

# CentOS Artwork Repository

---

The filesystem

Alain Reguera Delgado

---

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

<b>1</b>	<b>branches</b> .....	<b>1</b>
<b>2</b>	<b>tags</b> .....	<b>2</b>
<b>3</b>	<b>trunk</b> .....	<b>3</b>
3.1	trunk/Identity .....	3
3.1.1	Goals .....	3
3.1.2	Description .....	3
3.1.3	Usage .....	3
3.1.4	File name conventions .....	4
3.1.4.1	When text-based files are rendered .....	4
3.1.4.2	When image-based files are rendered .....	4
3.1.5	See also .....	4
3.1.6	References .....	4
3.2	trunk/Identity/Brands .....	4
3.2.1	Goals .....	4
3.2.2	Description .....	4
3.2.3	Usage .....	4
3.2.4	See also .....	4
3.3	trunk/Identity/Budokan .....	4
3.3.1	Goals .....	4
3.3.2	Description .....	5
3.3.3	Usage .....	5
3.3.4	See also .....	5
3.4	trunk/Identity/Fonts .....	5
3.4.1	Goals .....	5
3.4.2	Description .....	5
3.4.3	Usage .....	5
3.4.4	See also .....	6
3.5	trunk/Identity/Icons .....	6
3.5.1	Goals .....	6
3.5.2	Description .....	6
3.5.3	Usage .....	6
3.5.4	See also .....	6
3.6	trunk/Identity/Isolinux .....	6
3.6.1	Goals .....	6
3.6.2	Description .....	6
3.6.3	Usage .....	6
3.6.4	See also .....	6
3.7	trunk/Identity/Models .....	6
3.7.1	Goals .....	6
3.7.2	Description .....	6
3.7.3	Usage .....	6
3.7.4	See also .....	6
3.8	trunk/Identity/Models/Css .....	7
3.8.1	Goals .....	7
3.8.2	Description .....	7

3.8.3	Usage .....	7
3.8.4	See also .....	7
3.9	trunk/Identity/Models/Html .....	7
3.9.1	Goals .....	7
3.9.2	Description .....	7
3.9.3	Usage .....	7
3.9.4	See also .....	7
3.10	trunk/Identity/Models/Img/Promo/Web .....	7
3.10.1	Goals .....	7
3.10.2	Description .....	7
3.10.3	Usage .....	7
3.10.4	See also .....	7
3.11	trunk/Identity/Models/Tpl .....	7
3.11.1	Goals .....	7
3.11.2	Description .....	7
3.11.3	Usage .....	7
3.11.4	See also .....	7
3.12	trunk/Identity/Models/Tpl/Promo/Web .....	8
3.12.1	Goals .....	8
3.12.2	The CentOS web environment .....	8
3.12.2.1	Design model (without ads) .....	8
3.12.2.2	Design model (with ads) .....	8
3.12.2.3	HTML definitions .....	9
3.12.2.4	Controlling visual style .....	9
3.12.2.5	Producing visual style .....	9
3.12.2.6	Navigation .....	9
3.12.2.7	Development and release cycle .....	9
3.12.2.8	The [webenv-test] repository .....	11
3.12.2.9	The [webenv] repository .....	11
3.12.2.10	Priority configuration .....	12
3.12.3	Usage .....	12
3.12.4	See also .....	12
3.13	trunk/Identity/Models/Xcf .....	12
3.13.1	Goals .....	12
3.13.2	Description .....	12
3.13.3	Usage .....	12
3.13.4	See also .....	12
3.14	trunk/Identity/Release .....	12
3.14.1	Goals .....	12
3.14.2	Description .....	12
3.14.3	Usage .....	12
3.14.4	See also .....	12
3.15	trunk/Identity/Themes .....	12
3.15.1	Goals .....	12
3.15.2	Description .....	12
3.15.3	Usage .....	12
3.15.4	See also .....	12
3.16	trunk/Identity/Themes/Models .....	13
3.16.1	Goals .....	13
3.16.2	Description .....	13
3.16.3	Usage .....	13
3.16.4	See also .....	13
3.17	trunk/Identity/Themes/Models/Alternative .....	13
3.17.1	Goals .....	13

3.17.2	Description	13
3.17.3	Usage	13
3.17.4	See also	13
3.18	trunk/Identity/Themes/Models/Default	13
3.18.1	Goals	13
3.18.2	Description	13
3.18.3	Usage	13
3.18.4	See also	14
3.19	trunk/Identity/Themes/Models/Default/Distro	14
3.19.1	Goals	14
3.19.2	Description	14
3.19.2.1	One theme for all major releases	14
3.19.2.2	One theme for each major release	14
3.19.3	Usage	15
3.19.4	See also	15
3.20	trunk/Identity/Themes/Models/Default/Distro/Anaconda	15
3.20.1	Goals	15
3.20.2	Description	15
3.20.3	Usage	15
3.20.4	See also	15
3.21	trunk/Identity/Themes/Models/Default/Promo	15
3.21.1	Goals	15
3.21.2	Description	15
3.21.3	Usage	15
3.21.4	See also	15
3.22	trunk/Identity/Themes/Models/Default/Web	15
3.22.1	Goals	15
3.22.2	Description	15
3.22.3	Usage	16
3.22.4	See also	16
3.23	trunk/Identity/Themes/Motifs	16
3.23.1	Goals	16
3.23.2	Description	16
3.23.3	Usage	17
3.23.4	See also	17
3.24	trunk/Identity/Themes/Motifs/Flame	17
3.24.1	Goals	17
3.24.2	Description	17
3.24.3	Construction	18
3.24.3.1	Step 1: Set image size	18
3.24.3.2	Step 2: Add base color and pattern information	19
3.24.3.3	Step 3: Add flame motif	19
3.24.3.4	Step 4: Add foreground color	19
3.24.4	See also	20
3.25	trunk/Identity/Themes/Motifs/Modern	20
3.25.1	Presentation	20
3.25.2	Construction	20
3.25.3	Usage	20
3.25.4	See also	20
3.26	trunk/Identity/Themes/Motifs/Modern/Backgrounds	20
3.26.1	Goals	20
3.26.2	Description	20
3.26.3	Usage	21
3.26.4	See also	21

3.27	trunk/Identity/Themes/Motifs/Modern/Backgrounds/Img	21
3.27.1	Goals	21
3.27.2	Description	21
3.27.3	Usage	21
3.27.4	See also	21
3.28	trunk/Identity/Themes/Motifs/Modern/Backgrounds/Tpl	22
3.28.1	Goals	22
3.28.2	Description	22
3.28.3	Usage	22
3.28.4	See also	22
3.29	trunk/Identity/Themes/Motifs/Modern/Backgrounds/Xcf	22
3.29.1	Goals	22
3.29.2	Description	22
3.29.3	Usage	22
3.29.4	See also	22
3.30	trunk/Identity/Themes/Motifs/Modern/Distro/Anaconda/Progress	22
3.30.1	Goals	22
3.30.2	Description	22
3.30.3	Usage	22
3.30.4	See also	23
3.31	trunk/Identity/Themes/Motifs/Modern/Palettes	23
3.31.1	Goals	23
3.31.2	Description	23
3.31.3	Usage	23
3.31.4	See also	23
3.32	trunk/Identity/Themes/Motifs/TreeFlower	23
3.32.1	Goals	23
3.32.2	Description	23
3.32.3	Usage	23
3.32.4	See also	23
3.33	trunk/Identity/Themes/Motifs/TreeFlower/Backgrounds	23
3.33.1	Goals	24
3.33.2	Description	24
3.33.2.1	Desktop background	24
3.33.2.2	Anaconda Prompt (syslinux) background	25
3.33.2.3	Grub background	27
3.33.3	Usage	27
3.33.4	See also	27
3.34	trunk/Identity/Widgets	27
3.34.1	Goals	28
3.34.2	Description	28
3.34.3	Usage	28
3.34.4	See also	28
3.35	trunk/Locales	28
3.35.1	Goals	28
3.35.2	Description	28
3.35.2.1	Translation Entries	28
3.35.2.2	Translation Markers	29
3.35.2.3	Translation Files	30
3.35.2.4	Template Translation Files	31
3.35.2.5	Common Translation Files	33
3.35.2.6	Specific Translation Files	34
3.35.2.7	Translation Rendering	34
3.35.2.8	Translation (Pre-)Rendering Configuration Scripts	34

3.35.2.9	Translation Rendering Default Functionality	35
3.35.3	Usage	36
3.35.4	See also	36
3.36	trunk/Locales/Identity	36
3.36.1	Goals	36
3.36.2	Description	36
3.36.3	Usage	36
3.36.4	See also	36
3.37	trunk/Locales/Identity/Brands	36
3.37.1	Goals	36
3.37.2	Description	36
3.37.2.1	Conventional file names	37
3.37.2.2	Numeric file names	37
3.37.2.3	Translation markers	37
3.37.3	Usage	37
3.37.4	See also	37
3.38	trunk/Locales/Identity/Brands/Tpl	38
3.38.1	Goals	38
3.38.2	Description	38
3.38.3	Usage	38
3.38.4	See also	38
3.39	trunk/Locales/Identity/Fonts	38
3.39.1	Goals	38
3.39.2	Description	38
3.39.2.1	Translation Markers	38
3.39.3	Usage	38
3.39.4	See also	38
3.40	trunk/Locales/Identity/Models	39
3.40.1	Goals	39
3.40.2	Description	39
3.40.3	Usage	39
3.40.4	See also	39
3.41	trunk/Locales/Identity/Release	39
3.41.1	Goals	39
3.41.2	Description	39
3.41.3	Usage	39
3.41.4	See also	39
3.42	trunk/Locales/Identity/Themes	39
3.42.1	Goals	39
3.42.2	Description	39
3.42.3	Usage	39
3.42.4	See also	39
3.43	trunk/Locales/Identity/Themes/Backgrounds	39
3.43.1	Goals	39
3.43.2	Description	39
3.43.3	Usage	39
3.43.4	See also	39
3.44	trunk/Locales/Identity/Themes/Distro/Anaconda/Progress	39
3.44.1	Goals	39
3.44.2	Description	40
3.44.3	Usage	41
3.44.4	See also	41
3.45	trunk/Locales/Identity/Widgets	41
3.45.1	Goals	41

3.45.2	Description	41
3.45.3	Usage	41
3.45.4	See also	41
3.46	trunk/Manuals	41
3.46.1	Goals	41
3.46.2	Description	41
3.46.3	Usage	41
3.46.4	See also	41
3.47	trunk/Scripts	41
3.47.1	Goals	41
3.47.2	Description	42
3.47.3	Usage	42
3.47.4	See also	42
3.48	trunk/Scripts/Bash	42
3.48.1	Goals	42
3.48.2	Description	42
3.48.3	Usage	44
3.48.4	See also	44
3.49	trunk/Scripts/Bash/Functions	44
3.49.1	Goals	44
3.49.2	Description	44
3.49.3	Usage	48
3.49.3.1	Global variables	48
3.49.3.2	Global functions	51
3.49.3.3	Specific functions	60
3.49.4	See also	60
3.50	trunk/Scripts/Bash/Functions/Html	60
3.50.1	Goals	60
3.50.2	Description	60
3.50.3	Usage	60
3.50.4	See also	60
3.51	trunk/Scripts/Bash/Functions/Locale	60
3.51.1	Goals	60
3.51.2	Description	60
3.51.3	Usage	61
3.51.4	See also	61
3.52	trunk/Scripts/Bash/Functions/Manual	61
3.52.1	Goals	61
3.52.2	Description	61
3.52.3	Usage	62
3.52.4	See also	62
3.53	trunk/Scripts/Bash/Functions/Path	62
3.53.1	Goals	62
3.53.2	Description	62
3.53.2.1	Repository layout	62
3.53.2.2	Repository name conventions	62
3.53.2.3	Repository work flow	63
3.53.2.4	Parallel directories	64
3.53.2.5	Synchronizing path information	64
3.53.2.6	What is the right place to store it?	65
3.53.3	Usage	66
3.53.4	See also	66
3.54	trunk/Scripts/Bash/Functions/Render	66
3.54.1	Renderable identity directory structures	66



3.54.1.1	Design template without translation	66
3.54.1.2	Design template with translation (one-to-one)	69
3.54.1.3	Design template with translation (optimized)	70
3.54.1.4	Design template with translation (optimized+flexibility)	71
3.54.2	Renderable translation directory structures	74
3.54.3	Copying renderable directory structures	74
3.54.4	Usage	76
3.54.5	See also	76
3.55	trunk/Scripts/Bash/Functions/Render/Config	76
3.55.1	Goals	76
3.55.2	Description	76
3.55.2.1	The ‘render.conf.sh’ identity model	76
3.55.2.2	The ‘render.conf.sh’ translation model	77
3.55.2.3	The ‘render.conf.sh’ rendering actions	77
3.55.3	Usage	78
3.55.4	See also	78
3.56	trunk/Scripts/Bash/Functions/Shell	78
3.56.1	Goals	78
3.56.2	Description	78
3.56.3	Usage	81
3.56.4	See also	81
3.57	trunk/Scripts/Bash/Functions/Svg	81
3.57.1	Goals	81
3.57.2	Description	81
3.57.2.1	Metadata maintainance	81
3.57.2.2	Unused definitions	83
3.57.3	Usage	83
3.57.4	See also	83
3.58	trunk/Scripts/Bash/Functions/Verify	83
3.58.1	Goals	83
3.58.2	Description	84
3.58.2.1	Packages	84
3.58.2.2	Links	84
3.58.2.3	Environment variables	85
3.58.3	Usage	86
3.58.4	See also	86
3.59	trunk/Scripts/Bash/Locale	86
3.59.1	Goals	87
3.59.2	Description	87
3.59.3	Usage	87
3.59.4	See also	87
3.60	trunk/Scripts/Perl	87
3.60.1	Goals	87
3.60.2	Description	87
3.60.3	Usage	87
3.60.4	See also	87
3.61	trunk/Scripts/Python	87
3.61.1	Goals	87
3.61.2	Description	87
3.61.3	Usage	87
3.61.4	See also	87

**List of Figures** ..... **89**

# 1 branches

## 2 tags

## 3 trunk

### 3.1 trunk/Identity

#### 3.1.1 Goals

The ‘trunk/Identity’ directory exists to organize CentOS corporate identity artworks.

#### 3.1.2 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 visual 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.

The CentOS Project settles down its corporate visual identity on a “monolithic corporate visual identity structure”. In this structure The CentOS Project uses one unique name (The CentOS Brand) and one unique visual style (The CentOS Default Theme) in all its manifestations.

#### The CentOS Brands

The CentOS brand is the 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 distributions, the CentOS web sites, the CentOS promotion stuff, etc.

See [Section 3.2 \[trunk Identity Brands\], page 4](#), for more information.

#### The CentOS Themes

The CentOS themes are a set of image files connected all together by one unique visual style. Each theme is organized in different visual manifestations, in order to cover each visual manifestation of The CentOS Project (i.e., distributions, websites, promotion stuff, etc.). See [Section 3.15 \[trunk Identity Themes\], page 12](#), for more information.

Inside 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 websites, using the distribution, or acting on social events, the one unique name and one unique visual style connect them all to say: Hey! we are all parts of the CentOS project. And, probably, some visitor will say: Can I join the party? Yes you can, it is free. :)

#### 3.1.3 Usage

To produce identity artworks, use the following commands:

```
‘centos-art render ‘path/to/dir’
```

When ‘path/to/dir’ refers to one renderable directory under ‘trunk/Identity’, this command renders identity artworks using both related design models and related translation files.

```
‘centos-art render ‘path/to/dir’ --filter=‘pattern’
```

When ‘path/to/dir’ refers to one renderable directory under ‘trunk/Identity’, this command renders identity artworks using both related design models and related

translation files that match the regular expression passed in ‘`--filter=’pattern’`’ argument.

To control the number of files produced by `centos-art` command, you need to look into the translation path and provide a regular expression pattern that matches the translation path, or paths, related to the file, or files, you want to produce.

The regular expression pattern you provide to `centos-art` command is applied to the translation path from its very beginning. It is not the same to say ‘5/en/01-welcome’ that ‘01-welcome’, the first expression matches but the last one does not.

When using ‘`--filter=’pattern’`’ you don’t need to specify the file extension. It is removed from translation path before applying the pattern, so it doesn’t count here.

### 3.1.4 File name conventions

As file name convention, inside CentOS Artwork Repository, both text-based and image-based file name produced by `centos-art.sh` script has the same name of their translation files without the ‘.sed’ extension. The file extension is set as follow:

#### 3.1.4.1 When text-based files are rendered

Text-based files end up having the same extension of their design template file.

#### 3.1.4.2 When image-based files are rendered

Image-based files always end up having the ‘.png’ extension.

**Tip** Once ‘.png’ images are created, other image formats may be created using the `renderFormats` post-rendering action, inside the image-based related pre-rendering configuration script.

See [Section 3.48 \[trunk Scripts Bash\]](#), page 42, for more information.

### 3.1.5 See also

### 3.1.6 References

- [http://en.wikipedia.org/Corporate\\_identity](http://en.wikipedia.org/Corporate_identity) (and related links).

## 3.2 trunk/Identity/Brands

### 3.2.1 Goals

- ...

### 3.2.2 Description

### 3.2.3 Usage

### 3.2.4 See also

## 3.3 trunk/Identity/Budokan

### 3.3.1 Goals

- Test page.

This is a cross reference link using two lines — **Removed**(xref:trunk Identity Budokan Ozzy) —, which help me to test manual\_updateCrossReferences, manual\_deleteCrossReferences, manual\_updateMenu and manual\_updateNodes functionalities.

This is a cross reference link using two lines (— **Removed**(pxref:trunk Identity Budokan Ozzy) —) which help me to test manual\_updateCrossReferences, manual\_deleteCrossReferences, manual\_updateMenu and manual\_updateNodes functionalities.

This is a cross reference link using two lines — **Removed**(ref:trunk Identity Budokan Ozzy) —, which help me to test manual\_updateCrossReferences, manual\_deleteCrossReferences, manual\_updateMenu and manual\_updateNodes functionalities.

### 3.3.2 Description

- ...

### 3.3.3 Usage

- ...

### 3.3.4 See also

## 3.4 trunk/Identity/Fonts

### 3.4.1 Goals

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

### 3.4.2 Description

### 3.4.3 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 convention 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 ‘e1’ (l) or letter ‘ai’ (i)).

As CentOS logo typography convention, 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](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.

### 3.4.4 See also

## 3.5 trunk/Identity/Icons

### 3.5.1 Goals

- ...

### 3.5.2 Description

### 3.5.3 Usage

### 3.5.4 See also

## 3.6 trunk/Identity/Isolinux

### 3.6.1 Goals

- ...

### 3.6.2 Description

### 3.6.3 Usage

### 3.6.4 See also

## 3.7 trunk/Identity/Models

### 3.7.1 Goals

This section exists to organize design models.

### 3.7.2 Description

Design models are representative designs useful to understand how to build artworks.

### 3.7.3 Usage

### 3.7.4 See also



## **3.8 trunk/Identity/Models/Css**

### **3.8.1 Goals**

This directory exists to provide common style sheets (CSS) definitions to HTML design models.

### **3.8.2 Description**

- ...

### **3.8.3 Usage**

- ...

### **3.8.4 See also**

## **3.9 trunk/Identity/Models/Html**

### **3.9.1 Goals**

- ...

### **3.9.2 Description**

- ...

### **3.9.3 Usage**

- ...

### **3.9.4 See also**

## **3.10 trunk/Identity/Models/Img/Promo/Web**

### **3.10.1 Goals**

- Provide images related to CentOS web interface.

### **3.10.2 Description**

- ...

### **3.10.3 Usage**

- ...

### **3.10.4 See also**

## **3.11 trunk/Identity/Models/Tpl**

### **3.11.1 Goals**

- ...

### **3.11.2 Description**

- ...

### **3.11.3 Usage**

- ...

### **3.11.4 See also**

## 3.12 trunk/Identity/Models/Tpl/Promo/Web

### 3.12.1 Goals

Organize scalable vector graphics (svg) to help describe the CentOS web environment.

### 3.12.2 The CentOS web environment

Inside CentOS corporate identity, the CentOS web environment is considered a promotion component. 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 3.1 \[trunk Identity\], page 3](#)), 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<sup>1</sup> 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 inevitable 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.

#### 3.12.2.1 Design model (without ads)

#### 3.12.2.2 Design model (with ads)

---

<sup>1</sup> The theme support of Mailman may be introduced in mailman-3.x.x release.

### 3.12.2.3 HTML definitions

### 3.12.2.4 Controlling visual style

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

#### Webenv header background

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

#### CSS definitions

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

### 3.12.2.5 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 your 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, uploaded, and installed from [webenv] or [webenv-test] repositories.

### 3.12.2.6 Navigation

Inside CentOS web environment, the one-step navigation 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.

### 3.12.2.7 Development and release cycle

The CentOS web environment development and release 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 choice, 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 achieved using a [webenv-test] yum repository.

**Note** The [webenv-test] repository is not shipped inside CentOS distribution default yum configuration. 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 page is updated 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 is 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 is 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 committed by CentOS quality assurance team.

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

#### 3.12.2.8 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=$basearch&repo=webenv-test
#baseurl=http://mirror.centos.org/centos/$releasever/webenv-test/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-$releasever
enabled=1
priority=10

```

#### 3.12.2.9 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
```

### 3.12.2.10 Priority configuration

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

### 3.12.3 Usage

- ...

### 3.12.4 See also

## 3.13 trunk/Identity/Models/Xcf

### 3.13.1 Goals

- ...

### 3.13.2 Description

- ...

### 3.13.3 Usage

- ...

### 3.13.4 See also

## 3.14 trunk/Identity/Release

### 3.14.1 Goals

- ...

### 3.14.2 Description

### 3.14.3 Usage

### 3.14.4 See also

## 3.15 trunk/Identity/Themes

### 3.15.1 Goals

The ‘trunk/Identity/Themes/’ directory exists to organize production of CentOS themes.

### 3.15.2 Description

### 3.15.3 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.

### 3.15.4 See also

## 3.16 trunk/Identity/Themes/Models

### 3.16.1 Goals

- Organize theme models.

### 3.16.2 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.

### 3.16.3 Usage

Inside the framework location above, you find theme models organized by name. You can add your own theme models to the structure by adding a directory to the list. By default you have the ‘See [Section 3.18 \[trunk Identity Themes Models Default\], page 13,](#)’ and ‘See [Section 3.17 \[trunk Identity Themes Models Alternative\], page 13,](#)’ ready-to-use theme models.

### 3.16.4 See also

## 3.17 trunk/Identity/Themes/Models/Alternative

### 3.17.1 Goals

- ...

### 3.17.2 Description

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.

### 3.17.3 Usage

- ...

### 3.17.4 See also

## 3.18 trunk/Identity/Themes/Models/Default

### 3.18.1 Goals

This location stores CentOS default theme model. The default theme model of CentOS is used in all visual manifestations of CentOS Project corporate visual identity (e.g., distributions, web sites, promotion, etc.).

### 3.18.2 Description

### 3.18.3 Usage

Changing CentOS default theme is not very convenient because that affects the “recognition” of CentOS Project. Nevertheless, we are interested on seeing your art work propositions. Specially

if your art work is an improvement to the base idea behind CentOS default theme (**Modern**, squares and circles flowing up.).

If you are not happy with CentOS default theme, you can look inside CentOS alternative themes and download the one you are interested in. If you are not happy with any of the CentOS alternative themes available, then go and design your own CentOS alternative theme as described in See [Section 3.23 \[trunk Identity Themes Motifs\]](#), page 16.

### 3.18.4 See also

## 3.19 trunk/Identity/Themes/Models/Default/Distro

### 3.19.1 Goals

- ...

### 3.19.2 Description

It applies to all major releases of CentOS distribution.

#### 3.19.2.1 One theme for all major releases

Sometimes, specific visual manifestations are formed by common components which have internal differences. That is the case of CentOS distribution visual manifestation.

Since a visual style point of view, the CentOS distributions share common artwork components like Anaconda —to cover the CentOS distribution installation—, BootUp —to cover the CentOS distribution start up—, and Backgrounds —to cover the CentOS distribution desktop—. Now, since a technical point of view, those common artwork components are made of software improved constantly. So, we need to find a way to keep one unique name and one unique visual style in artwork components that have internal difference and also remark internal difference as well.

**Important** Remarking the CentOS release schema inside each major release of CentOS distribution —or similar visual manifestation— takes *high attention* inside The CentOS Project corporate visual identity. It should be very clear for people which major release of CentOS distribution they are using.

In order to remark the CentOS release schema, the CentOS Artwork SIG uses a release-specific brand design named “The CentOS Release Brand”. The CentOS release brand is composed by the CentOS logotype *and* the CentOS major release number (as specified in CentOS release schema definition). In this solution, the CentOS release brand is set inside all release-specific artworks (e.g., distribution, installation media, etc.) in remarkable way. The CentOS release brand is the design component that lets us remark the CentOS release schema inside the monolithic corporate visual identity structure we propose to use.

#### 3.19.2.2 One theme for each major release

Other way we’ve been using to remark CentOS release schema is applying one unique theme for *each* major release of CentOS distribution. That is, if we have 4 major releases of CentOS distribution, we need to provide 4 different themes to cover each CentOS distribution available.

Inside CentOS Artwork Repository, you can create many themes and that is very convenient. But using one unique theme for *each* major release of CentOS distribution would bring visual isolation among distributions, websites and promotion visual manifestations. If the CentOS project would maintain just one CentOS distribution (and many experienced graphic designers ready to create beautiful artworks) this model would be nice. Indeed, this model looks quite similar to that one used by Fedora project, doesn’t it. But no, the CentOS project maintains



near to 4 major releases of CentOS distribution in parallel, and that fact makes a huge difference since the corporate visual identity point of view.

If we use one unique theme for *each* major release of CentOS distribution, which one of those themes, does we use to cover other CentOS visual manifestations, like websites and promotion stuff?

In whatever case you choose some release-specific distribution user will be visually isolated from other CentOS visual manifestations like websites and promotion stuff, even if the CentOS brand is present in all visual manifestations. In such a case, probably, users will end up asking themselves, why my CentOS distribution has this design and the CentOS website another one? Isn't them on the same project? With luck the CentOS brand will exonerate user from visual isolation.

### 3.19.3 Usage

### 3.19.4 See also

## 3.20 trunk/Identity/Themes/Models/Default/Distro/Anaconda

### 3.20.1 Goals

- ...

### 3.20.2 Description

### 3.20.3 Usage

### 3.20.4 See also

## 3.21 trunk/Identity/Themes/Models/Default/Promo

### 3.21.1 Goals

- ...

### 3.21.2 Description

It applies to all tangible and non tangible items CentOS uses to promote its existence. Clothes, posters, installation media, stationery, release countdown images, banners, stickers, are all examples of promotion designs.

- ...

### 3.21.3 Usage

- ...

### 3.21.4 See also

## 3.22 trunk/Identity/Themes/Models/Default/Web

### 3.22.1 Goals

- ...

### 3.22.2 Description

It applies to all web applications CentOS uses to handle its needs (Ex. Portals, Wikis, Forums, Blogs, Bug Tracker). Anything involving HTML standards should be consider here.

- ...

### 3.22.3 Usage

- ...

### 3.22.4 See also

## 3.23 trunk/Identity/Themes/Motifs

### 3.23.1 Goals

The ‘trunk/Identity/Themes/Motifs’ directory exists to:

- Organize CentOS themes’ artistic motifs.

### 3.23.2 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
```

### 3.23.3 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/>).

### 3.23.4 See also

## 3.24 trunk/Identity/Themes/Motifs/Flame

### 3.24.1 Goals

This section describes the steps we followed to construct 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 (see [Section 3.1 \[trunk Identity\]](#), page 3).

### 3.24.2 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 3.48 \[trunk Scripts Bash\]](#), page 42) 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 ourselves 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 themselves, and so, the reason we use to define the word “theme” as: a set of images visually connected among themselves.

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:

```
trunk/Identity/Themes/Motifs/Flame/Backgrounds/Xcf/800x600.xcf-flame.def
```

### 3.24.3 Construction

#### 3.24.3.1 Step 1: Set image size

Create an empty image and fill the ‘`Background`’ layer with black (000000) color. Image dimensions depend on the final destination you plan to use the image for. For the sake of our construction example we used an image of 640x480 pixels and 300 pixels per inch (ppi).

### 3.24.3.2 Step 2: Add base color and pattern information

Create a new layer named `'Base'`, place it over `'Background'` layer and fill it with the base color (7800ff) you want to have your background image set in. Add a mask to `'Base'` layer using radial gradient and blur it. You may need to repeat this step more than once in order to achieve a comfortable black radial degradation on the right side of your design.

Duplicate `'Base'` layer and name it `'Paper'`. Place `'Paper'` layer over `'Base'` layer. Remove content of `'Paper'` layer and fill it with `'Paper (100x100)'` pattern. Once you've done with black radial degradation, reduce the `'Paper'` layer opacity to 20%.

Notice that when we duplicate one layer, the mask information related to layer is preserved from previous to next layer. This saves us some of the time required to produce different layers with the same mask information on them.

Duplicate `'Paper'` layer and rename it `'Stripes'`. Remove paper pattern from `'Stripes'` layer. Fill `'Stripes'` layer with `'Stripes (48x48)'` pattern and reduce the `'Stripes'` layer opacity to 15%.

### 3.24.3.3 Step 3: Add flame motif

Create a new layer named `'Flame'`. Set the foreground (003cff) and background (0084ff) colors to the gradient you want to build the flame motif.

To build flame motif, use the flame filter (`'Filters > Render > Nature > Flame...'`) on `'Flame'` layer. We used a layer mask, with a radial gradient on it to control the boundaries of flame motif on `'Flame'` layer.

Duplicate `'Flame'` layer and rename it `'Flame Blur'`. Place `'Flame Blur'` below `'Flame'` layer. Apply Gaussian blur filter (`'Filters > Blur > Gussian Blur...'`) until reaching the desired effect.

The opacity value, in `'Flame'` layers, may vary from one image to another based on the place the image will be finally placed on. For example, images used as desktop background have the `'Flame'` layer opacity set at 100% but `'Flame Blur'` is set to 70%. However, you may find that background images used in anaconda progress slides have opacity reduced differently, in order to reduce brightness in a way that texts could look clean and readable over it.

### 3.24.3.4 Step 4: Add foreground color

Create a new layer named `'Color'`, place it on top of all visible layers and fill it with plain color (4c005a). Reduce `'Color'` layer opacity to 20%. You can use the `'Color'` layer to control the right side color information you want to produce the image for.

Duplicate `'Flame'` layer and create a new layer named `'Color#1'`. Place `'Color#1'` layer on top of layer named `'Color'`. Remove the mask information from `'Color#1'` layer and recreate a new one using an inverted alpha channel as reference. Remove `'Color#1'` layer content and fill it back with plain black (000000) color. Reduce `'Color#1'` opacity to 20%. In this step we created a mask to protect the flame artistic motif from black color, so when we decrement or increment the opacity of layer, the flame artistic motif wouldn't be affected, just the environment surround it.

When you set color information, remember that the same artistic motif needs to be indexed to 14 and 16 colors, in order to produce Grub and Syslinux visual manifestations respectively. Using many different colors in the artistic motif may reduce the possibility of your design to fix all different situations in. Likewise, using more colors in one design, and less colors in another design will reduce the connectivity among your designs, since color information is relevant to visual identity.

When you propagate your artistic motif visual style to different visual manifestations of CentOS Project corporate visual identity, it is up to you to find out justice and compromise among all possible variables you may face.

### 3.24.4 See also

## 3.25 trunk/Identity/Themes/Motifs/Modern

### 3.25.1 Presentation

### 3.25.2 Construction

### 3.25.3 Usage

- ...

### 3.25.4 See also

## 3.26 trunk/Identity/Themes/Motifs/Modern/Backgrounds

### 3.26.1 Goals

- Organize background images for Modern theme.

### 3.26.2 Description

Inside ‘Motifs’ directory, the ‘Backgrounds/’ directory is used to create vectorial designs using Inkscape and background images using Gimp. Later, you can export background images as ‘.png’ and load them in your vectorial design project using the import feature of Inkscape.

You may need to repeat this technique for different screen resolutions. In that case you need to create one file for each screen resolution and do the appropriate linking inside .svg to .png files. For example if you need to produce background images in 800x600 you need to create the following file:

```
xcf/800x600.xcf
```

to produce the background image:

```
img/800x600-bg.png
```

which is loaded in:

```
svg/800x600.svg
```

to produce the final background image:

```
img/800x600.png
```

The ‘img/800x600.png’ background image is produced automatically by means of rendering scripts.

In other cases (e.g. Anaconda), it is possible that you need to make some variations to one background image that don’t want to appear on regular background images of the same resolution. In this case you need to create a new and specific background image for that art component. For example, if you need to produce the background image used by Anaconda (800x600) art works you create the file:

```
xcf/800x600-anaconda.xcf
```

to produce the background image:

```
img/800x600-anaconda-bg.png
```

which is loaded in:

```
svg/800x600-anaconda.svg
```

to produce the file:

`img/800x600-anaconda.png`

The `800x600-anaconda.png` file is used by all Anaconda art works sharing a common 800x600 screen resolution (e.g., Header, Progress, Splash, Firstboot, etc.). The Anaconda Prompt is indexed to 16 colors and 640x480 pixels so you need to create a 640x480 background image for it, and take the color limitation into account when designing it.

Background images without artistic motif are generally used as based to build the Background images that do contain the theme artistic motif.

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 structure 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 (Motifs) and theme visual structures (Models) are two different working lines. When you design an artistic motif for CentOS you concentrate on its visual style, and eventually, 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.

### 3.26.3 Usage

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.

### 3.26.4 See also

## 3.27 trunk/Identity/Themes/Motifs/Modern/Backgrounds/Img

### 3.27.1 Goals

- ...

### 3.27.2 Description

### 3.27.3 Usage

In this directory is where you store all background images (e.g., `.png`, `.jpg`, `.xpm`, etc.). This directory is required by `'centos-art'` command line interface.

### 3.27.4 See also



## 3.28 trunk/Identity/Themes/Motifs/Modern/Backgrounds/Tpl

### 3.28.1 Goals

- ...

### 3.28.2 Description

### 3.28.3 Usage

In this directory is where you store all the scalable vector graphics (e.g., .svg) files. This directory is required by ‘centos-art’ command line interface.

### 3.28.4 See also

## 3.29 trunk/Identity/Themes/Motifs/Modern/Backgrounds/Xcf

### 3.29.1 Goals

- ...

### 3.29.2 Description

- ...

### 3.29.3 Usage

In this directory is where you store the project files (e.g. .xcf) of Gimp. This directory is not required by ‘centos-art’ command line interface. If you can create a beautiful background images using scalable vector graphics only, then there is no need to use the ‘Xcf/’ directory to store background projects. Of course, you can merge both Gimp and Inkscape power to produce images based on them. In this last case you need the ‘Xcf/’ directory.

### 3.29.4 See also

## 3.30 trunk/Identity/Themes/Motifs/Modern/Distro/Anaconda/Progress

### 3.30.1 Goals

- ...

### 3.30.2 Description

### 3.30.3 Usage

To render Anaconda progress slide images using the *Modern* artistic motif design, the Default theme model, and available translation files (— **Removed**([pxref:trunk Translations Identity Themes Distro Anaconda Progress](#)) —); use the following commands:

```
cd /home/centos/artwork/trunk/Identity/Themes/Motifs/Modern/Distro/Anaconda/Progress/
centos-art render --identity
```

The above command will create the following structure:

```
trunk/Identity/Themes/Motifs/Modern/Distro/Anaconda/Progress
|-- 3
|   |-- en
|   |   |-- 01-welcome.png
|   |   |-- 02-donate.png
|   |   '-- 03-yum.png
```



```

|   '-- es
|       |-- 01-welcome.png
|       |-- 02-donate.png
|       '-- 03-yum.png
|-- 4
|   |-- en
|   |   |-- 01-welcome.png
|   |   |-- 02-donate.png
|   |   '-- 03-yum.png
|   '-- es
|       |-- 01-welcome.png
|       |-- 02-donate.png
|       '-- 03-yum.png
'-- 5
    |-- en
    |   |-- 01-welcome.png
    |   |-- 02-donate.png
    |   '-- 03-yum.png
    '-- es
        |-- 01-welcome.png
        |-- 02-donate.png
        '-- 03-yum.png

```

### 3.30.4 See also

## 3.31 trunk/Identity/Themes/Motifs/Modern/Palettes

### 3.31.1 Goals

- Organize palette files for Modern theme.

### 3.31.2 Description

### 3.31.3 Usage

Here is where graphic designers define theme palettes for color-limited art works. Theme palettes contain the color information that rendering functions need, in order to produce images with color limitations. Theme palettes contain the unique color information required by theme.

### 3.31.4 See also

## 3.32 trunk/Identity/Themes/Motifs/TreeFlower

### 3.32.1 Goals

- ...

### 3.32.2 Description

### 3.32.3 Usage

### 3.32.4 See also

## 3.33 trunk/Identity/Themes/Motifs/TreeFlower/Backgrounds

### 3.33.1 Goals

This section exists to organize backgrounds of *TreeFlower* artistic motif.

### 3.33.2 Description

#### 3.33.2.1 Desktop background

Once you have defined the vectorial artistic motif design, use the `centos-art.sh` script (as described in usage section below) to produce the png version of it. With the png version of your vectorial design do the following:

Open the png version with GIMP.

Save the png version as a project of GIMP inside `'trunk/Identity/Themes/Motifs/TreeFlower/Background'` directory, using the same name of your vectorial design but with the `'.xcf'` extension.

Now use GIMP to improve your design. Here you may add one layer for pattern, another for colors, and so on until you find yourself comfortable with your artwork. For example, the following layer distribution (from bottom to top) was used to build revision 285 of file `'1360x768.xcf'` using *TreeFlower* artistic motif at revision 241.

#### Layer 1: Background

The first thing we did with GIMP was to create a layer named `'Background'` to store the artistic motif (File > Open as layer). This layer is the lowest layer in the image. Later, we started to create layers one upon another to change the artistic motif visual style.

#### Layer 2: Shadow#1

This layer is above `'Background'` and contains a linear gradient from left (000000) to right (transparent) covering the whole image. This layer masks the artistic motif to avoid the effect of linear gradient. This layer is 100% of opacity.

#### Layer 3: Shadow#2

This layer is above `'Shadow#1'` and contains a linear gradient from left (000000) to right (transparent) covering just the 70% of the whole image approximately. This layer doesn't mask the artistic motif which make the left part of it fall into the dark of linear gradient. This layer is 100% of opacity.

#### Layer 4: Pattern (Paper)

This layer is above `'Shadow#2'` and contains the paper pattern shipped with GIMP 2.2. This layer doesn't mask the artistic motif so the pattern is applied over the whole image. This layer is set to 15% of opacity.

#### Layer 5: Pattern (Stripes)

This layer is above `'Pattern (Paper)'` and contains the stripes used over the artistic motif. This layer do masks the artistic motif so the stripes are only applied to it. This layer is set to 10% of opacity.

#### Layer 6: Shadow#3

This layer is above `'Pattern (Stripes)'` and contains a linear gradient from right (6600ff) to left (transparent). This layer masks the artistic motif so the linear gradient doesn't affect it. This layer is set to 15% of opacity.

#### Layer 7: Shadow#4

This layer is above `'Shadow#3'` and contains a linear gradient from left (000000) to right (transparent). This layer do masks the artistic motif so the linear gradient doesn't affect it. This layer is set to 10% of opacity.

**Layer 8: Color#1**

This layer is above ‘Shadow#4’ and is filled with orange (ffae00) color over the whole image. This layer is set to 10% of opacity.

**Layer 9: Color#2**

This layer is above ‘Color#1’ and is filled with blue (010a88) color over the whole image. This layer is set to 10% of opacity.

**Note** There is no definite combination. To get the appropriate visual design is a matter of constant testing and personal taste.

Finally, use ‘Save as copy ...’ option to export the final design. To export the final design use the same name of your vectorial design plus ‘-final.png’ extension.

You can repeat these steps to create images for other screen resolutions.

### 3.33.2.2 Anaconda Prompt (syslinux) background

When building syslinux backgrounds it is needed to take into account that the final image is reduced to 16 colors. In desktop background there is no color limitation but syslinux does have. The goal of this section is achieving a final syslinux background as close as possible to desktop backgrounds using 16 colors only.

Another point to consider is the foreground and background definition used by syslinux. The syslinux documentation says that the color set in position 0 is the background and color set in position 7 is the foreground. The final palette of color used by our background will match that specification. For great contrast we’ll use black as background and white as foreground. At this point we have black (000000) and white (ffffff) colors in our syslinux palette, which left us with 14 colors to play with.

Let’s begin with ‘Xcf/640x300.xcf’ layer distribution from bottom to top:

**Layer 1: Background**

This layer is the lowest layer in the image composition and contains the artistic motif image rendered for the same resolution (i.e., ‘Img/Png/640x300.png’). This layer is set to 100% of opacity.

**Layer 2: Pattern (Paper)**

This layer is placed above ‘Background’ layer and contains the paper pattern shipped with GIMP 2.2. This layer doesn’t mask the artistic motif. This layer is set to 30% of opacity.

**Layer 3: Pattern (Stripes)**

This layer is placed above ‘Pattern (Paper)’ layer and contains the stripes pattern shipped with GIMP 2.2. This layer does mask the artistic motif in order to apply the stripes over it only. The background is not affected by the stripes pattern just the artistic motif. This layer is set to 20% of opacity.

**Layer 4: Shadow#1**

This layer is placed above ‘Pattern (Stripes)’ layer and fills the entire layer area with violet (6600ff) color. This layer do mask the artistic motif in order to applied the violet color to the background area outside the artistic motif only. This layer is set to 15% of opacity.

**Layer 5: Color#1**

This layer is above ‘Shadow#1’ and is filled with orange (ffae00) color to cover the whole image. This layer is set to 10% of opacity.

**Layer 6: Color#2**

This layer is above ‘Color#1’ and is filled with blue (010a88) color to cover the whole image. This layer is set to 10% of opacity.

**Layer 7: Shadow#2**

This layer is above ‘Color#1’ and contains a linear gradient from left (000000) to right (transparent) covering 70% of the image approximately.

At this point we have the composition and should look like the desktop backgrounds. Compared with desktop backgrounds there are some differences in opacity. This is because in our testings the final color information found with this composition produces an acceptable 16 color image. Of course this is something we haven’t seen yet.

To define the color information of our current composition, save the syslinux background composition we’ve done using ‘File > Save as Copy ...’ option in the following location:

```
trunk/Identity/Themes/Motifs/TreeFlower/Backgrounds/Img/Png/640x300-final.png
```

Now, create the final png version of syslinux backgrounds using the following command:

```
centos-art render --entry=trunk/Identity/Themes/Motifs/TreeFlower/Distro/Anaconda/Prompt
```

This command will create syslinux-splash final images for all major releases of CentOS distribution the repository has been configured to. The important files here are ‘syslinux-splash.png’, other files may contain the wrong information because we haven’t defined yet the correct color information to use.

Open one ‘syslinux-splash.png’ file with GIMP and use the ‘Image > Mode > Indexed’ to reduce image colors up to 16 colors, using the ‘Generate optimum palette’ feature of GIMP. If the image looks acceptable after reducing colors, use the ‘Palettes’ menu (Ctrl+P) of GIMP to import a new palette from file and name it ‘CentOS-TreeFlower-Syslinux’. Once you’ve saved the palette, the color information is stored at:

```
~/gimp-2.2/palettes/CentOS-TreeFlower-Syslinux.gpl
```

You need to edit ‘CentOS-TreeFlower-Syslinux.gpl’ file in order to set the appropriate order of colors. Remember black (000000) in position 0, and white (ffffff) in position 7. Other positions are irrelevant. When editing this file you may find that color reduction did not set black and white colors to their respective values exactly. Change that manually. For example, consider the following palette:

```
GIMP Palette
Name: CentOS-TreeFlower-Syslinux
Columns: 16
#
 0  0  0      Background (black)
23 20 35     Untitled
34 25 48     Untitled
37 35 60     Untitled
47 36 68     Untitled
37 54 86     Untitled
60 48 90     Untitled
255 255 255  Foreground (white)
66 54 99     Untitled
74 61 98     Untitled
49 78 126    Untitled
43 87 151    Untitled
92 89 95     Untitled
54 104 183   Untitled
158 153 156  Untitled
201 196 195  Untitled
```

Update the ‘Palettes’ menu to get the new color positions from the file you just edited and open the palette with double click.

Update the 'syslinux.gpl' file copying the following file:

```
~/gimp-2.2/palettes/CentOS-TreeFlower-Syslinux.gpl
```

to

```
trunk/Identity/Themes/Motifs/TreeFlower/Colors/syslinux.gpl
```

With the 'CentOS-TreeFlower-Syslinux' palette opened in the 'Palette Editor', open (Ctrl+O) the following file:

```
trunk/Identity/Themes/Motifs/TreeFlower/Colors/syslinux.ppm
```

and replace its color information with that one in 'CentOS-TreeFlower-Syslinux' palette. When you are replacing color information inside 'syslinux.ppm', remember to keep the order of colors just as they are in the 'CentOS-TreeFlower-Palette' palette.

The 'syslinux.ppm' file is 16 pixels width and 1 pixel height, so you probably need to zoom it a bit to set the color information in their place when using the pen tool with the brush 'Circle (01) (1 x 1)'.

Once you've updated the 'syslinux.ppm' file, it is time to update the following file:

```
trunk/Identity/Themes/Motifs/TreeFlower/Colors/syslinux.hex
```

The 'syslinux.hex' file contains the color information in hexadecimal notation. The color information in hexadecimal notation is required by `ppmtolss16` command. The `ppmtolss16` command produces the final LSS16 image format that is used by syslinux program inside CentOS distribution.

The color information inside 'syslinux.hex' must match the one in 'syslinux.ppm' and 'syslinux.gpl'. For example, based on 'CentOS-TreeFlower-Syslinux' palette of colors above, consider the following 'syslinux.hex' file:

```
#000000=0
#171423=1
#221930=2
#25233c=3
#2f2444=4
#253656=5
#3c305a=6
#ffffff=7
#423663=8
#4a3d62=9
#314e7e=10
#2b5797=11
#5c595f=12
#3668b7=13
#9e999c=14
#c9c4c3=15
```

### 3.33.2.3 Grub background

### 3.33.3 Usage

- ...

### 3.33.4 See also

## 3.34 trunk/Identity/Widgets

### 3.34.1 Goals

- ...

### 3.34.2 Description

### 3.34.3 Usage

### 3.34.4 See also

## 3.35 trunk/Locales

### 3.35.1 Goals

The ‘trunk/Translations’ directory exists to:

- Organize translation files.
- Organize translation templates used to produce translation files.

### 3.35.2 Description

When you create artwork for CentOS distribution you find that some artworks need to be created for different major releases of CentOS distribution and inside each major release that needs to be created for different locales. To get an approximate idea of how many files we are talking about, consider the following approximate statistic:

- Inside CentOS distribution, there are around 30 images to rebrand.<sup>2</sup>
- There are near to four major releases of CentOS distribution to rebrand in parallel development.<sup>3</sup>
- Each CentOS distribution in parallel development supports more than two hundreds locales.<sup>4</sup>

In order to alleviate maintenance of artwork production for such environment, we divided artwork production in three production lines:

1. See [Section 3.16 \[trunk Identity Themes Models\]](#), page 13, to define artworks characteristics (e.g., dimensions, position on the screen, etc.).
2. See [Section 3.23 \[trunk Identity Themes Motifs\]](#), page 16, to define artworks visual styles (e.g., the look and feel).
3. Translations, to define which major releases and locales artworks are produced for.

Inside CentOS Artwork Repository, the artworks’ translation production line is stored under ‘trunk/Translations’ directory.

Inside ‘trunk/Translations’ directory, we use “translation entries” to organize artworks’ “translation files” and artworks’ “translation templates”.

#### 3.35.2.1 Translation Entries

Translation entries exist for each artwork you want to produce. Translation entries can be empty directories, or directories containing translation files and translation templates.

When translation entries are empty directories, the identity entry is used as reference to create file names and directories layout for rendered files. In this case, the `centos-art` script takes one design template and outputs one non-translated file for each design template available. This configuration is mainly used to produce non-translatable artworks like themes’ backgrounds.

---

<sup>2</sup> This number is an approximate value and may change. It is mainly based on CentOS 5 rebranding experience.

<sup>3</sup> This value was taken from CentOS release schema.

<sup>4</sup> This value was taken from the output of `locale -a` command.

When translation entries contain translation files, the translation entry implements the CentOS release schema and is used as reference to create file names and directories layout for translated artworks. In this case, the `centos-art` script applies one translation file to one design template to create one translated instance which is used to output one translated file. When the translated file is rendered, the `centos-art` script remove the previous instance and takes the next file in the list of translation files to repate the whole process once again, and so on for all files in the list. This configuration is mainly used to produce translatable artworks like Anaconda progress slide images.

To find out correspondence between translation entries and identity entries, you need to look the path of both translation entries and identity entries. For example, if you are using the *Modern* artistic motif, the identity entry for Anaconda progress artwork is:

```
trunk/Identity/Themes/Motifs/Modern/Distro/Anaconda/Progress
```

and its translation entry is:

```
trunk/Translations/Identity/Themes/Distro/Anaconda/Progress
```

Note how the ‘Translations/’ directory prefixes ‘Identity/’ directory, also how static values (e.g., Identity, Themes, Distro, etc.) in the identity entry path remain in translation entry path, and how variable values like theme names (e.g., *Modern*) are stript out from translation entry path. The same convention can be applied to other identity entries in order to determine their translation entries, or to other translation entries to determine their identity entries.

**Note** Translation entries related to identity entries under ‘trunk/Identity/Themes/Motifs’ do not use ‘Motifs/’ in the path. We’ve done this because ‘trunk/Identity/Themes/Models’ structure, the other structure under ‘trunk/Identity/Themes’, doesn’t require translation paths so far. So in the sake of saving characters space when building translation entries for ‘trunk/Identity/Themes/Motifs’ structure, we organize Motifs translation entries under ‘trunk/Translations/Identity/Themes/’ directly.

If for some reason ‘trunk/Identity/Themes/Models’ structure requires translation entries, we need to re-oraganize the current directory structure accordingly.

Translation entries, as described above, can be re-used by similar identity entries. For example the following identity entries:

```
trunk/Identity/Themes/Motifs/Modern/Distro/Anaconda/Progress/  
trunk/Identity/Themes/Motifs/TreeFlower/Distro/Anaconda/Progress/  
trunk/Identity/Themes/Motifs/Mettle/Distro/Anaconda/Progress/
```

are all valid identity entries able to re-use translation files inside Anaconda progress translation entry (the one shown in our example above). This way, you can create several identity entries and maintain just one translation entry for all of them. Once you change the translation files inside the common translation entry, changes inside identity entries will take effect inside the next you render them.

Trying to make things plain and simple: inside CentOS Artwork Repository, graphic designers can concentrate their efforts in artworks look and feel (the identity entries), and translators in artworks translations (the translation entries).

### 3.35.2.2 Translation Markers

Translation markers are used in “Theme Model Designs” and “Translation Files” as replacement patterns to commit content translation. When you are rendering content using `centos-art` script inside ‘trunk/Identity’ structure, artistic motifs and translation files are applied to model designs to produce translated content as result. In order to have the appropriate translation in content rendered, markers defintion in translation files should match markers in model designs exactly.



Translation markers can be whatever text you want, but as convention we use the following to represent releases of CentOS distribution:

`'=MINOR_RELEASE='`

Replace with minor release of CentOS distribution. In the schema M.N, the minor release is represented by the N letter.

`'=MAJOR_RELEASE='`

Replace with major release of CentOS distribution. In the schema M.N, the major release is represented by the M letter.

`'=RELEASE='`

Replace the full release of CentOS distribution. It is `'=MAJOR_RELEASE=. =MINOR_RELEASE='` basically.

Specific translation markers conventions are described inside specific translation entries. Read translation entries documentation to know more about supported translation markers.

Translation markers standardization creates a common point of reference for translators and graphic designers. To have translation markers well defined makes possible that translators and graphic designers can work together but independently one another.

### 3.35.2.3 Translation Files

Translation files are text files with `sed` commands inside, replacement commands mainly. As convention, translation file names end in `'.sed'`. Translation files are used by `centos-art` script to produce translated artworks for specific major releases of CentOS Distribution. There are common translation files, specific translation, and template translation files.

For example, the Firstboot artwork of CentOS distribution uses the images `'splash-small.png'` and `'firstboot-left.png'` as based to control its visual style. The `'splash-small.png'` image contains, in its graphic design, the release number information of CentOS distribution. So the `'splash-small.png'` is release-specific. In the other hand, the `'firstboot-left.png'` doesn't contain release number information. So the `'firstboot-left.png'` is not release-specific.

If we want to produce Firstboot artwork for different major releases of CentOS distribution, using a monolithic visual identity, all Firstboot images should have the same visual style and, at the same time, the release-specific information in the release-specific images.

**Note** The monolithic visual identity is implemented using theme models (see [Section 3.16 \[trunk Identity Themes Models\]](#), page 13) and artistic motifs (see [Section 3.23 \[trunk Identity Themes Motifs\]](#), page 16).

Assuming that both theme models and theme motifs are ready for using, the initial translation entry to produce Firstboot artworks would look like the following:

```
trunk/Translations/Identity/Themes/Distro/BootUp/Firstboot/
|-- Tpl
|   '-- splash-small.sed
'-- firstboot-left.sed
```

With the translation entry above, `centos-art` command is able to produce the image `'firstboot-left.png'` only. To produce `'splash-small.png'` images for major releases (e.g., 3, 4, 5, and 6) of CentOS distribution we need to produce the release-specific translation files using the `centos-art` script as following:

```
centos-art render --entry=/home/centos/artwork/trunk/Translations/Identity/Themes/BootUp/
```

The above command produces the following translation entry:

```
trunk/Translations/Identity/Themes/Distro/BootUp/Firstboot/
|-- 3
```



```

|   '-- splash-small.sed
|-- 4
|   '-- splash-small.sed
|-- 5
|   '-- splash-small.sed
|-- 6
|   '-- splash-small.sed
|-- Tpl
|   '-- splash-small.sed
'-- firstboot-left.sed

```

At this point `centos-art` is able to produce the Firstboot artwork images for major releases of CentOS distribution. To add new release-specific translation files, run the translation rendering command with the release number you want to produce translation files for in the `'--filter=release-number'` argument.

### 3.35.2.4 Template Translation Files

Template translation files are translation files stored inside translation template directory. Template translation files are used by `centos-art` script to produce specific translation files only. Template translation files may be empty or contain `sed` replacement commands. If template translation files are empty files, the final specific translation file built from it contains release-specific replacement commands only. For example, see the following translation entry:

```

trunk/Translations/Identity/Themes/Distro/BootUp/Firstboot/
|-- 3
|   '-- splash-small.sed
|-- 4
|   '-- splash-small.sed
|-- 5
|   '-- splash-small.sed
|-- 6
|   '-- splash-small.sed
|-- Tpl
|   '-- splash-small.sed    <-- template translation file.
'-- firstboot-left.sed

```

In the above example, the `'splash-small.sed'` file is a template translation file and looks like:

```

# -----
# $Id: splash-small.sed 94 2010-09-18 10:59:42Z al $
# -----

```

In the above template translation file there are three comment lines, but when you render it, the `centos-art` adds the release-specific replacement commands. In our Firstboot example, after rendering Firstboot translation entry, the `'splash-small.sed'` translation file specific to CentOS 5, looks like the following:

```

# Warning: Do not modify this file directly. This file is created
# automatically using 'centos-art' command line interface. Any change
# you do in this file will be lost the next time you update
# translation files using 'centos-art' command line interface. If you
# want to improve the content of this translation file, improve its
# template file instead and run the 'centos-art' command line
# interface later to propagate your changes.
# -----

```

```
# $Id: splash-small.sed 94 2010-09-18 10:59:42Z al $
# -----
```

```
# Release number information.
s!=RELEASE=!MAJOR_RELEASE=.MINOR_RELEASE=!g
s!=MINOR_RELEASE=!0!g
s!=MAJOR_RELEASE=!5!g
```

If template translation files are not empty, replacement commands inside template translation files are preserved inside release-specific translation files. For example, consider the English template translation file of Anaconda progress welcome slide. The translation template directory structure looks like the following:

```
trunk/Translations/Identity/Themes/Distro/Anaconda/Progress/
'-- Tpl
  '-- en
    '-- 01-welcome.sed
```

and if we render translation files for CentOS 4 and CentOS 5 major releases, the translation entry would look like the following:

```
trunk/Translations/Identity/Themes/Distro/Anaconda/Progress/
|-- 4
|   '-- en
|     '-- 01-welcome.sed
|-- 5
|   '-- en
|     '-- 01-welcome.sed
'-- Tpl
  '-- en
    '-- 01-welcome.sed
```

**Note** Release-specific translation directories preserve template translation directory structure and file names.

In the example above, the template translation file looks like the following:

```
# -----
# $Id: 01-welcome.sed 94 2010-09-18 10:59:42Z al $
# -----
s/=TITLE=/Welcome to CentOS =MAJOR_RELEASE= !/
s/=TEXT1=/Thank you for installing CentOS =MAJOR_RELEASE=./
s/=TEXT2=/CentOS is an enterprise-class Linux Distribution derived from sources freely pr
s/=TEXT3=/CentOS conforms fully with the upstream vendors redistribution policy and aims
s/=TEXT4=//
s/=TEXT5=//
s/=TEXT6=//
s!=URL=!http://www.centos.org/!
```

and, after render the translation entry, specific translation files look like the following:

```
# Warning: Do not modify this file directly. This file is created
# automatically using 'centos-art' command line interface. Any change
# you do in this file will be lost the next time you update
# translation files using 'centos-art' command line interface. If you
# want to improve the content of this translation file, improve its
# template file instead and run the 'centos-art' command line
# interface later to propagate your changes.
# -----
```

```
# $Id: 01-welcome.sed 94 2010-09-18 10:59:42Z al $
# -----

s/=TITLE=/Welcome to CentOS =MAJOR_RELEASE= !/
s/=TEXT1=/Thank you for installing CentOS =MAJOR_RELEASE=./
s/=TEXT2=/CentOS is an enterprise-class Linux Distribution derived from sources freely pr
s/=TEXT3=/CentOS conforms fully with the upstream vendors redistribution policy and aims
s/=TEXT4=//
s/=TEXT5=//
s/=TEXT6=//
s!=URL=!http://www.centos.org/!

# Release number information.
s!=RELEASE=!MAJOR_RELEASE=.MINOR_RELEASE=!g
s!=MINOR_RELEASE=!0!g
s!=MAJOR_RELEASE=!5!g
```

In the example above, relevant lines begin with the ‘s’ word followed by a separation character (e.g., ‘/’, ‘!’, etc.). These lines have the following format:

```
s/REGEXP/REPLACEMENT/FLAGS
```

The ‘/’ characters may be uniformly replaced by any other single character within any given s command. The ‘/’ character (or whatever other character is used in its stead) can appear in the REGEXP or REPLACEMENT only if it is preceded by a ‘\’ character.

The s command is probably the most important in sed and has a lot of different options. Its basic concept is simple: the s command attempts to match the pattern space against the supplied REGEXP; if the match is successful, then that portion of the pattern space which was matched is replaced with REPLACEMENT.

In the context of our translation files, the REGEXP is where you define translation markers and REPLACEMENT where you define the translation text you want to have after artworks rendering. Sometimes we use the FLAG component with the ‘g’ command to apply the replacements globally.

**Tip** More information about how to use sed replacement commands and flags is available in sed documentation manual. To read sed documentation manual type the following command:

```
info sed
```

Inside translation files, you can use translation markers not only inside the REGEXP but in the REPLACEMENT too. In order for this configuration to work, the REPLACEMENT of translation markers needs to be define *after* its definition. For example, see in the release-specific translation file above, how the ‘s!=MAJOR\_RELEASE=!5!g’ replacement command is defined *after* ‘=MAJOR\_RELEASE=’ translation marker definition in the REPLACEMENT of ‘=TITLE=’ translation marker replacement command.

### 3.35.2.5 Common Translation Files

Common translation files contain common translations or no translation at all for their related artworks. They are in the root directory of the translation entry. Common translation files create common artworks for all major releases of CentOS Distribution.

Translation entries, with common translation files inside, look like the following:

```
trunk/Translations/Identity/Themes/Distro/BootUp/Firstboot/
|-- 3
|   '-- splash-small.sed
```

```

|-- 4
|   '-- splash-small.sed
|-- 5
|   '-- splash-small.sed
|-- 6
|   '-- splash-small.sed
|-- Tpl
|   '-- splash-small.sed
'-- firstboot-left.sed      <-- common translation file.

```

### 3.35.2.6 Specific Translation Files

Specific translation files contain specific translations for their related artworks. Specific translation files are not in the root directory of the translation entry, but inside directories which describe the type of translation they are doing. Specific translation files are produced automatically using the `centos-art` script.

```

trunk/Translations/Identity/Themes/Distro/BootUp/Firstboot/
|-- 3
|   '-- splash-small.sed      <-- CentOS 3 specific translation file.
|-- 4
|   '-- splash-small.sed      <-- CentOS 4 specific translation file.
|-- 5
|   '-- splash-small.sed      <-- CentOS 5 specific translation file.
|-- 6
|   '-- splash-small.sed      <-- CentOS 6 specific translation file.
|-- Tpl
|   '-- splash-small.sed
'-- firstboot-left.sed

```

### 3.35.2.7 Translation Rendering

When rendering translations, the `centos-art` script checks the translation entry to verify that it has a translation template directory inside. The translation template directory (`'Tpl/'`) contains common translation files used to build release-specific translation files. If the translation template directory doesn't exist inside the translation entry the translation rendering fails. In this case the `centos-art` script outputs a message and quits script execution.

### 3.35.2.8 Translation (Pre-)Rendering Configuration Scripts

When the `centos-art` script finds a translation template directory inside translation entry, it looks for translations pre-rendering configuration scripts for that translation entry. Translation pre-rendering configuration scripts let you extend translation default functionality (described below).

Translation pre-rendering configuration scripts are stored under `'trunk/Scripts'` directory, specifically under the appropriate language implementation. If you are using `centos-art` Bash implementation, the translation pre-rendering scripts are store in the `'trunk/Scripts/Bash/Config'` location; if you are using `centos-art` Python implementation, then translation pre-rendering scripts are stored in the `'trunk/Scripts/Python/Config'` location, and so on for other implementations.

Bash translation pre-rendering configuration scripts look like the following:

```

#!/bin/bash
#
# render_loadConfig.sh -- brief description here.
#

```

```

# Copyright (C) YEAR YOURNAME
#
# 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: render_loadConfig.sh 94 2010-09-18 10:59:42Z al $
# -----

function render_loadConfig {
...
}

```

Translation pre-rendering scripts are function scripts loaded and executed when rendering a translation entry. Translation pre-rendering scripts are loaded using the translation entry being rendered as reference. For example, suppose you are using the `centos-art` Bash implementation, and you are rendering translations for CentOS brands, in this situation the translation entry would be:

```
trunk/Translations/Identity/Brands
```

and the entry inside the translation pre-rendering configuration structure would be:

```
trunk/Scripts/Bash/Config/Identity/Brands
```

Once the `centos-art` script detects that translation pre-rendering configuration directory exists, the `centos-art` script looks for the translation pre-rendering configuration file. If the translation pre-rendering configuration file exists, it is loaded and executed. Once the translation pre-rendering configuration file has been executed the translation rendering process is over, and so the script execution.

**Note** Translation pre-rendering configuration files have the following form:

```
render.conf.extension
```

where ‘`extension`’ refers the programming language implementation you are using.

For example, ‘`sh`’ for Bash, ‘`py`’ for Python, ‘`pl`’ for Perl, and so on for other implementations.

As we are using Bash implementation to describe the translation pre-rendering configuration example, the translation pre-rendering configuration file that `centos-art` looks for, inside the above translation pre-rendering configuration directory, is ‘`render.conf.sh`’.

### 3.35.2.9 Translation Rendering Default Functionality

In the other hand, if the translation pre-rendering configuration file doesn’t exist, or it isn’t written as function script, the `centos-art` script ignore translation pre-rendering configuration functionality and passes to render translation using default functionality instead.

The translation rendering default functionality takes template translation directory structure, duplicates it for each release number specified in the ‘`--filter=release-number`’ argument and produces release-specific directories. As part of template translation duplication process take place, the `centos-art` script adds release-specific replacement commands for each specific translation file inside release-specific directories. As result, specific translation files, inside release-specific directories, contain template translation replacement commands *plus*, release-specific replacement commands.

**Note** Release-specific replacement commands are standardized inside `centos-art` script using predefined release translation markers. Release translation markers are described in the translation marker section (see [\[trunk:Translations:TranslationMarkers\]](#), page 29).

### 3.35.3 Usage

```
‘centos-art render --entry=’path/to/dir’
```

When ‘`path/to/dir`’ refers one directory under ‘`trunk/Translations`’, this command overwrites available translation files using translation templates.

```
‘centos-art render --entry=’path/to/dir’ --filter=’pattern’
```

When ‘`path/to/dir`’ refers one directory under ‘`trunk/Translations`’, this command renders release-specific translation files as you specify in the ‘`--filter=pattern`’ argument. In this case, ‘`pattern`’ not a regular expression but an number (e.g., ‘`5`’) or a list of numbers separated by commas (e.g., ‘`3,4,5,6`’) that specify the major release of CentOS distribution you want to render translations for.

### 3.35.4 See also

## 3.36 trunk/Locales/Identity

### 3.36.1 Goals

- ...

### 3.36.2 Description

- ...

### 3.36.3 Usage

- ...

### 3.36.4 See also

## 3.37 trunk/Locales/Identity/Brands

### 3.37.1 Goals

- Organize brands’ translation files.

### 3.37.2 Description

Translation files, inside ‘`trunk/Translations/Identity/Brands`’ translation entry, don’t use default rendering translation functionality, they use the following translation pre-rendering configuration file instead:

```
/home/centos/artwork/trunk/Translation/Identity/Brands/render.conf.sh
```

Inside `'trunk/Translations/Identity/Brands'` translation entry, translation files are symbolic links pointing to the common template translation structure, inside the translation template (`'Tpl/'`) directory.

Inside `'trunk/Translations/Identity/Brands'` translation entry, translation files are created using identity design templates as reference. The translation pre-rendering script creates a translation structure where the translation template (`'Tpl/'`) directory structure applies to each single design template available.

For example, if the brands' translation template (`'Tpl/'`) directory has 30 translation files, and there are 20 design templates; the brands' translation pre-rendering script creates a translation structure of symbolic links where the 30 translation files apply the 20 design templates one by one, producing 600 translation symbolic links as result. At this point, when rendering identity, the `centos-art` script considers translation symbolic links as translation files.

Translation file names, inside brands' translation template (`'Tpl'`) directory have special meaning:

### 3.37.2.1 Conventional file names

Conventional file names look like `'blue.sed'`, `'2c-a.sed'`, etc. Replacement commands inside translation file are applied to design templates and translation file names are used as final image name. The image dimensions use the same dimensions that design template has.

### 3.37.2.2 Numeric file names

Numeric file names look like `'300.sed'`, `'200.sed'`, etc. Replacements commands inside translation files are applied to design templates, and translation file names are used as final image name. The final image is saved using an specific `'width'` defined by the number part of the translation file name. The image `'height'` is automatically scaled based on the previous `'width'` definition to maintain the designing ratio.

For example, if your design template has 400x200 pixels of dimension, and you apply a translation file named `'300.sed'` to it, the final image you get as result will have 300x100 pixels of dimension. The same is true if you use higher numbers like `'1024.sed'`, `'2048.sed'`, etc. In these cases you have bigger images proportionally.

As we are using scalable vector graphics to design identity templates, the image size you produce is not limited in size. You can use one design template produced in 400x200 pixels to produce larger or shorter PNG images using numeric translation files as described above.

### 3.37.2.3 Translation markers

Inside `'trunk/Translations/Identity/Brands/'`, translation files combine the following translation markers:

`'#000000'`

Specify which color to use when rendering brand images.

**Note** As translation files inside `'trunk/Translations/Identity/Brands'` are symbolic links that point to template translation files, translation markers are defined inside template translation files.

### 3.37.3 Usage

To render brands' translation files, use the following command:

```
centos-art render --translation=/home/centos/artwork/trunk/Translations/Identity/Brands
```

### 3.37.4 See also

## 3.38 trunk/Locales/Identity/Brands/Tpl

### 3.38.1 Goals

### 3.38.2 Description

### 3.38.3 Usage

### 3.38.4 See also

## 3.39 trunk/Locales/Identity/Fonts

### 3.39.1 Goals

This section exists to organize fonts translation files.

### 3.39.2 Description

Translation files, inside ‘trunk/Translations/Fonts’, have the following structure:

```
s!font-family:Denmark!font-family:DejaVu LGC Sans!  
s!font-weight:normal!font-weight:bold!  
s!font-style:normal!font-style:italic!
```

Inside ‘trunk/Translations/Fonts’, there is one translation file for each font preview image you want to produce. This way, we create one translation file for each font-family we use somewhere inside CentOS visual identity.

**Important** Do not create translation files for font-families not used somewhere inside CentOS visual identity. The identity of font entry (see [Section 3.4 \[trunk Identity Fonts\], page 5](#)) is used as reference when someone needs to know which font-families are allowed to use inside CentOS visual identity.

#### 3.39.2.1 Translation Markers

Inside ‘trunk/Translations/Identity/Fonts’, translation files combine the following translation markers:

```
‘font-family:Denmark’
```

Specify which font family to use when rendering font preview images.

```
‘font-weight:normal’
```

Specify which font weight to use when rendering font preview images.

```
‘font-style:normal’
```

Specify which font style to use when rendering font preview images.

### 3.39.3 Usage

Inside ‘trunk/Translations/Fonts’ you use your favorite text editor to create translation files. Inside ‘trunk/Translations/Fonts’ there is not translation template directory (‘Tpl/’), nor translation rendering using `centos-art` script. For example, to create the ‘`dejavu_lgc_sans-boldoblique.sed`’ translation file using `vim` editor, type the following command:

```
vim /home/centos/artwork/trunk/Translations/Fonts/dejavu_lgc_sans-boldoblique.sed
```

### 3.39.4 See also



### **3.40 trunk/Locales/Identity/Models**

#### **3.40.1 Goals**

#### **3.40.2 Description**

#### **3.40.3 Usage**

#### **3.40.4 See also**

### **3.41 trunk/Locales/Identity/Release**

#### **3.41.1 Goals**

#### **3.41.2 Description**

#### **3.41.3 Usage**

#### **3.41.4 See also**

### **3.42 trunk/Locales/Identity/Themes**

#### **3.42.1 Goals**

#### **3.42.2 Description**

#### **3.42.3 Usage**

#### **3.42.4 See also**

### **3.43 trunk/Locales/Identity/Themes/Backgrounds**

#### **3.43.1 Goals**

- ...

#### **3.43.2 Description**

- ...

#### **3.43.3 Usage**

- ...

#### **3.43.4 See also**

### **3.44 trunk/Locales/Identity/Themes/Distro/Anaconda/Progress**

#### **3.44.1 Goals**

- Organize Anaconda progress translation templates.
- Organize Anaconda progress translation files in several languages and major releases of CentOS distribution.

### 3.44.2 Description

Use the following command to produce translation files based:

```
trunk/Translations/Identity/Themes/Distro/Anaconda/Progress
'-- Tpl
  |-- en
  |   |-- 01-welcome.sed
  |   |-- 02-donate.sed
  |   '-- 03-yum.sed
  '-- es
     |-- 01-welcome.sed
     |-- 02-donate.sed
     '-- 03-yum.sed
```

In order to produce the slide images in PNG format we need to have the translation files first. So we use the following commands to create translation files for CentOS 3, 4, and 5 major releases:

```
centos-art render --translation --filter='3,4,5'
```

The above commands will produce the following translation structure:

```
trunk/Translations/Identity/Themes/Distro/Anaconda/Progress
|-- 3
|   |-- en
|   |   |-- 01-welcome.sed
|   |   |-- 02-donate.sed
|   |   '-- 03-yum.sed
|   '-- es
|       |-- 01-welcome.sed
|       |-- 02-donate.sed
|       '-- 03-yum.sed
|-- 4
|   |-- en
|   |   |-- 01-welcome.sed
|   |   |-- 02-donate.sed
|   |   '-- 03-yum.sed
|   '-- es
|       |-- 01-welcome.sed
|       |-- 02-donate.sed
|       '-- 03-yum.sed
|-- 5
|   |-- en
|   |   |-- 01-welcome.sed
|   |   |-- 02-donate.sed
|   |   '-- 03-yum.sed
|   '-- es
|       |-- 01-welcome.sed
|       |-- 02-donate.sed
|       '-- 03-yum.sed
'-- Tpl
   |-- en
   |   |-- 01-welcome.sed
   |   |-- 02-donate.sed
   |   '-- 03-yum.sed
```

```

    '-- es
    |-- 01-welcome.sed
    |-- 02-donate.sed
    '-- 03-yum.sed

```

At this point we have all the translation files we need to produce Anaconda progress welcome, donate and yum slides images; in English and Spanish languages; for CentOS 3, CentOS 4, and CentOS 5. That is, a sum of 18 images around.

Now, with translation files in place, let's move to 'trunk/Identity' structure and render them.

- See [Section 3.30 \[trunk Identity Themes Motifs Modern Distro Anaconda Progress\]](#), page 22.

### 3.44.3 Usage

Translation rendering is described in 'trunk/Translations' documentation entry ([Removed\(pxref:trunk Translations\)](#) —).

### 3.44.4 See also

## 3.45 trunk/Locales/Identity/Widgets

### 3.45.1 Goals

- ...

### 3.45.2 Description

- ...

### 3.45.3 Usage

- ...

### 3.45.4 See also

## 3.46 trunk/Manuals

### 3.46.1 Goals

- ...

### 3.46.2 Description

- ...

### 3.46.3 Usage

- ...

### 3.46.4 See also

## 3.47 trunk/Scripts

### 3.47.1 Goals

The 'trunk/Scripts' directory exists to:

- Organize the "trunk" development line of automation scripts by programming language.

### 3.47.2 Description

- ...

### 3.47.3 Usage

- ...

### 3.47.4 See also

## 3.48 trunk/Scripts/Bash

### 3.48.1 Goals

The `'trunk/Scripts/Bash'` directory exists to organize the trunk development line of `'centos-art.sh'` automation script. The `'centos-art.sh'` script standardizes frequent tasks inside your working copy of CentOS Artwork Repository.

### 3.48.2 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 tries 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 (see [Section 3.58 \[trunk Scripts Bash Functions Verify\]](#), page 83), 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 retrieve command-line arguments and define some default values that may be used later by specific function scripts (see [Section 3.49 \[trunk Scripts Bash Functions\]](#), page 44).

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

```
centos-art arg1 --arg2=val2 --arg3=val3
```

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 resets, specifically, the action inside the functionality you want to perform. The `'--arg3'` and on, are optional.

Once command-line arguments have been retrieved, the `'centos-art.sh'` script loads specific functionalities using the `'cli_getFunctions.sh'` function script. Only one specific functional-





### 3.49.2 Description

The specific functions of ‘centos-art.sh’ script are designed with “Software Toolbox” philosophy (see [\[Toolbox introduction\]](#), page [\[Toolbox introduction\]](#)) 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](mailto: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 convention, 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 decide 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 convention 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 organized 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 (see [Section 3.52 \[trunk Scripts Bash Functions Manual\]](#), page 61) 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 (see [Section 3.51 \[trunk Scripts Bash Functions Locale\]](#), page 60) 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 retrieve 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?

### 3.49.3 Usage

#### 3.49.3.1 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]
- FUNCSRIPT** [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." --
```

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 retrieve 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 convention, 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.

### 3.49.3.2 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]

Synchronize 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 3.3: 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 environment, `cli_getArguments` uses regular expression to retrieve 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 retrieve action related variables) and redefine the arguments (*ARGUMENTS*) global variable with the new positional parameters information.

**cli\_getFunctions** [Function]

Initialize functionalities 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 functionality 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 convention 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 retrieve 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 immediately 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 specified 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 absolutely 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 concatenation 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 conventions 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 sanitize.

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 *NAMES*.

'f'

'regular-file'

Sanitate regular file *NAMES*.

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

Use `cli_getRepoName` when you need to change file name conventions inside 'centos-art.sh' script.

When we change file name conventions 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 conventions in the repository (see [Section 3.53 \[trunk Scripts Bash Functions Path\]](#), page 62, 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. See [Section 3.51 \[trunk Scripts Bash Functions Locale\]](#), page 60, 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
```

**3.49.3.3 Specific functions**

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

**3.49.4 See also****3.50 trunk/Scripts/Bash/Functions/Html****3.50.1 Goals**

- ...

**3.50.2 Description**

- ...

**3.50.3 Usage**

- ...

**3.50.4 See also****3.51 trunk/Scripts/Bash/Functions/Locale****3.51.1 Goals**

- ...

**3.51.2 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 sorted 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 `CLLLANG` variable describes 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 `CLLLANG` 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 export 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 running `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’
- ...

### 3.51.3 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.

### 3.51.4 See also

## 3.52 trunk/Scripts/Bash/Functions/Manual

### 3.52.1 Goals

- ...

### 3.52.2 Description

- ...



### 3.52.3 Usage

- ...

### 3.52.4 See also

## 3.53 trunk/Scripts/Bash/Functions/Path

### 3.53.1 Goals

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

### 3.53.2 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."*

#### 3.53.2.1 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 convention, 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 [Chapter 3 \[trunk\], page 3](#)) 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 [Chapter 1 \[branches\], page 1](#)) organizes parallel development lines to `'trunk/'` directory. The `'branches/'` directory is used to set points in time where development 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 [Chapter 2 \[tags\], page 2](#)) 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.

#### 3.53.2.2 Repository name conventions

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

Repository name conventions are applied to files and directories inside the repository layout. As convention, 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 conventions are implemented inside the `cli_getRepoName` function of ‘centos-art.sh’ script. With `cli_getRepoName` function we reduce the amount of commands and conventions to remember, concentrating them in just one single place to look for fixes and improvements.

### 3.53.2.3 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 tuning it up.

Once the branch has been tuned 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.

**Convention** 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 committed 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.

### 3.53.2.4 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 retrieve 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.

### 3.53.2.5 Synchronizing 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 don'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 synchronize 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.

### 3.53.2.6 What is the right place to store it?

Occasionally, 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 3.1 \[trunk Identity\], page 3](#)) 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 similarly as we did above, just change the location we used above by the one you are trying to know concepts for.

### 3.53.3 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.

### 3.53.4 See also

## 3.54 trunk/Scripts/Bash/Functions/Render

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.

### 3.54.1 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).

#### 3.54.1.1 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 structures and pre-rendition configuration structures). The bond component is the path convention 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 convention 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. Without 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 (see [Section 3.55 \[trunk Scripts Bash Functions Render Config\]](#), page 76).

Inside the working copy, you can find an example of “design template without translation” configuration at ‘trunk/Identity/Models/’.

See [Section 3.1 \[trunk Identity\]](#), page 3, 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.

See [Section 3.55 \[trunk Scripts Bash Functions Render Config\]](#), page 76, for more information.

### 3.54.1.2 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 untranslated 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-rendering 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-rendering 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 translation 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.
```

### 3.54.1.3 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?

Certainly, 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-rendering 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.

### 3.54.1.4 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 combined with design templates produce translated images. That may seem 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 independently 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

```

Inside 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 backgrounds) 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...

### 3.54.2 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.

### 3.54.3 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 convention, 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/Id  
- 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 transferred 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 individually 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 imposed 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.

### 3.54.4 Usage

- ...

### 3.54.5 See also

## 3.55 trunk/Scripts/Bash/Functions/Render/Config

### 3.55.1 Goals

The ‘trunk/Scripts/Bash/Config’ directory exists to organize pre-rendering configuration scripts.

### 3.55.2 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 entries. Pre-rendering configuration entries are required for each identity entry, but not for translation entries.

#### 3.55.2.1 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.

### 3.55.2.2 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.

### 3.55.2.3 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** See [Section 3.54 \[trunk Scripts Bash Functions Render\]](#), page 66, 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'
```

### 3.55.3 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.

### 3.55.4 See also

## 3.56 trunk/Scripts/Bash/Functions/Shell

### 3.56.1 Goals

This section exists to organize files related to `shell` functionality of 'centos-art.sh' script.

### 3.56.2 Description

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 maintenance 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| }
```

Figure 3.4: The functions script base comment structure

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| # doSomething.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| }

```

Figure 3.5: The function script comment example

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](#).

### 3.56.3 Usage

```
centos-art sh --update-copyright='path/to/dir'
centos-art sh --update-copyright='path/to/dir' --filter='regex'
```

Use these commands to update copyright information in `.sh` files under `'path/to/dir'` directory.

When you provide `'--filter=regex'` argument, the list of files to process is reduced as specified in `'regex'` regular expression. Inside `'centos-art.sh'` script, the `'regex'` regular expression is used in combination with `find` command to look for files matching the regular expression path pattern.

**Warning** In order for `'regex'` regular expression to match a file, the `'regex'` regular expression must match the whole file path not just the file name.

For example, if you want to match all `'render.conf.sh'` files inside `'path/to/dir'`, use the `./render.conf` regular expression. Later, `'centos-art.sh'` script uses this value inside `~$REGEX\.sh$` expression in order to build the final regular expression (i.e., `^./render.conf\.sh$`) that is evaluated against available file paths inside the list of files to process.

Exceptionally, when you provide `'--filter=regex'` in the way that `'regex'`, appended to `'path/to/dir/'` (i.e. `'path/to/dir/regex'`), matches a regular file; the `'centos-art.sh'` script uses the file matching as only file in the list of files to process.

### 3.56.4 See also

## 3.57 trunk/Scripts/Bash/Functions/Svg

### 3.57.1 Goals

This section exists to organize files related to `svg` functionality of `'centos-art.sh'` script.

### 3.57.2 Description

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 definitions 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.

#### 3.57.2.1 Metadata maintainance

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 commands 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

translated. 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.

The `'centos-art.sh'` script modifies the following metadata:

- 'Title'      Name by which this document is formally known. If no value is set here, `'centos-art.sh'` script uses the file name as title.
- 'Date'        Date associated with the creation of this document (YYYY-MM-DD). If no value is set here, `'centos-art.sh'` script uses the current date information as in `date +%Y-%m-%d`.
- 'Creator'     Name of entity primarily responsible for making the content of this document. If no value is set here, `'centos-art.sh'` script uses the string `'The CentOS Project'`.
- 'Rights'      Name of entity with rights to the intellectual Property of this document. If no value is set here, `'centos-art.sh'` script uses the string `'The CentOS Project'`.
- 'Publisher'   Name of entity responsible for making this document available. If no value is set here, `'centos-art.sh'` script uses the string `'The CentOS Project'`.
- 'Identifier'   Unique URI to reference this document. If no value is set here, `'centos-art.sh'` script uses the current file path to build the related url that points to current file location inside repository central server.
- 'Source'      Unique URI to reference the source of this document. If no value is set here, `'centos-art.sh'` script uses current file path to build the related url that points to current file location inside repository central server.
- 'Relation'    Unique URI to a related document. If no value is set here, `'centos-art.sh'` script uses current file path to build the related url that points to current file location inside repository central server.
- 'Language'    Two-letter language tag with optional subtags for the language of this document. (e.g. `'en-GB'`). If no value is set here, `'centos-art.sh'` script uses the current locale information as in `cli_getCurrentLocale` function.
- 'Keywords'    The topic of this document as comma-separated key words, phrases, or classifications. If no value is set here, `'centos-art.sh'` script uses file path to build
- 'Coverage'    Extent or scope of this document. If no value is set here, `'centos-art.sh'` script uses the string `'The CentOS Project'`.
- 'Description' Description about the document. If no value is set here, `'centos-art.sh'` script uses empty value as default.
- 'Contributors' People that contributes in the creation/maintainance of the document. If no value is set here, `'centos-art.sh'` script uses empty value as default.

The ‘License’ metadatum is not set as a choice, by now. It is fixed **Creative Commons Attribution Share-Alike 3.0 License**. This is done in order to grant license consistency among all SVG files we manage inside CentOS Artwork Repository.

### 3.57.2.2 Unused definitions

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 ‘--vacuum-defs’ option.

### 3.57.3 Usage

```
centos-art svg --update-metadata='path/to/dir'
```

```
centos-art svg --update-metadata='path/to/dir' --filter='regex'
```

Use these commands to update metadata information to ‘.svg’ files under ‘path/to/dir’ directory.

```
centos-art svg --vacuum-defs='path/to/dir'
```

```
centos-art svg --vacuum-defs='path/to/dir' --filter='regex'
```

Use these commands to remove unused definitions inside ‘.svg’ files under ‘path/to/dir’ directory.

When you provide ‘--filter=’regex’’ argument, the list of files to process is reduced as specified in ‘regex’ regular expression. Inside ‘centos-art.sh’ script, the ‘regex’ regular expression is used in combination with `find` command to look for files matching the regular expression path pattern.

**Warning** In order for ‘regex’ regular expression to match a file, the ‘regex’ regular expression must match the whole file path not just the file name.

For example, if you want to match all ‘summary.svg’ files inside ‘path/to/dir’, use the `./summary` regular expression. Later, ‘centos-art.sh’ script uses this value inside `^$REGEX\.svg$` expression in order to build the final regular expression (i.e., `^./summary\.svg$`) that is evaluated against available file paths inside the list of files to process.

Exceptionally, when you provide ‘--filter=’regex’’ in the way that ‘regex’, appended to ‘path/to/dir/’ (i.e. ‘path/to/dir/regex’), matches a regular file; the ‘centos-art.sh’ script uses the file matching as only file in the list of files to process.

### 3.57.4 See also

## 3.58 trunk/Scripts/Bash/Functions/Verify

### 3.58.1 Goals

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

## 3.58.2 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 `verify` 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 `verify` 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.

### 3.58.2.1 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 `verify` 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.

### 3.58.2.2 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.

### 3.58.2.3 Environment variables

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

The `'verify'` 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 physical 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.

**Convention** In order to keep changes synchronized 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.

### 3.58.3 Usage

#### `centos-art verify --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 verify --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 verify --environment`

#### `centos-art verify --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.

### 3.58.4 See also

## 3.59 trunk/Scripts/Bash/Locale



### 3.59.1 Goals

This section exists to organize translation messages and templates used by ‘centos-art.sh’ script.

### 3.59.2 Description

Translated messages of ‘centos-art.sh’ script are managed using GNU `gettext` utilities. Most translation actions have been automated through ‘centos-art.sh’ script “locale” functionality (see [Section 3.51 \[trunk Scripts Bash Functions Locale\]](#), page 60).

### 3.59.3 Usage

The content of ‘trunk/Scripts/Bash/Locale’ directory should not be managed manually. Instead, use the “locale” functionality of ‘centos-art.sh’ script. See [Section 3.51 \[trunk Scripts Bash Functions Locale\]](#), page 60, for more information on how to use ‘centos-art.sh’ “locale” functionality.

### 3.59.4 See also

## 3.60 trunk/Scripts/Perl

### 3.60.1 Goals

- ...

### 3.60.2 Description

### 3.60.3 Usage

### 3.60.4 See also

## 3.61 trunk/Scripts/Python

### 3.61.1 Goals

- ...

### 3.61.2 Description

- ...

### 3.61.3 Usage

- ...

### 3.61.4 See also

# Index

## B

branches ..... 1

## C

Common translation files ..... 33

## H

How to render brands' translation files ..... 37

How to render fonts' translation files ..... 38

How to render translation files ..... 36

## M

Metadata maintainance ..... 81

## S

Specific translation files ..... 34

## T

tags ..... 2

Template translation files ..... 31

Translation brands file names ..... 37

Translation configuration scripts ..... 34

Translation entries ..... 28

Translation files ..... 30

Translation markers ..... 29

Translation paths ..... 28

Translation pre-rendering configuration scripts ... 34

Translation rendering ..... 34

Translation rendering default functionality ..... 35

trunk ..... 3

trunk Identity ..... 3

trunk Identity Brands ..... 4

trunk Identity Budokan ..... 4

trunk Identity Fonts ..... 5

trunk Identity Icons ..... 6

trunk Identity Isolinux ..... 6

trunk Identity Models ..... 6

trunk Identity Models Css ..... 7

trunk Identity Models Html ..... 7

trunk Identity Models Img Promo Web ..... 7

trunk Identity Models Tpl ..... 7

trunk Identity Models Tpl Promo Web ..... 8

trunk Identity Models Xcf ..... 12

trunk Identity Release ..... 12

trunk Identity Themes ..... 12

trunk Identity Themes Models ..... 13

trunk Identity Themes Models Alternative ..... 13

trunk Identity Themes Models Default ..... 13

trunk Identity Themes Models Default Distro .... 14

trunk Identity Themes Models Default Distro

    Anaconda ..... 15

trunk Identity Themes Models Default Promo .... 15

trunk Identity Themes Models Default Web ..... 15

trunk Identity Themes Motifs ..... 16

trunk Identity Themes Motifs Flame ..... 17

trunk Identity Themes Motifs Modern ..... 20

trunk Identity Themes Motifs Modern Backgrounds

    ..... 20

trunk Identity Themes Motifs Modern Backgrounds

    Img ..... 21

trunk Identity Themes Motifs Modern Backgrounds

    Tpl ..... 22

trunk Identity Themes Motifs Modern Backgrounds

    Xcf ..... 22

trunk Identity Themes Motifs Modern Distro

    Anaconda Progress ..... 22

trunk Identity Themes Motifs Modern Palettes ... 23

trunk Identity Themes Motifs TreeFlower ..... 23

trunk Identity Themes Motifs TreeFlower

    Backgrounds ..... 23

trunk Identity Widgets ..... 27

trunk Locales ..... 28

trunk Locales Identity ..... 36

trunk Locales Identity Brands ..... 36

trunk Locales Identity Brands Tpl ..... 38

trunk Locales Identity Fonts ..... 38

trunk Locales Identity Models ..... 39

trunk Locales Identity Release ..... 39

trunk Locales Identity Themes ..... 39

trunk Locales Identity Themes Backgrounds ..... 39

trunk Locales Identity Themes Distro Anaconda

    Progress ..... 39

trunk Locales Identity Widgets ..... 41

trunk Manuals ..... 41

trunk Scripts ..... 41

trunk Scripts Bash ..... 42

trunk Scripts Bash Functions ..... 44

trunk Scripts Bash Functions Html ..... 60

trunk Scripts Bash Functions Locale ..... 60

trunk Scripts Bash Functions Manual ..... 61

trunk Scripts Bash Functions Path ..... 62

trunk Scripts Bash Functions Render ..... 66

trunk Scripts Bash Functions Render Config ..... 76

trunk Scripts Bash Functions Shell ..... 78

trunk Scripts Bash Functions Svg ..... 81

trunk Scripts Bash Functions Verify ..... 83

trunk Scripts Bash Locale ..... 86

trunk Scripts Perl ..... 87

trunk Scripts Python ..... 87

## U

Unused definitions ..... 83

## List of Figures

Figure 3.1: The functionalities initialization environment. ....	43
Figure 3.2: The actions initialization environment. ....	44
Figure 3.3: The <code>cli_commitRepoChanges</code> function output. ....	52
Figure 3.4: The functions script base comment structure ....	79
Figure 3.5: The function script comment example. ....	80