/Scripts/Bash/Functions/Help/cli_localeMessages.sh`'
- '`trunk/Scripts/Bash/Functions/Help/cli_localeMessagesStatus.sh`'
- ...

2.34.3 Usage

'`centos-art locale --edit`'
: Use this command to translate command-line interface output messages in the current system locale you are using (as specified in `LANG` environment variable).

'`centos-art locale --list`'
: Use this command to see the command-line interface locale report.

2.34.4 See also

2.35 The '`trunk/Scripts/Functions/Path`' Directory

2.35.1 Goals

This section exists to organize files related to `path` functiontionality. The `path` functionality standardizes movement, syncronization, branching, tagging, and general file maintainance inside the repository.

2.35.2 Description

"CentOS like trees, has roots, trunk, branches, leaves and flowers. Day by day they work together in freedom, ruled by the laws of nature and open standards, to show the beauty of its existence."

2.35.2.1 Repository layout

The repository layout describes organization of files and directories inside the repository. The repository layout provides the standard backend required for automation scripts to work correctly. If such layout changes unexpectedly, automation scripts may confuse themselves and stop doing what we expect from them to do.

As convenction, inside CentOS Artwork Repository, we organize files and directories related to CentOS corporate visual identity under three top level directories named: '`trunk/`', '`branches/`', and '`tags/`'.

The '`trunk/`' directory (see Section 2.3 [Directories trunk], page 12) organizes the main development line of CentOS corporate visual identity. Inside '`trunk/`' directory structure, the CentOS corporate visual identity concepts are implemented using directories. There is one directory level for each relevant concept inside the repository. The '`trunk/`' directory structure is mainly used to perform development tasks related to CentOS corporate visual identity.

The '`branches/`' directory () oranizes parallel development lines to '`trunk/`' directory. The '`branches/`' directory is used to set points in time where develpment lines are devided one from another taking separte and idependent lives that share a common past from the point they were devided on. The '`branches/`' directory is mainly used to perform quality assurance tasks related to CentOS corporate visual identity.

The '`tags/`' directory (see Section 2.2 [Directories tags], page 11) organizes parallel frozen lines to '`branches/`' directory. The parallel frozen lines are immutable, nothing change inside them once they has been created. The '`tags/`' directory is mainly used to publish final releases of CentOS corporate visual identity.

The CentOS Artwork Repository layout is firmly grounded on a Subversion base. Subversion (`http://subversion.tigris.org`) is a version control system, which allows you to keep old versions of files and directories (usually source code), keep a log of who, when, and why changes occurred, etc., like CVS, RCS or SCCS. Subversion keeps a single copy of the master sources. This copy is called the source "repository"; it contains all the information to permit extracting previous versions of those files at any time.

2.35.2.2 Repository name convenctions

Repository name convenctions help us to maintain consistency of names inside the repository.

Repository name convenctions are applied to files and directories inside the repository layout. As convenction, inside the repository layout, file names are all written in lowercase ('`01-welcome.png`', '`splash.png`', '`anaconda_header.png`', etc.) and directory names are all written capitalized (e.g., '`Identity`', '`Themes`', '`Motifs`', '`TreeFlower`', etc.).

Repository name convenctions are implemented inside the `cli_getRepoName` function of '`centos-art.sh`' script. With `cli_getRepoName` function we reduce the amount of commands and convenctions to remember, concentrating them in just one single place to look for fixes and improvements.

2.35.2.3 Repository work flow

Repository work flow describes the steps and time intervals used to produce CentOS corporate visual identity inside CentOS Artwork Repository.

To illustrate repository work flow let's consider themes' development cycle.

Initially, we start working themes on their trunk development line (e.g., '`trunk/Identity/Themes/Motifs/TreeFlower/`'), here we organize information that

cannot be produced automatically (i.e., background images, concepts, color information, screenshots, etc.).

Later, when theme trunk development line is considered "ready" for implementation (e.g., all required backgrounds have been designed), we create a branch for it (e.g., '`branches/Identity/Themes/Motifs/TreeFlower/1/`'). Once the branch has been created, we forget that branch and continue working the trunk development line while others (e.g., an artwork quality assurance team) test the new branch for tunning it up.

Once the branch has been tunned up, and considered "ready" for release, it is freezed under '`tags/`' directory (e.g., '`tags/Identity/Themes/Motifs/TreeFower/1.0/`') for packagers, webmasters, promoters, and anyone who needs images from that CentOS theme the tag was created for.

Both branches and tags, inside CentOS Artwork Repository, use numerical values to identify themselves under the same location. Branches start at one (i.e., '`1`') and increment one unit for each branch created from the same trunk development line. Tags start at zero (i.e., '`0`') and increment one unit for each tag created from the same branch development line.

> **Convenction** Do not freeze trunk development lines using tags directly. If you think you need to freeze a trunk development line, create a branch for it and then freeze that branch instead.

The trunk development line may introduce problems we cannot see immediatly. Certainly, the high changable nature of trunk development line complicates finding and fixing such problems. On the other hand, the branched development lines provide a more predictable area where only fixes/corrections to current content are commited up to repository.

If others find and fix bugs inside the branched development line, we could merge such changes/experiences back to trunk development line (not visversa) in order for future branches, created from trunk, to benefit.

Time intervals used to create branches and tags may vary, just as different needs may arrive. For example, consider the release schema of CentOS distribution: one major release every 2 years, security updates every 6 months, support for 7 years long. Each time a CentOS distribution is released, specially if it is a major release, there is a theme need in order to cover CentOS distribution artwork requirements. At this point, is where CentOS Artwork Repository comes up to scene.

Before releasing a new major release of CentOS distribution we create a branch for one of several theme development lines available inside the CentOS Artwork Repository, perform quality assurance on it, and later, freeze that branch using tags. Once a the theme branch has been frozen (under '`tags/`' directory), CentOS Packagers (the persons whom build CentOS distribution) can use that frozen branch as source location to fulfill CentOS distribution artwork needs. The same applies to CentOS Webmasters (the persons whom build CentOS websites), and any other visual manifestation required by the project.

2.35.2.4 Parallel directories

Inside CentOS Artwork Repository, parallel directories are simple directory entries built from a common parent directory and placed in a location different to that, the common parent directory is placed on. Parallel directories are useful to create branches, tags, translations, documentation, pre-rendering configuration script, and similar directory structures.

Parallel directories take their structure from one unique parent directory. Inside CentOS Artwork Repository, this unique parent directory is under '`trunk/Identity`' location. The '`trunk/Identity`' location must be considered the reference for whatever information you plan to create inside the repository.

In some circumstances, parallel directories may be created removing uncommon information from their paths. Uncommon path information refers to those directory levels

in the path which are not common for other parallel directories. For example, when rendering '`trunk/Identity/Themes/Motifs/TreeFlower/Distro`' directory structure, the '`centos-art.sh`' script removes the '`Motifs/TreeFlower/`' directory levels from path, in order to build the parallel directory used to retrived translations, and pre-rendering configuration scripts required by `render` functionality.

Another example of parallel directory is the documentation structure created by `manual` functionality. This time, '`centos-art.sh`' script uses parallel directory information with uncommon directory levels to build the documentation entry required by Texinfo documentation system, inside the repository.

Othertimes, parallel directories may add uncommon information to their paths. This is the case we use to create branches and tags. When we create branches and tags, a numerical identifier is added to parallel directory structure path. The place where the numerical identifier is set on is relevant to corporate visual identity structure and should be carefully considered where it will be.

When one parent directory changes, all their related parallel directories need to be changed too. This is required in order for parallel directories to retain their relation with the parent directory structure. In the other hand, parallel directories should never be modified under no reason but to satisfy the relation to their parent directory structure. Liberal change of parallel directories may suppresses the conceptual idea they were initially created for; and certainly, things may stop working the way they should do.

2.35.2.5 Syncronizing path information

Parallel directories are very useful to keep repository organized but introduce some complications. For instance, consider what would happen to functionalities like `manual` ('`trunk Scripts Bash Functions Manual`') that rely on parent directory structures to create documentation entries (using parallel directory structures) if one of those parent directory structures suddenly changes after the documentation entry has been already created for it?

In such cases, functionalities like `manual` may confuse themselves if path information is not updated to reflect the relation with its parent directory. Such functionalities work with parent directory structure as reference; if a parent directory changes, the functionalities dont't even note it because they work with the last parent directory structure available in the repository, no matter what it is.

In the specific case of documentation (the `manual` functionality), the problem mentioned above provokes that older parent directories, already documented, remain inside documentation directory structures as long as you get your hands into the documentation directory structure ('`trunk/Manuals`') and change what must be changed to match the new parent directory structure.

There is no immediate way for `manual`, and similar functionalities that use parent directories as reference, to know when and how directory movements take place inside the repository. Such information is available only when the file movement itself takes place inside the repository. So, is there, at the moment of moving files, when we need to syncronize parallel directories with their unique parent directory structure.

> **Warning** There is not support for URL reference inside '`centos-art.sh`' script. The '`centos-art.sh`' script is designed to work with local files inside the working copy only.

As CentOS Artwork Repository is built over a version control system, file movements inside the repository are considered repository changes. In order for these repository changes to be versioned, we need to, firstly, add changes into the version control system, commit them, and later, perform movement actions using version control system commands. This configuration

makes possible for everyone to know about changes details inside the repository; and if needed, revert or update them back to a previous revision.

Finally, once all path information has been corrected, it is time to take care of information inside the files. For instance, considere what would happen if you make a reference to a documentation node, and later the documentation node you refere to is deleted. That would make Texinfo to produce error messages at export time. So, the '`centos-art.sh`' script needs to know when such changes happen, in a way they could be noted and handled without producing errors.

2.35.2.6 What is the right place to store it?

Occasionly, you may find that new corporate visual identity components need to be added to the repository. If that is your case, the first question you need to ask yourself, before start to create directories blindly all over, is: What is the right place to store it?

The CentOS Community different free support vains (see: http://wiki.centos.org/GettingHelp) are the best place to find answers to your question, but going there with hands empty is not good idea. It may give the impression you don't really care about. Instead, consider the following suggestions to find your own comprehension and so, make your propositions based on it.

When we are looking for the correct place to store new files, to bear in mind the corporate visual identity structure used inside the CentOS Artwork Repository (see Section 2.4 [Directories trunk Identity], page 12) would be probaly the best advice we could offer, the rest is just matter of choosing appropriate names. To illustrate this desition process let's consider the '`trunk/Identity/Themes/Motifs/TreeFlower`' directory as example. It is the trunk development line of *TreeFlower* artistic motif. Artistic motifs are considered part of themes, which in turn are considered part of CentOS corporate visual identity.

When building parent directory structures, you may find that reaching an acceptable location may take some time, and as it uses to happen most of time; once you've find it, that may be not a definite solution. There are many concepts that you need to play with, in order to find a result that match the conceptual idea you try to implement in the new directory location. To know which these concepts are, split the location in words and read its documentation entry from less specific to more specific.

For example, the '`trunk/Identity/Themes/Motifs/TreeFlower`' location evolved through several months of contant work and there is no certain it won't change in the future, even it fixes quite well the concept we are trying to implement. The concepts used in '`trunk/Identity/Themes/Distro/Motifs/TreeFlower`' location are described in the following commands, respectively:

```
centos-art manual --read=turnk/
centos-art manual --read=turnk/Identity/
centos-art manual --read=turnk/Identity/Themes/
centos-art manual --read=turnk/Identity/Themes/Motifs/
centos-art manual --read=turnk/Identity/Themes/Motifs/TreeFlower/
```

Other location concepts can be found similary as we did above, just change the location we used above by the one you are trying to know concepts for.

2.35.3 Usage

`centos-art path --copy='SRC' --to='DST'`
: Copy '`SRC`' to '`DST`' and schedule '`DST`' for addition (with history). In this command, '`SRC`' and '`DST`' are both working copy (WC) entries.

`centos-art path --delete='SRC'`
: Delete '`DST`'. In order for this command to work the file or directory you intend to delete should be under version control first. In this command, '`SRC`' is a working copy (WC) entry.

2.35.4 See also

2.36 The '`trunk/Scripts/Functions/Prepare`' Directory

2.36.1 Goals

This section exists to organize files related to '`centos-art.sh`' script '`prepare`' functionality. The '`prepare`' functionality of '`centos-art.sh`' script helps you to prepare the workstation configuration you are planning to use as host for your working copy of CentOS Artwork Repository.

2.36.2 Description

The first time you download CentOS Artwork Repository you need to configure your workstation in order to use '`centos-art.sh`' script. These preliminar configurations are based mainly on auxiliar RPM packages installation, symbolic links creations, and environment variables definitions. The '`prepare`' functionality of '`centos-art.sh`' script guides you through this preliminar configuration process.

If this is the first time you run '`centos-art.sh`' script, the appropriate way to use its '`prepare`' functionality is not using the '`centos-art.sh`' script directly, but the absolute path to `centos-art.sh` script instead (i.e., '`~/artwork/trunk/Scripts/Bash/centos-art.sh`'). This is necessary because '`centos-art`' symbolic link, under '`~/bin/`' directory, has not been created yet.

2.36.2.1 Packages

Installation of auxiliar RPM packages provides the software required to manipulate files inside the repository (e.g., image files, documentation files, translation files, script files, etc.). Most of RPM packages `centos-art.sh` script uses are shipped with CentOS distribution, and can be installed from CentOS base repository. The only exception is '`inkscape`', the package we use to manipulate SVG files. The '`inkscape`' package is not inside CentOS distribution so it needs to be installed from third party repositories.

> **Note** Configuration of third party repositories inside CentOS distribution is described in CentOS wiki, specifically in the following URL: http://wiki.centos.org/AdditionalResources/Repositories

Before installing packages, the '`centos-art.sh`' script uses `sudo` to request root privileges to execute `yum` installation functionality. If your user isn't defined as a privileged user—at least to run `yum` commands— inside '`/etc/sudoers`' configuration file, you will not be able to perform package installation tasks as set in '`centos-art.sh`' script '`prepare`' functionality.

Setting sudo privileges to users is an administrative task you have to do by yourself. If you don't have experience with `sudo` command, please read its man page running the command: `man sudo`. This reading will be very useful, and with some practice, you will be able to configure your users to have `sudo` privileges.

2.36.2.2 Links

Creation of symbolic links helps us to alternate between different implementations of '`centos-art.sh`' script-line (e.g., '`centos-art.sh`', for Bash implementation; '`centos-art.py`', for Python implementation; '`centos-art.pl`', for Perl implementation; and so on for other implementations). The '`centos-art.sh`' script-line definition takes place inside

your personal binary (‘~/bin/’) directory in order to make the script implementation —the one that ‘centos-art’ links to— available to *PATH* environment variable.

Creation of symbolic links helps us to reuse components from repository working copy. For example, color information files maintained inside your working copy must never be duplicated inside program-specific configuration directories that uses them in your workstation (e.g., Gimp, Inkscape, etc.). Instead, a symbolic link must be created for each one of them, from program-specific configuration directories to files in the working copy. In this configuration, when someone commits changes to color information files up to central repository, they—the changes committed— will be immediatly available to your programs the next time you update your working copy —the place inside your workstation those color information files are stored—.

Creation of symbolic links helps us to make ‘centos-art.sh’ script functionalities available outside ‘trunk/’ repository directory structure, but at its same level in repository tree. This is useful if you need to use the “render” functionality of centos-art.sh under ‘branches/’ repository directory structure as you usually do inside ‘trunk/’ repository directory structure. As consequence of this configuration, automation scripts cannot be branched under ‘branches/Scripts’ directory structure.

2.36.2.3 Environment variables

Definition of environemnt variables helps us to set default values to our user session life. The user session environment variable defintion takes place in the user’s ‘~/.bash_profile’ file. The ‘prepare’ functionality of ‘centos-art.sh’ script doesn’t modify your ‘~/.bash_profile’ file.

The ‘prepare’ functionality of ‘centos-art.sh’ script evaluates the following environment variables:

EDITOR Default text editor.

The ‘centos-art.sh’ script uses default text EDITOR to edit pre-commit subversion messages, translation files, configuration files, script files, and similar text-based files.

If EDITOR environment variable is not set, ‘centos-art.sh’ script uses ‘/usr/bin/vim’ as default text editor. Otherwise, the following values are recognized by ‘centos-art.sh’ script:

- ‘/usr/bin/vim’
- ‘/usr/bin/emacs’
- ‘/usr/bin/nano’

If no one of these values is set in EDITOR environment variable, ‘centos-art.sh’ uses ‘/usr/bin/vim’ text editor by default.

TEXTDOMAIN

Default domain used to retrieve translated messages. This variable is set in ‘initFunctions.sh’ and shouldn’t be changed.

TEXTDOMAINDIR

Default directory used to retrieve translated messages. This variable is set in ‘initFunctions.sh’ and shouldn’t be changed.

LANG

Default locale information.

This variable is initially set in the configuration process of CentOS distribution installer (i.e., Anaconda), specifically in the ‘Language’ step; or once installed using the system-config-language tool.

The ‘centos-art.sh’ script uses the *LANG* environment variable to know in which language the script messages are printed out.

`TZ`

Default time zone representation.

This variable is initially set in the configuration process of CentOS distribution installer (i.e., Anaconda), specifically in the '`Date and time`' step; or once installed using the `system-config-date` tool.

The '`centos-art.sh`' script doesn't use the *TZ* environment variable information at all. Instead, this variable is used by the system shell to show the time information according to your phisical location on planet Earth.

Inside your computer, the time information is firstly set in the BIOS clock (which may need correction), and later in the configuration process of CentOS distribution installer (or later, by any of the related configuration tools inside CentOS distribution). Generally, setting time information is a straight-forward task and configuration tools available do cover most relevant location. However, if you need a time precision not provided by the configuration tools available inside CentOS distribution then, using *TZ* variable may be necessary.

> **Convenction** In order to keep changes syncronized between central repository and its working copies: configure both repository server and workstations (i.e., the place where each working copy is set on) to use Coordinated Universal Time (UTC) as base time representation. Later, correct the time information for your specific location using time zone correction.

The format of *TZ* environment variable is described in '`tzset(3)`' manual page.

2.36.2.4 Shell Script Files

The `shell` functionality of '`centos-art.sh`' script helps you to maintain bash scripts inside repository. For example, suppose you've created many functionalities for '`centos-art.sh`' script, and you want to use a common copyright and license note for consistency in all your script files. If you have a bunch of files, doing this one by one wouldn't be a big deal. In contrast, if the amount of files grows, updating the copyright and license note for all of them would be a task rather tedious. The `shell` functionality exists to solve maintainance tasks just as the one previously mentioned.

When you use `shell` functionality to update copyright inside script files, it is required that your script files contain (at least) the following top commentary structure:

```
 1| #!/bin/bash
 2| #
 3| # doSomething.sh -- The function description goes here.
 4| #
 5| # Copyright
 6| #
 7| # ...
 8| #
 9| # ------------------------------------------------------------------------
10| # $Id$
11| # ------------------------------------------------------------------------
12|
13| function doSomething {
14|
15| }
```

Relevant lines in the above structure are lines from 5 to 9. Everything else in the file is left immutable.

When you are updating copyright through `shell` functionality, the '`centos-art.sh`' script replaces everything in-between line 5 —the first one matching '`^# Copyright .+$`' string— and line 9—the first long dash separator matching '`^# -+$`'— with the content of copyright template instance.

> **Caution** Be sure to add the long dash separator that matches '`^# -+$`' regular expression *before* the function definition. Otherwise, if the '`Copyright`' line is present but no long dash separator exists, '`centos-art.sh`' will remove anything in-between the '`Copyright`' line and the end of file. This way you may lost your function definitions entirely.

The copyright template instance is created from one copyright template stored in the '`Config/tpl_forCopyright.sed`' file. The template instance is created once, and later removed when no longer needed. At this moment, when template instance is created, the '`centos-art.sh`' script takes advantage of automation in order to set copyright full name and date dynamically.

When you use `shell` functionality to update copyright, the first thing '`shell`' functionality does is requesting copyright information to user, and later, if values were left empty (i.e., no value was typed before pressing ⟨RET⟩ key), the '`shell`' functionality uses its own default values.

When `shell` functionality uses its own default values, the final copyright note looks like the following:

```
 1| #!/bin/bash
 2| #
 3| # doSomthing.sh -- The function description goes here.
 4| #
 5| # Copyright (C) 2003, 2010 The CentOS Project
 6| #
 7| # This program is free software; you can redistribute it and/or modify
 8| # it under the terms of the GNU General Public License as published by
 9| # the Free Software Foundation; either version 2 of the License, or
10| # (at your option) any later version.
11| #
12| # This program is distributed in the hope that it will be useful, but
13| # WITHOUT ANY WARRANTY; without even the implied warranty of
14| # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
15| # General Public License for more details.
16| #
17| # You should have received a copy of the GNU General Public License
18| # along with this program; if not, write to the Free Software
19| # Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
20| # USA.
21| #
22| # ------------------------------------------------------------------------
23| # $Id$
24| # ------------------------------------------------------------------------
25|
26| function doSomething {
27|
28| }
```

Relevant lines in the above structure are lines from 5 to 22. Pay attention how the copyright line was built, and how the license was added into the top comment where previously was just three dots. Everything else in the file was left immutable.

To change copyright information (i.e., full name or year information), run the `shell` functionality over the root directory containing the script files you want to update copyright in and enter the appropriate information when it be requested. You can run the `shell` functionality as many times as you need to.

To change copyright license (i.e., the text in-between lines 7 and 20), you need to edit the '`Config/tpl_forCopyright.sed`' file, set the appropriate information, and run the `shell` functionality once again for changes to take effect over the files you specify.

Important The '`centos-art.sh`' script is released as:

```
GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
```

Do not change the license information under which '`centos-art.sh`' script is released. Instead, if you think a different license must be used, please share your reasons at CentOS Developers mailing list.

See file trunk/Scripts/COPYING, for a complete license description.

2.36.2.5 SVG Files

The `svg` functionality of '`centos-art.sh`' script helps you to maintain scalable vector graphics (SVG) inside repository. For example, suppose you've been working in CentOS default design models under '`trunk/Identity/Themes/Models/`', and you want to set common metadata to all of them, and later remove all unused SVG defintions from '`*.svg`' files. Doing so file by file may be a tedious task, so the '`centos-art.sh`' script provides the `svg` functionality to aid you maintain such actions.

The metadata used is defined by Inkscape 0.46 using the SVG standard markup. The '`centos-art.sh`' script replaces everything in-between `<metadata` and `</metadata>` tags with a predefined metadata template we've set for this purpose.

The metadata template was created using the metadata information of a file which, using Inkscape 0.46, all metadata fields were set. This created a complete markup representation of how SVG metadata would look like. Later, we replaced every single static value with a translation marker in the form '`=SOMETEXT=`', where `SOMETEXT` is the name of its main opening tag. Later, we transform the metadata template into a sed replacement set of commads escaping new lines at the end of each line.

With metadata template in place, the '`centos-art.sh`' script uses it to create a metadata template instance for the file being processed currently. The metadata template instance contains the metadata portion of sed replacement commands with translation markers already traduced. In this action, instance creation, is where we take advantage of automation and generate metadata values like title, date, keywords, source, identifier, and relation dynamically, based on the file path '`centos-art.sh`' script is currently creating metadata information for.

With metadata template instance in place, the '`centos-art.sh`' script uses it to replace real values inside all '`.svg`' files under the current location you're running the '`centos-art.sh`' script on. Default behaviour is to ask user to enter each metadatum required, one by one. If user leaves metadatum empty, by pressing ⟨RET⟩ key, '`centos-art.sh`' uses its default value.

Many of the no-longer-used gradients, patterns, and markers (more precisely, those which you edited manually) remain in the corresponding palettes and can be reused for new objects. However if you want to optimize your document, use the '`Vacuum Defs`' command in '`File`' menu. It will remove any gradients, patterns, or markers which are not used by anything in the document, making the file smaller.

If you have one or two couple of files, removing unused definitions using the graphical interface may be enough to you. In contrast, if you have dozens or even houndreds of scalable vector graphics files to maintain it is not a fun task to use the graphical interface to remove unused definitions editing those files one by one.

To remove unused definitions from several scalable vector graphics files, the '`centos-art.sh`' script uses Inkscape command-line interface, specifically with the '`--vaccum-defs`' option.

2.36.2.6 XHTML Files

2.36.3 Usage

`centos-art prepare --packages`
: Verify required packages your workstation needs in order to run the '`centos-art.sh`' script correctly. If there are missing packages, the '`centos-art.sh`' script asks you to confirm their installation. When installing packages, the '`centos-art.sh`' script uses the `yum` application in order to achieve the task.

  In case all packages required by '`centos-art.sh`' script are already installed in your workstation, the message '`The required packages are already installed.`' is output for you to know.

`centos-art prepare --links`
: Verify required links your workstation needs in order to run the centos-art command correctly. If any required link is missing, the `centos-art.sh` script asks you to confirm their installation. To install required links, the `centos-art.sh` script uses the `ln` command.

  In case all links required by '`centos-art.sh`' script are already created in your workstation, the message '`The required links are already installed.`' is output for you to know.

  In case a regular file exists with the same name of a required link, the '`centos-art.sh`' script outputs the '`Already exists as regular file.`' message when listing required links that will be installed. Of course, as there is already a regular file where must be a link, no link is created. In such cases the '`centos-art.sh`' script will fall into a continue installation request for that missing link. To end this continue request you can answer '`No`', or remove the existent regular file to let '`centos-art.sh`' script install the link on its place.

`centos-art prepare --environment`
`centos-art prepare --environment --filter='regex'`
: Output a brief description of environment variables used by '`centos-art.sh`' script.

  If '`--filter`' option is provided, output is reduced as defined in the '`regex`' regular expression value. If '`--filter`' option is specified but '`regex`' value is not, the '`centos-art.sh`' script outputs information as if '`--filter`' option had not been provided at all.

2.36.4 See also

2.37 The '`trunk/Scripts/Functions/Render`' Directory

The `render` functionality exists to produce both identity and translation files on different levels of information (i.e., different languages, release numbers, architectures, etc.).

The `render` functionality relies on "renderable directory structures" to produce files. Renderable directory structures can be either "identity directory structures" or "translation directory structures" with special directories inside.

2.37.1 Renderable identity directory structures

Renderable identity directory structures are the starting point of identity rendition. Whenever we want to render a component of CentOS corporate visual identity, we need to point '`centos-art.sh`' to a renderable identity directory structure. If such renderable identity directory structure doesn't exist, then it is good time to create it.

Inside the working copy, one renderable identity directory structures represents one visual manifestation of CentOS corporate visual identity, or said differently, each visual manifestation of CentOS corporate visual identity should have one renderable identity directory structure.

Inside renderable identity directory structures, '`centos-art.sh`' can render both image-based and text-based files. Specification of whether a renderable identity directory structure produces image-based or text-based content is a configuration action that takes place in the pre-rendition configuration script of that renderable identity directory structure.

Inside renderable identity directory structures, content production is organized in different configurations. A content production configuration is a unique combination of the components that make an identity directory structure renderable. One content production configuration does one thing only (e.g., to produce untranslated images), but it can be extended (e.g., adding translation files) to achieve different needs (e.g., to produce translated images).

2.37.1.1 Design template without translation

The design template without translation configuration is based on a renderable identity directory structure with an empty translation directory structure. In this configuration, one design template produces one untranslated file. Both design templates and final untranslated files share the same file name, but they differ one another in file-type and file-extension.

For example, to produce images without translations (there is no much use in producing text-based files without translations), consider the following configuration:

One renderable identity directory structure:

In this example we used '`Identity/Path/To/Dir`' as the identity component we want to produce untranslated images for. Identity components can be either under '`trunk/`' or '`branches/`' directory structure.

The identity component (i.e., '`Identity/Path/To/Dir`', in this case) is also the bond component we use to connect the identity directory structures with their respective auxiliar directories (i.e., translation directory structres and pre-rendition configuration structures). The bond component is the path convenction that '`centos-art.sh`' uses to know where to look for related translations, configuration scripts and whatever auxiliar thing a renderable directory structure may need to have.

```
        | The bond component
        |----------------->|
trunk/Identity/Path/To/Dir  <-- Renderable identity directory structure.
|-- Tpl                     <-- Design template directory.
|   `-- file.svg            <-- Design template file.
`-- Img                     <-- Directory used to store final files.
    `-- file.png            <-- Final image-based file produced from
                                design template file.
```

Inside design template directory, design template files are based on SVG (Scalable Vector Graphics) and use the extension `.svg`. Design template files can be organized using several directory levels to create a simple but extensible configuration, specially if translated images are not required.

In order for SVG (Scalable Vector Graphics) files to be considered "design template" files, they should be placed under the design template directory and to have set a `CENTOSARTWORK` object id inside.

The `CENTOSARTWORK` word itself is a convenction name we use to define which object/design area, inside a design template, the '`centos-art.sh`' script will use to export as PNG (Portable Network Graphic) image at rendition time. Whithout such object id specification, the '`centos-art.sh`' script cannot know what object/design area you (as designer) want to export as PNG (Portable Network Graphic) image file.

> **Note** At rendition time, the content of '`Img/`' directory structure is produced by '`centos-art.sh`' automatically.

When a renderable identity directory structure is configured to produce image-based content, '`centos-art.sh`' produces PNG (Portable Network Graphics) files with the `.png` extension. Once the base image format has been produced, it is possible for '`centos-art.sh`' to use it in order to automatically create other image formats that may be needed (— **Removed**(pxref:trunk Scripts Bash Functions Render Config) —).

Inside the working copy, you can find an example of "design template without translation" configuration at '`trunk/Identity/Models/`'.

See Section 2.4 [Directories trunk Identity], page 12, for more information.

One translation directory structure:

In order for an identity entry to be considered an identity renderable directory structure, it should have a translation entry. The content of the translation entry is relevant to determine how to process the identity renderable directory entry.

If the translation entry is empty (i.e., there is no file inside it), '`centos-art.sh`' interprets the identity renderable directory structure as a "design templates without translation" configuration.

```
                      | The bond component
                      |----------------->|
trunk/Translations/Identity/Path/To/Dir
'-- (empty)
```

If the translation entry is not empty, '`centos-art.sh`' can interpret the identity renderable directory structure as one of the following configurations: "design template with translation (one-to-one)" or "design template with translation (optimized)". Which one of these configurations is used depends on the value assigned to the matching list (*MATCHINGLIST*) variable in the pre-rendition configuration script of the renderable identity directory structure we are producing images for.

If the matching list variable is empty (as it is by default), then "design template with translation (one-to-one)" configuration is used. In this configuration it is required that both design templates and translation files have the same file names. This way, *one* translation files is applied to *one* design template, to produce *one* translated image.

If the matching list variable is not empty (because you redefine it in the pre-rendition configuration script), then "design template with translation (optimized)" configuration is used instead. In this configuration, design templates and translation files don't need to have the same names since such name relationship between them is specified in the matching list properly.

— **Removed**(xref:trunk Translations) —, for more information.

One pre-rendition configuration script:

In order to make an identity directory structure renderable, a pre-rendition configuration script should exist for it. The pre-rendition configuration script specifies

what type of rendition does ‘centos-art.sh’ will perform over the identity directory structure and how does it do that.

```
                                                | The bond component
                                                |----------------->|
trunk/Scripts/Bash/Functions/Render/Config/Identity/Path/To/Dir
‘-- render.conf.sh
```

In this configuration the pre-rendition configuration script (‘render.conf.sh’) would look like the following:

```
function render_loadConfig {

    # Define rendition actions.
    ACTIONS[0]='BASE:renderImage'

}
```

Since translation directory structure is empty, ‘centos-art.sh’ assumes a “design template without translation” configuration to produce untranslated images.

To produce untranslated images, ‘centos-art.sh’ takes one design template and creates one temporal instance from it. Later, ‘centos-art.sh’ uses the temporal design template instance as source file to export the final untranslated image. The action of exporting images from SVG (Scalable Vector Graphics) to PNG (Portable Network Graphics) is possible thanks to Inkscape’s command-line interface and the CENTOSARTWORK object id we previously set inside design templates.

```
centos-art.sh render --identity=trunk/Identity/Path/To/Dir
-------------------------------------------------
0 | Execute centos-art.sh on renderable identity directory structure.
--v----------------------------------------------
trunk/Identity/Path/To/Dir/Tpl/file.svg
-------------------------------------------------
1 | Create instance from design template.
--v----------------------------------------------
/tmp/centos-art.sh-a07e824a-5953-4c21-90ae-f5e8e9781f5f-file.svg
-------------------------------------------------
2 | Render untranslated image from design template instance.
--v----------------------------------------------
trunk/Identity/NewDir/Img/file.png
-------------------------------------------------
3 | Remove design template instance.
```

Finally, when the untranslated image has been created, the temporal design template instance is removed. At this point, ‘centos-art.sh’ takes the next design template and repeats the whole production flow once again (design template by design template), until all design templates be processed.

— **Removed**(xref:trunk Scripts Bash Functions Render Config) —, for more information.

2.37.1.2 Design template with translation (one-to-one)

Producing untranslated images is fine in many cases, but not always. Sometimes it is required to produce images in different languages and that is something that untrasnlated image production cannot achieve. However, if we fill its empty translation entry with translation files (one for each design template) we extend the production flow from untranslated image production to translated image production.

In order for '`centos-art.sh`' to produce images correctly, each design template should have one translation file and each translation file should have one design template. Otherwise, if there is a missing design template or a missing translation file, '`centos-art.sh`' will not produce the final image related to the missing component.

In order for '`centos-art.sh`' to know which is the relation between translation files and design templates the translation directory structure is taken as reference. For example, the '`trunk/Translations/Identity/Path/To/Dir/file.sed`' translation file does match '`trunk/Identity/Path/To/Dir/Tpl/file.svg`' design template, but it doesn't match '`trunk/Identity/Path/To/Dir/File.svg`' or '`trunk/Identity/Path/To/Dir/Tpl/File.svg`' or '`trunk/Identity/Path/To/Dir/Tpl/SubDir/file.svg`' design templates.

The pre-rendition configuration script used to produce untranslated images is the same we use to produce translated images. There is no need to modify it. So, as we are using the same pre-rendition configuration script, we can say that translated image production is somehow an extended/improved version of untranslated image production.

> **Note** If we use no translation file in the translation entry (i.e., an empty directory), '`centos-art.sh`' assumes the untranslated image production. If we fill the translation entry with translation files, '`centos-art.sh`' assumes the translated image production.

To produce final images, '`centos-art.sh`' applies one translation file to one design template and produce a translated design template instance. Later, '`centos-art.sh`' uses the translated template instance to produce the translated image. Finally, when the translated image has been produced, '`centos-art.sh`' removes the translated design template instance. This production flow is repeated for each translation file available in the translatio entry.

```
centos-art.sh render --identity=trunk/Identity/Path/To/Dir
------------------------------------------------
0 | Execute centos-art.sh on directory structure.
--v---------------------------------------------
trunk/Translations/Identity/Path/To/Dir/file.sed
------------------------------------------------
1 | Apply translation to design template.
--v---------------------------------------------
trunk/Identity/Path/To/Dir/Tpl/file.svg
------------------------------------------------
2 | Create design template instance.
--v---------------------------------------------
/tmp/centos-art.sh-a07e824a-5953-4c21-90ae-f5e8e9781f5f-file.svg
------------------------------------------------
3 | Render PNG image from template instance.
--v---------------------------------------------
trunk/Identity/NewDir/Img/file.png
------------------------------------------------
4 | Remove design template instance.
```

2.37.1.3 Design template with translation (optimized)

Producing translated images satisfies almost all our production images needs, but there is still a pitfall in them. In order to produce translated images as in the "one-to-one" configuration describes previously, it is required that one translation file has one design template. That's useful in many cases, but what would happen if we need to apply many different translation files to the same design template? Should we have to duplicate the same design template file for each translation file, in order to satisfy the "one-to-one" relation? What if we need to assign translation files to design templates arbitrarily?

Certenly, that's something the "one-to-one" configuration cannot handle. So, that's why we had to "optimize" it. The optimized configuration consists on using a matching list (*MATCHINGLIST*) variable that specifies the relationship between translation files and design templates in an arbitrary way. Using such matching list between translation files and design templates let us use as many assignment combinations as translation files and design templates we are working with.

The *MATCHINGLIST* variable is set in the pre-rendition configuration script of the component we want to produce images for. By default, the *MATCHINGLIST* variable is empty which means no matching list is used. Otherwise, if *MATCHINGLIST* variable has a value different to empty value then, '`centos-art.sh`' interprets the matching list in order to know how translation files are applied to design templates.

For example, consider the following configuration:

One entry under '`trunk/Identity/`':

In this configuration we want to produce three images using a paragraph-based style, controlled by '`paragraph.svg`' design template; and one image using a list-based style, controlled by '`list.svg`' design template.

```
trunk/Identity/Path/To/Dir
|-- Tpl
|   |-- paragraph.svg
|   `-- list.svg
`-- Img
    |-- 01-welcome.png
    |-- 02-donate.png
    |-- 03-docs.png
    `-- 04-support.png
```

One entry under '`trunk/Translations/`':

In order to produce translated images we need to have one translation file for each translated image we want to produce. Notice how translation names do match final image file names, but how translation names do not match design template names. When we use matching list there is no need for translation files to match the names of design templates, such name relation is set inside the matching list itself.

```
trunk/Translations/Identity/Path/To/Dir
|-- 01-welcome.sed
|-- 02-donate.sed
|-- 03-docs.sed
`-- 04-support.sed
```

One entry under '`trunk/trunk/Scripts/Bash/Functions/Render/Config/`':

In order to produce different translated images using specific design templates, we need to specify the relation between translation files and design templates in a way that '`centos-art.sh`' could know exactly what translation file to apply to what design template. This relation between translation files and design templates is set using the matching list *MATCHINGLIST* variable inside the pre-rendition configuration script of the component we want to produce images for.

```
trunk/Scripts/Bash/Functions/Render/Config/Identity/Path/To/Dir
`-- render.conf.sh
```

In this configuration the pre-rendition configuration script ('`render.conf.sh`') would look like the following:

```
function render_loadConfig {
```

```
        # Define rendition actions.
        ACTIONS[0]='BASE:renderImage'

        # Define matching list.
        MATCHINGLIST="\
        paragraph.svg:\
            01-welcome.sed\
            02-donate.sed\
            04-support.sed
        list.svg:\
            03-docs.sed
        "

    }
```

As result, '`centos-art.sh`' will produce '`01-welcome.png`', '`02-donate.png`' and '`04-support.png`' using the paragraph-based design template, but '`03-docs.png`' using the list-based design template.

2.37.1.4 Design template with translation (optimized+flexibility)

In the production models we've seen so far, there are design templates to produce untranslated images and translation files which combiend with design templates produce translated images. That may seems like all our needs are covered, doesn't it? Well, it *almost* does.

Generally, we use design templates to define how final images will look like. Generally, each renderable directory structure has one '`Tpl/`' directory where we organize design templates for that identity component. So, we can say that there is only one unique design template definition for each identity component; or what is the same, said differently, identity components can be produced in one way only, the way its own design template directory specifies. This is not enough for theme production. It is a limitation, indeed.

Initially, to create one theme, we created one renderable directory structure for each theme component. When we found ourselves with many themes, and components inside them, it was obvious that the same design model was duplicated inside each theme. As design models were independently one another, if we changed one theme's design model, that change was useless to other themes. So, in order to reuse design model changes, we unified design models into one common directory structure.

With design models unified in a common structure, another problem rose up. As design models also had the visual style of theme components, there was no difference between themes, so there was no apparent need to have an independent theme directory structure for each different theme. So, it was also needed to separate visual styles from design models.

At this point there are two independent worklines: one directory structure to store design models (the final image characteristics [i.e., dimensions, translation markers, etc.]) and one directory structure to store visual styles (the final image visual style [i.e., the image look and feel]). So, it is possible to handle both different design models and different visual styles independtly one another and later create combinations among them using '`centos-art.sh`'.

For example, consider the following configuration:

One entry under '`trunk/Identity/Themes/Models/`':

The design model entry exists to organize design model files (similar to design templates). Both design models and design templates are very similar; they both should have the `CENTOSARTWORK` export id present to identify the exportation area, translation marks, etc. However, design models do use dynamic backgrounds inclusion while design templates don't.

```
                              THEMEMODEL | | The bond component
                                   |<----| |--------------------->|
trunk/Identity/Themes/Models/Default/Distro/Anaconda/Progress/
|-- paragraph.svg
`-- list.svg
```

Inisde design models, dynamic backgrounds are required in order for different artistic motifs to reuse common design models. Firstly, in order to create dynamic backgrounds inside design models, we import a bitmap to cover design model's background and later, update design model's path information to replace fixed values to dynamic values.

One entry under 'trunk/Identity/Themes/Motifs/':

The artistic motif entry defines the visual style we want to produce images for, only. Final images (i.e., those built from combining both design models and artistic motif backrounds) are not stored here, but under branches directory structure. In the artistic motif entry, we only define those images that cannot be produced automatically by 'centos-art.sh' (e.g., Backgrounds, Color information, Screenshots, etc.).

```
                         Artistic motif name | | Artistic motif backgrounds
                                   |<-------| |-------->|
trunk/Identity/Themes/Motifs/TreeFlower/Backgrounds/
|-- Img
|   |-- Png
|   |   |-- 510x300.png
|   |   `-- 510x300-final.png
|   `-- Jpg
|       |-- 510x300.jpg
|       `-- 510x300-final.jpg
|-- Tpl
|   `-- 510x300.svg
`-- Xcf
    `-- 510x300.xcf
```

One entry under 'trunk/Translations/':

The translation entry specifies, by means of translation files, the language-specific information we want to produce image for. When we create the translation entry we don't use the name of neither design model nor artistic motif, just the design model component we want to produce images for.

```
                                        | The bond component
                                        |--------------------->|
trunk/Translations/Identity/Themes/Distro/Anaconda/Progress/
`-- 5
    |-- en
    |   |-- 01-welcome.sed
    |   |-- 02-donate.sed
    |   `-- 03-docs.sed
    `-- es
        |-- 01-welcome.sed
        |-- 02-donate.sed
        `-- 03-docs.sed
```

One entry under ‘trunk/Scripts/Bash/Functions/Render/Config/’:

There is one pre-rendition configuration script for each theme component. So, each time a theme component is rendered, its pre-rendition configuration script is evaluated to teach ‘centos-art.sh’ how to render the component.

```
trunk/Scripts/Bash/Functions/Render/Config/Identity/Themes/Distro/Anaconda/Prog
‘-- render.conf.sh
```

In this configuration the pre-rendition configuration script (‘render.conf.sh’) would look like the following:

```
function render_loadConfig {

    # Define rendition actions.
    ACTIONS[0]='BASE:renderImage'

    # Define matching list.
    MATCHINGLIST="\
    paragraph.svg:\
        01-welcome.sed\
        02-donate.sed
    list.svg:\
        03-docs.sed
        "

    # Deifne theme model.
    THEMEMODEL='Default'

}
```

The production flow of “optimize+flexibility” configuration. . .

2.37.2 Renderable translation directory structures

Translation directory structures are auxiliar structures of renderable identity directory structures. There is one translation directory structure for each renderable identity directory structure. Inside translation directory structures we organize translation files used by renderable identity directory structures that produce translated images. Renderable identity directory structures that produce untranslated images don’t use translation files, but they do use a translation directory structure, an empty translation directory structure, to be precise.

In order to aliviate production of translation file, we made translation directory structures renderable adding a template (‘Tpl/’) directory structure to handle common content inside translation files. This way, we work on translation templates and later use ‘centos-art.sh’ to produce specific translation files (based on translation templates) for different information (e.g., languages, release numbers, architectures, etc.).

If for some reason, translation files get far from translation templates and translation templates become incovenient to produce such translation files then, care should be taken to avoid replacing the content of translation files with the content of translation templates when ‘centos-art.sh’ is executed to produce translation files from translation templates.

Inside renderable translation directory structures, ‘centos-art.sh’ can produce text-based files only.

2.37.3 Copying renderable directory structures

A renderable layout is formed by design models, design images, pre-rendition configuration scripts and translations files. This way, when we say to duplicate rendition stuff we are saying

to duplicate these four directory structures (i.e., design models, design images, pre-rendition configuration scripts, and related translations files).

When we duplicate directories, inside 'trunk/Identity' directory structure, we need to be aware of renderable layout described above and the source location used to perform the duplication action. The source location is relevant to centos-art.sh script in order to determine the required auxiliar information inside directory structures that need to be copied too (otherwise we may end up with orphan directory structures unable to be rendered, due the absence of required information).

In order for a renderable directory structure to be valid, the new directory structure copied should match the following conditions:

1. To have a unique directory structure under '`trunk/Identity`', organized by any one of the above organizational designs above.
2. To have a unique directory structure under '`trunk/Translations`' to store translation files.
3. To have a unique directory structure under '`trunk/Scripts/Bash/Functions/Render/Config`' to set pre-rendition configuration script.

As convenction, the `render_doCopy` function uses '`trunk/Identity`' directory structure as source location. Once the '`trunk/Identity`' directory structure has been specified and verified, the related path information is built from it and copied automatically to the new location specified by *FLAG_TO* variable.

Design templates + No translation:

Command: - centos-art render –copy=trunk/Identity/Path/To/Dir –to=trunk/Identity/NewPath/To/Dir

Sources: - trunk/Identity/Path/To/Dir - trunk/Translations/Identity/Path/To/Dir - trunk/Scripts/Bash/Functions/Render/Config/Identity/Path/To/Dir

Targets: - trunk/Identity/NewPath/To/Dir - trunk/Translations/Identity/NewPath/To/Dir - trunk/Scripts/Bash/Functions/Render/Config/Identity/NewPath/To/Dir

Renderable layout 2:

Command: - centos-art render –copy=trunk/Identity/Themes/Motifs/TreeFlower \ –to=trunk/Identity/Themes/Motifs/NewPath/To/Dir

Sources: - trunk/Identity/Themes/Motifs/TreeFlower - trunk/Translations/Identity/Themes - trunk/Translations/Identity/Themes/Motifs/TreeFlower - trunk/Scripts/Bash/Functions/Render/Config/Ide - trunk/Scripts/Bash/Functions/Render/Config/Identity/Themes/Motifs/TreeFlower

Targets: - trunk/Identity/Themes/Motifs/NewPath/To/Dir - trunk/Translations/Identity/Themes - trunk/Translations/Identity/Themes/Motifs/NewPath/To/Dir - trunk/Scripts/Bash/Functions/Render/Con - trunk/Scripts/Bash/Functions/Render/Config/Identity/Themes/Motifs/NewPath/To/Dir

Notice that design models are not included in source or target locations. This is intentional. In "Renderable layout 2", design models live by their own, they just exist, they are there, available for any artistic motif to use. By default 'Themes/Models/Default' design model directory structure is used, but other design models directory structures (under Themes/Models/) can be created and used changing the value of THEMEMODEL variable inside the pre-rendition configuration script of the artistic motif source location you want to produce.

Notice how translations and pre-rendition configuration scripts may both be equal in source and target. This is because such structures are common to all artistic motifs (the default values to use when no specific values are provided).

- The common directory structures are not copied or deleted. We cannot copy a directory structure to itself.

- The common directory structures represent the default value to use when no specific translations and/or pre-rendition configuration script are provided inside source location.

- The specific directory structures, if present, are both copiable and removable. This is, when you perform a copy or delete action from source, that source specific auxiliar directories are transfered in the copy action to a new location (that specified by FLAG_TO variable).

- When translations and/or pre-rendition configuration scripts are found inside the source directory structure, the centos-art.sh script loads common auxiliar directories first and later specific auxiliar directories. This way, identity rendition of source locations can be customized idividually over the base of common default values.

- The specific auxiliar directories are optional.

- The common auxiliar directories should be present always. This is, in order to provide the information required by render functionality (i.e., to make it functional in the more basic level of its existence).

Notice how the duplication process is done from 'trunk/Identity' on, not the oposite. If you try to duplicate a translation structure (or similar auxiliar directory structures like pre-rendition configuration scripts), the 'trunk/Identity' for that translation is not created. This limitation is impossed by the fact that many 'trunk/Identity' directory structures may reuse/share the same translation directory structure. We cannot delete one translation (or similar) directory structures while a related 'trunk/Identity/' directory structure is still in need of it.

The 'render_doCopy' functionality does duplicate directory structures directly involved in rendition process only. Once such directories have been duplicated, the functionality stops thereat.

2.37.4 Usage

- ...

2.37.5 See also

2.38 The '`trunk/Scripts/Functions/Render/Config`' Directory

2.38.1 Goals

The '`trunk/Scripts/Bash/Config`' directory exists to oraganize pre-rendering configuration scripts.

2.38.2 Description

Pre-rendering configuration scripts let you customize the way `centos-art.sh` script renders identity and translation repository entries. Pre-rendering configuration scripts are '`render.conf.sh`' files with `render_loadConfig` function definition inside.

There is one '`render.conf.sh`' file for each pre-rendering configuration entry. Pre-rendering configuration entries can be based both on identity and translation repository entires. Pre-rendering configuration entries are required for each identity entry, but not for translation entries.

2.38.2.1 The '`render.conf.sh`' identity model

Inside CentOS Artwork Repository, we consider identity entries to all directories under '`trunk/Identity`' directory. Identity entries can be image-based or text-based. When you render image-based identity entries you need to use image-based pre-rendering configuration scripts. Likewise, when you render text-based identity entries you need to use text-based pre-rendering configuration scripts.

Inside identity pre-rendering configuration scripts, image-based pre-rendering configuration scripts look like the following:

```
#!/bin/bash
```

```
function render_loadConfig {

    # Define rendering actions.
    ACTIONS[0]='BASE:renderImage'
    ACTIONS[1]='POST:renderFormats: tif xpm pdf ppm'

}
```

Inside identity pre-rendering configuration scripts, text-based pre-rendering configuration scripts look like the following:

```
#!/bin/bash

function render_loadConfig {

    # Define rendering actions.
    ACTIONS[0]='BASE:renderText'
    ACTIONS[1]='POST:formatText: --width=70 --uniform-spacing'

}
```

When using identity pre-rendering configuration scripts, you can extend both image-based and text-based pre-rendering configuration scripts using image-based and text-based post-rendering actions, respectively.

2.38.2.2 The 'render.conf.sh' translation model

Translation pre-rendering configuration scripts take precedence before default translation rendering action. Translation pre-rendering actions are useful when default translation rendering action do not fit itself to translation entry rendering requirements.

2.38.2.3 The 'render.conf.sh' rendering actions

Inside both image-based and text-based identity pre-rendering configuration scripts, we use the '`ACTIONS`' array variable to define the way `centos-art.sh` script performs identity rendering. Identity rendering is organized by one '`BASE`' action, and optional '`POST`' and '`LAST`' rendering actions.

The '`BASE`' action specifies what kind of rendering does the `centos-art.sh` script will perform with the files related to the pre-rendering configuration script. The '`BASE`' action is required. Possible values to '`BASE`' action are either '`renderImage`' or '`renderText`' only.

To specify the '`BASE`' action you need to set the '`BASE:`' string followed by one of the possible values. For example, if you want to render images, consider the following definition of '`BASE`' action:

```
ACTIONS[0]='BASE:renderImage'
```

Only one '`BASE`' action must be specified. If more than one '`BASE`' action is specified, the last one is used. If no '`BASE`' action is specified at all, an error is triggered and the `centos-art.sh` script ends its execution.

The '`POST`' action specifies which action to apply for each file rendered (at the rendering time). This action is optional. You can set many different '`POST`' actions to apply many different actions over the same already rendered file. Possible values to '`POST`' action are '`renderFormats`', '`renderSyslinux`', '`renderGrub`', etc.

To specify the '`POST`' action, you need to use set the '`POST:`' followed by the function name of the action you want to perform. The exact form depends on your needs. For example, consider

the following example to produce ‘xpm’, ‘jpg’, and ‘tif’ images, based on already rendered ‘png’ image, and also organize the produced files in directories named as their own extensions:

```
ACTIONS[0]='BASE:renderImage'
ACTIONS[1]='POST:renderFormats: xpm jpg tif'
ACTIONS[2]='POST:groupByFormat: png xpm jpg tif'
```

In the previous example, file organization takes place at the moment of rendering, just after producing the ‘png’ base file and before going to the next file in the list of files to render. If you don’t want to organized the produced files in directories named as their own extensions, just remove the ‘POST:groupByFormat’ action line:

```
ACTIONS[0]='BASE:renderImage'
ACTIONS[1]='POST:renderFormats: xpm jpg tif'
```

The ‘LAST’ action specifies which actions to apply once the last file in the list of files to process has been rendered. The ‘LAST’ action is optional. Possible values for ‘LAST’ actions may be ‘groupByFormat’, ‘renderGdmTgz’, etc.

> **Note** — **Removed**(xref:trunk Scripts Bash Functions Render) —, to know more about possible values for ‘BASE’, ‘POST’ and ‘LAST’ action definitions.

To specify the ‘LAST’ action, you need to set the ‘LAST:’ string followed by the function name of the action you want to perform. For example, consider the following example if you want to render all files first and organize them later:

```
ACTIONS[0]='BASE:renderImage'
ACTIONS[1]='POST:renderFormats: xpm jpg tif'
ACTIONS[2]='LAST:groupByformat: png xpm jpg tif'
```

2.38.3 Usage

Use the following commands to administer both identity and translation pre-rendering configuration scripts:

‘`centos-art config --create='path/to/dir/'`’
: Use this command to create ‘path/to/dir’ related pre-rendering configuration script.

‘`centos-art config --edit='path/to/dir/'`’
: Use this command to edit ‘path/to/dir’ related pre-rendering configuration script.

‘`centos-art config --read='path/to/dir/'`’
: Use this command to read ‘path/to/dir’ related pre-rendering configuration script.

‘`centos-art config --remove='path/to/dir/'`’
: Use this command to remove ‘path/to/dir’ related pre-rendering configuration script.

In the commands above, ‘path/to/dir’ refers to one renderable directory path under ‘trunk/Identity’ or ‘trunk/Translations’ structures only.

2.38.4 See also

Index

List of Figures