CentOS Artwork Repository

Manual

Alain Reguera Delgado

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

Table of Contents

1 Introduction

Welcome to CentOS Artwork Repository Manual.

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

This manual discusses the following intermedite topics:

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

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

1.1 History

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

2008

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

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

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

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

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

2009

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

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

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

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

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

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

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

2010

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

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

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

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

2011

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

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

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

1.2 Authors

This section records authoring information of CentOS Artwork Repository along the years:

Graphic Design

Work line: Section 2.4 [Directories trunk Identity], page 14

- Guideon de Kok
- Alain Reguera Delgado
- Marcus Moeller

Documentation

Work line: Section 2.35 [Directories trunk Manual], page 41

- Alain Reguera Delgado
- Ralph Angenendt

Localization

Work line: Section 2.34 [Directories trunk Locales], page 40

- Alain Reguera Delgado (Spanish)

Automation

Work line: Section 2.36 [Directories trunk Scripts], page 42

- Alain Reguera Delgado

Infrastructure

- Karanbirn Singh
- Ralph Angenendt

Packaging

- Karanbirn Singh
- Ralph Angenendt

1.3 Copying Conditions

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

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

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

The `centos-art.sh` script

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

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

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of the `centos-art.sh` script, you must give

the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

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

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

1.4 Document Convenctions

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

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

`command`

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

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

'`file name`'

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

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

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

⟨key⟩

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

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

⟨key-combination⟩

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

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

`computer output`

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

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

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

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

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

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

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

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

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

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

1.5 Repository Convenctions

The CentOS Artwork Repository is supported by Subversion (http://subversion.tigris.org/), 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.

When using Subversion there is one *source repository* and many *working copies* of that source repository. The working copies are independent one another, can be distributed all around the world and provide a local place for designers, documentors, translators and programmers to perform their works in a descentralized way. The source repository, on the other hand, provides a central place for all independent working copies to interchange data and provides the information required to permit extracting previous versions of files at any time.

1.5.1 Repository policy

The CentOS Artwork Repository is a collaborative tool that anyone can have access to. However, changing that tool in any form is something that should be requested in centos-devel@centos.org mailing list. Generally, people download working copies from CentOS Artwork Repository, study the repository organization, make some changes in their working copies, make some tests to verify such changes do work the way expected and finally request access to commit them up to the CentOS Artwork Repository (i.e., the source repository) for others to benefit from them.

Once you've received access to commit your changes, there is no need for you to request permission again to commit other changes from your working copy to CentOS Artwork Repository as long as you behave as a *good community citizen*.

As a good community citizen one understand of a person who respects the work already done for others and share ideas with authors before changing relevant parts of their work, specially in situations when the access required to realize the changes has been granted already. Of course, there is a time when conversation has taken place, the paths has been traced and changing the work is so obvious that there is no need for you to talk about it; that's because you already did, you already built the trust to keep going. Anyway, the mailing list mentioned above is

available for sharing ideas in a way that good relationship between community citizens could be constantly balanced.

The relationship between community citizens is monitored by repository administrators. Repository administrators are responsible of granting everything goes the way it needs to go in order for the CentOS Artwork Repository to comply its mission which is: to provide a colaborative tool for The CentOS Community where The CentOS Project Corporate Identity is built and maintained from The CentOS Community itself.

It is also important to remember that all source files inside CentOS Artwork Repository should comply the terms of GNU General Public License in order for them to remain inside the repository. See file trunk/Scripts/COPYING, for a complete license description.

1.5.2 Repository organization

The CentOS Artwork Repository uses a '`trunk`', '`branches`', and '`tags`' organization.

'`trunk`'

The '`trunk`' directory organizes the main development line of CentOS Artwork Repository. See Section 2.3 [Directories trunk], page 14, for more information.

'`branches`'

The '`branches`' directory oranizes intermediate development lines taken from the main development line. See Section 2.1 [Directories branches], page 13, for more information.

'`tags`'

The '`tags`' directory organizes frozen development lines taken either from the main or the intermediate lines of development. See Section 2.2 [Directories tags], page 13, for more information.

1.5.3 Repository file names

Inside the CentOS Artwork Repository, file names are all written in lowercase (e.g., '`01-welcome.png`', '`splash.png`', '`anaconda_header.png`', etc.) and directory names are all written capitalized (e.g., '`Identity`', '`Themes`', '`Motifs`', '`TreeFlower`', etc.).

1.5.4 Repository work lines

Inside CentOS Artwork Repository there are four major work lines of production which are: *graphic design*, *documentation*, *localization* and *automation*. These work lines describe different areas of content production. Content production inside these specific areas may vary as much as persons be working on them. Producing content in too many different ways may result innapropriate in a collaborative environment like CentOS Artwork Repository where content produced in one area depends somehow from content produced in another different area. So, a *content production standard* is required for each available work line.

1.5.4.1 Graphic design

The graphic design work line exists to cover brand design, typography design and themes design mainly. Additionally, some auxiliar areas like icon design, illustration design, brushes design, patterns designs and palettes of colors are also included here for completeness.

Inside CentOS Artwork Repository graphic design is performed through Inkscape (`http://www.inkscape.org/`) and GIMP (`http://www.gimp.org/`). The Inkscape tool is used to create and manipulate scalable vector graphics and export them to PNG format; it also provides a command-line interface that we use to perform massive exportation from SVG files to PNG files in automation scripts. On the other hand, GIMP is used to create and manipulate rastered images, create brushes, patterns and palettes of colors.

Tip Combine both Inkscape and GIMP specific functionalities and possibilities to produce very beautiful images.

The CentOS Project Corporate Visual Identity is made of different visual manifestations (e.g., Distributions, Web sites, Stationery, etc.). Visual manifestations implement the corporate identity concepts by mean of images. To produce these images, we decompose image production in *design models* and *artistic motifs*.

Design models provide the structural information of images (i.e., dimension, position of common elements in the visible area, translation markers, etc.) and they are generally produced as scalable vector graphics to take advantage of SVG standard, an XML-based standard.

Artistic motifs provide the visual style (i.e., the background information, the look and feel) some design models need to complete the final image produced by automation scripts. Artistic motifs are generally produced as rastered images.

The result produced from combining one design model with one artistic motif is what we know as a *theme*. Inside themes directory structure (see Section 2.12 [Directories trunk Identity Themes], page 24), you can find several design models and several artistic motifs independently one another that can be albitrarily combined through *theme rendition*, a flexible way to produce images for different visual manifestations in very specific visual styles. Inside themes directory structure, theme rendition is performed in '`trunk/Identity/Themes/Motifs`' directory structure, the required design models are taken from '`trunk/Identity/Themes/Models`' directory structure and the action itself is controlled by the `render` functionality of `centos-art.sh` script.

In addition to theme rendition you can find *direct rendition*, too. Direct rendition is another way of image production where there is no artistic motif at all but design models only. Direct rendition is very useful to produce simple content that doesn't need specific background information. Some of these contents are brands, icons and illustrations. Direct rendition is performed in '`trunk/Identity/Images`', the required design models are taken from '`trunk/Identity/Models`' directory structure and the action itself is controlled by the `render` functionality of `centos-art.sh` script.

See Section 2.4 [Directories trunk Identity], page 14, for more information about The CentOS Corporate Identity and how graphic design fits on it.

1.5.4.2 Documentation

The documentation work line exists to describe what each directory inside the CentOS Artwork Repository is for, the conceptual ideas behind them and, if possible, how automation scripts make use of them.

The CentOS Artwork Repository documentation is supported by Texinfo, a documentation system that uses a single source file to produce both online information and printed output.

The repository documentation is organized under '`trunk/Manual`' directory structure and uses the repository directories as reference. Each directory structure in the repository has a documentation entry associated in the documentation manual. Documentation entries are stored under '`trunk/Manual/Directories`' directory structure and the action itself is controlled by the `help` functionality of `centos-art.sh` script.

The `help` functionality let you create, edit and delete documentation entries in a way that you don't need to take care of updating menus, nodes and cross reference information inside the manual structure; the functionality takes care of it for you. However, if you need to write repository documentation that have nothing to do with repository directories (e.g., Preface, Introduction and similar) you need to do it manually, there is no functionality to automate such process yet.

See Section 2.35 [Directories trunk Manual], page 41, for more information on documentation.

1.5.4.3 Localization

The localization work line exists to provide the translation messages required to produce content in different languages. Translation messages inside the repository are stored as portable objects (e.g., .po, .pot) and machine objects (.mo) under '`trunk/Locales`' directory structure.

The procedure used to localize content is taken from `gettext` standard specification. Basically, translatable strings are retrived from source files in order to create portable objects and machine objects for them. These portable objects are editable files that contain the information used by translators to localize the translatable strings retrived from source files. On the other hand, machine objects are produced to be machine-redable only, as its name implies, and are produced from portable objects.

Since `gettext` needs to extract translatable strings form source files in order to let translators to localize them, we are limitted to use source files supported by `gettext` program. This is not a limitation at all since `gettext` supports most popular programming laguages (e.g., C, C++, Java, Bash, Python, Perl, PHP and GNU Awk just to mention a few ones). Nevertheless, formats like SVG, XHTML and Docbook don't figure as supported formats in the list of `gettext` supported source files.

To translate XML based source files like SVG, XHTML and Docbook we use the `xml2po` program instead. The `xml2po` comes with the '`gnome-doc-utils`' package and retrives translatable strings from one XML file to produce portable objects for them.

> **Note** Portable objects produced by `xml2po` have the same format that portable objects produced by `gettext`. This make the localization process quite consistent from translators' point of view. No matter what the source file be, the translator will always face the same translation file format (i.e., the portable object format).

With the portable object in place, the `xml2po` program is used again to create the final translated XML, just with the same definition of the source file where translatable strings were taken from (e.g., if we extract translatable strings from a SVG file, as result we get the same SVG file but with translatable strings already localized —obviously, for this to happen translators need to localize translatable strings inside the portable object first, localization won't appear as art of magic—). When using `xml2po`, the machine object is used as temporal file to produce the final translated XML file.

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

See Section 2.34 [Directories trunk Locales], page 40, for more information.

1.5.4.4 Automation

The automation work line exists to standardize content production in CentOS Artwork Repository. There is no need to type several tasks, time after time, if they can be programmed into just one executable script.

The automation work line takes place under '`trunk/Scripts`' directory structure. Here is developed the `centos-art.sh` script, a bash script specially designed to automate most frequent tasks (e.g., rendition, documentation and localization) inside the repository. Basically, the `centos-art.sh` script is divided in several functionalities independent one another that perform specific tasks and relay on repository organization to work as expected.

> **Tip** If you need to improve the way content is produced, look inside automation scripts and make your improvement there for everyone to benefit.

See Section 2.36 [Directories trunk Scripts], page 42, for more information on automation.

1.5.5 Connection between directories

In order to produce content in CentOS Artwork Repository, it is required that all work lines be connected somehow. This is the way automation scripts can know where to retrive the

information they need to work with (e.g., design model, translation messages, output location, etc.). We build this kind of connection using two path constructions named *master paths* and *auxiliar paths*.

The master path points only to directories that contain the source files (e.g., SVG files) required to produce base content (e.g., PNG files) through automation scripts. Each master path inside the repository may have several auxiliar paths associated, but auxiliar paths can only have one master path associated.

The auxiliar paths can point either to directories or files. When an auxiliar path points to a directory, that directory contains information that modifies somehow the content produced from master paths (e.g., translation messages) or provides the output information required to know where to store the content produced from master path. When an auxiliar path points to a file, that file has no other purpose but to document the master path it refers to.

The relation between auxiliar paths and master paths is realized combining two path informations which are: the master path itself and one second level directory structure from the repository. Generally, the master path is considered the path identifier and the second level directory structure taken from the repository is considered the common part of the path where the identifier is appended.

```
-----+---------------+---------------------------+------+-----------
Path | Suffix        | Identifier                |Prefix| Type
-----+---------------+---------------------------+------+-----------
  A  |               |trunk/Identity/Models/Brands|      | Directory
-----+---------------+---------------------------+------+-----------
  B  |  trunk/Manual/|trunk/Identity/Models/Brands|.texi | File
-----+---------------+---------------------------+------+-----------
  C  | trunk/Locales/|trunk/Identity/Models/Brands|      | Directory
-----+---------------+---------------------------+------+-----------
  D  |               |trunk/Identity/Images/Brands|      | Directory
-----+---------------+---------------------------+------+-----------
  E  | trunk/Locales/|trunk/Identity/Images/Brands|.texi | File
-----+---------------+---------------------------+------+-----------

  A = Master path.
  B = Auxiliar path to documentation entry.
  C = Auxiliar path to translation messages.
  D = Auxiliar path to final content output.
  E = Auxiliar path to documentation entry.
```

Figure 1.1: Path construction.

The path information described above (see Figure 1.1) is used by direct rendition and can be taken as reference to add other components that are equally produced in the repository. To add new components that make use of direct rendition inside the repository, change just the component name used above (e.g., '`Brands`') to that one you want to add, without changing the path structure around it.

The file organization used by theme rendition extends direct rendition by separating design models information from backgrounds information. To better understand this configuration, you can consider it as two independent lists, one of design models and one of artistic motifs, which are arbitrary combined between themselves in order to render images in specific ways. The possibilities of this configuration are endless and let us describe visual manifestations very well. For example, consider the organization used to produce '`Anaconda`' images; for CentOS distribution major release 5; using '`Default`' design models and version '`3`' of '`Flame`' artistic motif:

```
-----+---------------+-----------------------------------------------------+------+-----
Path | Suffix        | Identifier                                           |Prefix| Type
-----+---------------+-----------------------------------------------------+------+-----
  A  |               |trunk/Identity/Themes/Models/Default/Distro/5/Anaconda|      | Dire
-----+---------------+-----------------------------------------------------+------+-----
  B  |  trunk/Manual/|trunk/Identity/Themes/Models/Default/Distro/5/Anaconda|.texi | File
-----+---------------+-----------------------------------------------------+------+-----
  C  | trunk/Locales/|trunk/Identity/Themes/Models/Default/Distro/5/Anaconda|      | Dire
-----+---------------+-----------------------------------------------------+------+-----
  D  |               |trunk/Identity/Themes/Motifs/Flame/3/Distro/5/Anaconda|      | Dire
-----+---------------+-----------------------------------------------------+------+-----
  E  | trunk/Locales/|trunk/Identity/Themes/Motifs/Flame/3/Distro/5/Anaconda|.texi | File
-----+---------------+-----------------------------------------------------+------+-----

  A = Master path.
  B = Auxiliar path to documentation entry.
  C = Auxiliar path to translation messages.
  D = Auxiliar path to final content output.
  E = Auxiliar path to documentation entry.
```

Figure 1.2: Path construction extended.

The path information described above (see Figure 1.2) is used by theme rendition and can be taken as reference to add other components that are equally produced in the repository.

In this configuration we can change both design model name (e.g., '`Default`') and artistic motif name (e.g., '`Flame/3`') to something else in order to achieve a different result. The only limitations impossed are the storage space provided in the server machine and your own creativeness as graphic designer.

> **Note** A theme ready for implementation may consume from 100 MB to 400 MB of storage space. The exact space consumed by a theme depends on the amount of screen resolutions the theme supports. The more screen resolutions the theme supports, the more storage space demanded for it.

In this configuration we saw how to build the path information for '`Anaconda`' component as part of CentOS Distribution visual manifestation, but that is not the only component we have inside CentOS Distribution visual manifestation. There are other components like Syslinux, Grub, Rhgb, Gdm, Kdm, Gsplash and Ksplash that share a similar file organization to that described above for '`Anaconda`' component.

1.5.6 Syncronizing path information

Syncronizing path information is the action that keeps all path information up to date in the repository. This action implies both *file movement* and *file content replacement* in this very specific order. File movement is related to duplicate, delete and rename files and directories in the repository. File content replacement is related to replace information, path information in this case, inside files in the repository.

The order followed to syncronize path information is relevant because the versioned nature of the files we are working with. We don't perform file content replacement first because that would imply a repository change which will immediatly demmand a commit in order for actions like duplicate, delete or rename to take place. However, if we perform file movement first, it is possible to commit both file moved and file content replacements as if they were just one change. In this case the file content replacement takes palce in the target location that have been duplicated or renamed, not the one use as source location. This configuration is specially

useful when files are renamed (i.e., one file is copied from a source location to a target location and then the source location of it is removed from repository).

> **Warning** There is no support for URLs actions inside `centos-art.sh` script. The `centos-art.sh` script is designed to work with local files inside the working copy only. If you need to perform URL actions directly, use Subversion commands instead.

When one master path is changed it is required that all related auxiliar paths be changed, too. This is required in order for master paths to retain their relation with auxiliar paths. This way, automation scripts are able to know where to retrive translation messages from, where to store final output images to and where to look for documentation. If relation between master paths and auxiliar paths is lost, there is no way for automation scripts to know where to retrive the information they need.

The auxiliar paths should never be modified under any reason but to satisfy the relationship with the master path. Liberal change of auxiliar paths may suppress the conceptual idea they were initially created for; and certainly, automation scripts may stop working as expected. The update direction to rename path information must be from master path to auxiliar path and never the opposite.

The relation between master and auxiliar paths is useful to keep repository organized but introduce some complications when we work with files that use master path information as reference to build structural information. This is the case of repository documentation manual source files where inclusions, menus, nodes and cross references are built using master path information as reference. Now, to see what kind of complication we are talking about, consider what would happen to a structural definitions (i.e., inlusions, menus, nodes and cross refereces) already set in the manual from one master path that is suddenly renamed to something different. If the path information is not syncronized, at this point, we lose connection between the master path and the auxiliar path created to store the related documentation entry, as well as the related structural definitions that end up pointing to a master path that no longer exist.

The syncronization of path information is aimed to solve these kind of issues.

1.5.7 Extending repository organization

Occasionly, you may find that new components of The CentOS Project Corporate Identity need to be added to the repository in order to work them out. If that is the case, the first question we need to ask ourselves, before start to create directories blindly all over, is: *What is the right place to store it?*

The best place to find answers is in The CentOS Community (see page `http://wiki.centos.org/GettingHelp`), 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 in order to make your own propositions based on it.

When extending respository structure it is very useful to bear in mind The CentOS Project Corporate Identity Structure (see Section 2.4 [Directories trunk Identity], page 14) The CentOS Mission and The CentOS Release Schema. The rest is just matter of choosing appropriate names. It is also worth to know that each directory in the repository responds to a conceptual idea that justifies its existence.

To build a directory structure, you need to define the conceptual idea first and later create the directory. There are some locations inside the repository that already define some concepts you probably want to reuse. For example, '`trunk/Identity/Themes/Motifs`' to store theme artistic motifs, '`trunk/Identity/Themes/Models`' to store theme design models, '`trunk/Manual`' to store documentation files, '`trunk/Locales`' to store translation messages, '`trunk/Scripts`' to store automation scripts and so on.

To illustrate this desition process let's consider the '`trunk/Identity/Themes/Motifs/TreeFlower/3`' directory structure as example. This directory can be read as: the theme development line of version '`3`' of '`TreeFlower`' artistic motif. Additional, we can identify that artistic motifs are part of themes as well as themes are part of The CentOS Project Corporate Identity. These concepts are better described independently in each documentation entry related to the directory structure as it is respectively shown in the list of commands bellow.

```
centos-art help --read turnk
centos-art help --read turnk/Identity
centos-art help --read turnk/Identity/Themes
centos-art help --read turnk/Identity/Themes/Motifs
centos-art help --read turnk/Identity/Themes/Motifs/TreeFlower
centos-art help --read turnk/Identity/Themes/Motifs/TreeFlower/3
```

The concepts behind other location can be found in the same way described above, just change the path information used above to the one you are trying to know concepts for.

1.6 Send in Your Feedback

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

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

2 The Repository Directories

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

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

2.1 The 'branches' Directory

Goals

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

Description

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

Usage

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

See also

- Section 2.2 [Directories tags], page 13.
- Section 2.3 [Directories trunk], page 14.
- The Subversion book (`http://svnbook.red-bean.com/`).

2.2 The 'tags' Directory

Goals

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

Description

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

Usage

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

See also

- Section 2.1 [Directories branches], page 13.
- Section 2.3 [Directories trunk], page 14.
- The subversion book (`http://svnbook.red-bean.com/`).

2.3 The 'trunk' Directory

Goals

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

Description

The '`trunk/`' directory structure provides the main development line inside the CentOS Artwork Repository.

Usage

- See Section 2.4 [Directories trunk Identity], page 14.
- See Section 2.35 [Directories trunk Manual], page 41.
- See Section 2.34 [Directories trunk Locales], page 40.
- See Section 2.36 [Directories trunk Scripts], page 42.

See also

- Section 2.1 [Directories branches], page 13.
- Section 2.2 [Directories tags], page 13.
- The Subversion book (`http://svnbook.red-bean.com/`).

2.4 The 'trunk/Identity' Directory

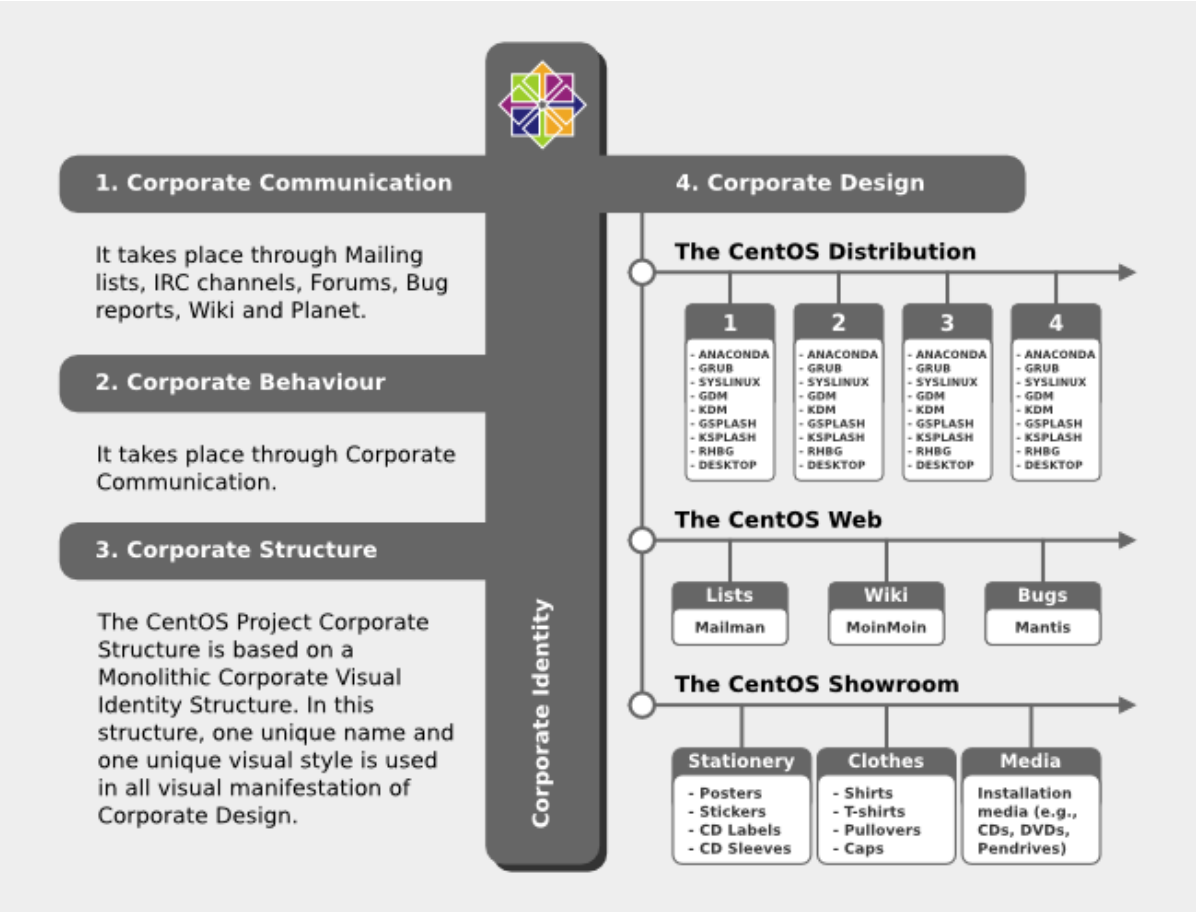
Goals

The '`trunk/Identity`' describes what The CentOS Project Corporate Identity is and the components it is made of.

Description

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

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



Corporate Mission

The CentOS Project exists to provide The CentOS Distribution. Additionally, The CentOS Project provides The CentOS Web and The CentOS Showroom to support and promote the existence of The CentOS Distribution, respectively.

Corporate Design

Corporate design is focused on the effective communication of corporate visual messages. Corporate visual messages are all the information emitted by a corporation that can be perceived by the people through their visual sence (i.e., the human eye). In order for such visual communication to happen, it is required to put the visual message on medium available for people to see. These kind of media are know as corporate visual manifestations, since the corporate manifests its existence through them using corporate design.

The amount of visual manifestations a corporation uses to communicate its existence is very specific to each corporation itself. Inside The CentOS Project Corporate Identity, considering *The CentOS Project Corporate Structure*, *The CentOS Project Corporate Mission* and *The CentOS Project Release Schema*, the following visual manifestations were defined:

The CentOS Distribution
: The CentOS Distribution visual manifestation exists to cover all actions related to artwork production and rebranding required by the The CentOS Distribution (see

Section 2.16 [Directories trunk Identity Themes Models Default Distro], page 27) in order to comply with its upstream redistribution guidelines.

The CentOS Distribution is made of software packages. Inside the distribution there are packages that make a remarkable use of images and there are packages that don't use images at all. The CentOS Distribution visual manifestation gets focused on software packages that do use images in a remarkable way (e.g., '`anaconda`', '`grub`', '`syslinux`', '`gdm`', '`kdm`') and that way, through images, implements the corporate design in The CentOS Distribution (i.e., the operating system).

The CentOS Web

The CentOS Web visual manifestation exists to support The CentOS Distribution.

The CentOS Web covers web applications which let The CentOS Project to manifest its existence on the Internet. Through these web applications The CentOS Project provides Corporate Communication. These web applications are free software and come from different providers which distribute their work with predefined visual styles. Frequently, these predefined visual styles have no visual relation among themselves and introduce some visual contraditions when they all are put together. These visual contraditions need to be removed in order to comply with The CentOS Project Corporate Structure guidelines.

The CentOS Showroom

The CentOS Showroom visual manifestation exists to promote The CentOS Distribution.

The CentOS Showroom covers industrial production of objects branded by The CentOS Project (e.g., clothes, stationery and installation media). These branded objects are for distribution on social events and/or shops. They provide a way of promotion and a route for commercialization that may help to aliviate The CentOS Project expenses (e.g., electrical power, hosting, servers, full-time-developers, etc.), in a similar way as donations may do.

The visual manifestations above seem to cover all the media required by The CentOS Project, as organization, to show its existence. However, other visual manifestations could be added in the future, if needed, to cover different areas like building, offices, road transportation and whaterver visual manifestation The CentOS Project thouches to show its existence.

Corporate Communication

The CentOS Project Corporate Communication is based on *Community Communication* and takes place through the following avenues:

- The CentOS Chat (`#centos`, `#centos-social`, `#centos-devel` on irc.freenode.net)
- The CentOS Mailing Lists (`http://lists.centos.org/`).
- The CentOS Forums (`http://forums.centos.org/`).
- The CentOS Wiki (`http://wiki.centos.org/`).
- Social events, interviews, conferences, etc.

Corporate Behaviour

The CentOS Project Corporate Behaviour is based on *Community Behaviour* which take place on *Corporate Communication*.

Corporate Structure

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

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

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

The CentOS Project, as organization, is mainly made of (but not limited to) three visual manifestions: Distribution, Web and Showroom. Inside the Distribution visual manifestations, The CentOS Project maintains near to four different major releases of CentOS Distribution, parallely in time. However, inside The CentOS Web visual manifestations, the content is produced for no specific release information (e.g., there is no a complete web site for each major release of The CentOS Distribution individually, but one web site to cover them all). Likewise, the content produced in The CentOS Showroom is created for no release-specific at all, but for The CentOS Project in general.

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

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

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

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

Usage

The '`trunk/Identity`' directory structure organizes most files used to build and implement The CentOS Project Corporate Identity. In that sake, the following work lines are available:

Brushes

This work line provides brushes for GIMP. When you prepare the repository, brushes in this location are made available immediatly for you to use in the "Brushes" panel of GIMP.

See Section 2.5 [Directories trunk Identity Brushes], page 19, for more information.

Fonts

This work line provides the typography information required by all different visual manifestations of The CentOS Project. When you prepare the repository, fonts in this location are made available immediatly for you to use in GIMP and Inkscape.

See Section 2.6 [Directories trunk Identity Fonts], page 20, for more information.

Images

This work line provides output location for final images that don't need to use background images (e.g., brands, icons, illustrations, etc.).

See Section 2.7 [Directories trunk Identity Images], page 22, for more information.

Models

This work line provides design models for final images that don't need to use background images (e.g., brands, icons, illustrations, etc.).

See Section 2.8 [Directories trunk Identity Models], page 22, for more information.

Palettes

This work line provides palettes of colors for GIMP and Inkscape. When you prepare the repository, palettes of colors in this location are made available immediatly for you to use in the "Palettes" panel of GIMP and Inkscape.

See Section 2.10 [Directories trunk Identity Palettes], page 23, for more information.

Patterns

This work line provides patterns for GIMP. When you prepare the repository, patterns in this location are made available immediatly for you to use in the "Patterns" panel of GIMP.

See Section 2.11 [Directories trunk Identity Patterns], page 24, for more information.

Themes

This work line provides theme design models and theme artistic motifs for The CentOS Project. If you are interested in creating brand new visual styles for The CentOS Project this is the place for you.

See Section 2.12 [Directories trunk Identity Themes], page 24, for more information.

Webenv

This work line provides the HTML/XHTML and CSS standard definitions used by The CentOS Web visual manifestation. If you are a web developer and plan to improve The CentOS Web visual manifestation, then the files in this location may result very useful to you.

See Section 2.33 [Directories trunk Identity Webenv], page 36, for more information.

See also

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

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

2.5 The 'trunk/Identity/Brushes' Directory

Goals

This section describes how brushes are organized in the repository and how to make them available for you to use in GIMP (GNU Image Manipulation Program).

Description

A brush is a pixmap or set of pixmaps used for painting through an image manipulation program like GIMP. Inside the repository, we've organized brushes in *common brushes* and *theme-specific brushes*. In both cases, brushes are initially created in '`.xcf`' format and later exported to any of the brush formats recognized by GIMP (e.g., '`.gbr`' or '`.gih`') using the same name of its source file.

```
1. Common brushes        2. Theme-specific brushes
-----------------------  ------------------------------------------------------------
trunk/Identity/Brushes   trunk/Identity/Themes/Motifs/THEMENAME/THEMEVERSION/Brushes
|-- Xcf                  |-- Xcf
|   |-- 1.xcf            |   |-- 1.xcf
|   |-- 2.xcf            |   |-- 2.xcf
|   `-- 3.xcf            |   `-- 3.xcf
|-- 1.gbr                |-- 1.gbr
|-- 2.gih                |-- 2.gih
`-- 3.gbr                `-- 3.gbr
```

In order for both common brushes and theme-specific brushes to be loaded by GIMP, related '`.gbr`' and '`.gih`' brush files need to be stored under '`~/.gimp-2.2/brushes`' directory. This location is out of CentOS Artwork Repository and provides no version control by itself. This way, brushes aren't exported to this location but into the repository directory structure which is versioned. Later, we create symbolic links in '`~/.gimp-2.2/brushes`' to connect file brushes inside the repository and, this way, provide the configuration needed by GIMP to use the brush files produced inside the repository.

> **Warning** When brushes are added to or removed from the repository, you need to update your working copy and all information related to brushes inside your workstation (e.g., brush links in '`~/.gimp-2.2/brushes`' and the Brushes panel in GIMP). Otherwise, you may end up with broken links or brushes in the repository that wouldn't be available for you to use in GIMP.

Inside the repository, common brushes and theme-specific brushes are created individually in different locations, but they all are linked from one unique location (i.e., '`~/.gimp-2.2/brushes`'). This configuration may provoke brush overlapping if a name convenction is not implemented correctly. In that sake, file names used for brushes inside the repository must be unique, no matter where they be.

As file name convenction inside the repository, brushes are named using lowercase letters, numbers, minus characters and dot characters, only. Additionally, when links are built, we use one suffix for those brushes retrived from '`trunk/Identity/Brushes`' and another suffix for those brushes retrivided from theme-specific directories. Using both the brush file name and the suffix information, it is possible to build unique names for links under '`~/.gimp-2.2/brushes`' directory, scalably.

```
trunk/Identity/Brushes
|-- 1.gbr (file) <-- ~/.gimp-2.2/brushes/centos-1.gbr (link)
|-- 2.gbr (file) <-- ~/.gimp-2.2/brushes/centos-2.gbr (link)
`-- 3.gbr (file) <-- ~/.gimp-2.2/brushes/centos-3.gbr (link)
```

```
trunk/Identity/Themes/Motifs/THEMENAME/THEMEVERSION/Brushes
```

```
|-- 1.gbr (file) <-- ~/.gimp-2.2/brushes/centos-THEMENAME-THEMEVERSION-1.gbr (link)
|-- 2.gbr (file) <-- ~/.gimp-2.2/brushes/centos-THEMENAME-THEMEVERSION-2.gbr (link)
`-- 3.gbr (file) <-- ~/.gimp-2.2/brushes/centos-THEMENAME-THEMEVERSION-3.gbr (link)
```

Brushes produced with GIMP has a description field associated that is shown in the Brushes panel of GIMP. This description is set when the brush is created as '`.xcf`' file and can be updated when it is exported either to '`.gbr`' or '`.gih`' format. It wouldn't be too useful to have two or more brushes using the same description so, we also make description of brush files unique, too. In that sake, we use the same name schema used to name brush links as description but without including the file extension (e.g., if we have the '`centos-flame-3.gbr`' brush, its description would be `centos-flame-3`).

Usage

The way you use brushes is up to your creativeness. However, the way brushes are made available needs to be standardized. That's the reason of organizing brushes in common brushes and theme-specific brushes.

Common brushes

Common brushes exist to organize brushes that can be used anywhere inside the repository. Inside the repository, common brushes under '`trunk/Identity/Brushes`' are mainly used to hold brand information related to The CentOS Project (e.g., symbols, logos, trademarks, etc.).

Common brushes are always made available under '`~/.gimp-2.2/brushes`' directory after preparing the repository (see Section 2.40 [Directories trunk Scripts Functions Prepare], page 50).

Theme-specific brushes

Theme-specific brushes exist to organize brushes that can be used inside specific artistic motifs only. Inside the repository, theme-specific brushes are stored in a directory named '`Brushes`' which is stored in the first directory level under the artistic motif directory structure. Each artistic motif inside the repository has its own '`Brushes`' directory and uses it to store brushes that can be considered auxiliars to that artistic motif construction.

Theme-specific brushes aren't made available under '`~/.gimp-2.2/brushes`' directory after preparing the repository. In order to make theme-specific brushes available under '`~/.gimp-2.2./brushes`' it is required to activate/deactivate them using the `theme` functionality of `centos-art.sh` script.

See also

- The Gimp Manual, specifically the section related to Brushes.

2.6 The '`trunk/Identity/Fonts`' Directory

Goals

This section describes how typographies are organized in the repository and how to make them available for you to use in GIMP (GNU Image Manipulation Program) and Inkscape.

Description

The CentOS Project Corporate Identity is attached to '`DejaVu LGC`' font-family and '`Denmark`' font-family.

> **Caution** The copyright and license of '`Denmark`' typography aren't very specific and that issue may represent a threat to The CentOS Project Corporate Identity.

The '`Denmark`' typography is used as base to build The CentOS Logo (i.e., the main graphic design that connects/identifies all visual manifestations related to The CentOS Project). If the typography used to build The CentOS Logo is compromised somehow, the whole corporate visual identity it represents would be compromised, as well. To prevent such issues, it would be better for The CentOS Project to move on from '`Denmark`' typography to another typography (free, preferably) that retain the same visual style of '`Denmark`', but intruce a clearer copyright and license notice.

Usage

- See Section 2.9 [Directories trunk Identity Models Brands], page 22.

See also

- Section 2.4 [Directories trunk Identity], page 14.
- Section 2.3 [Directories trunk], page 14.

2.7 The 'trunk/Identity/Images' Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

2.8 The 'trunk/Identity/Models' Directory

Goals

- ...

Description

- ...

Usage

- . . .

See also

- ...

2.9 The 'trunk/Identity/Models/Brands' Directory

Goals

This section describes The CentOS Brand design models.

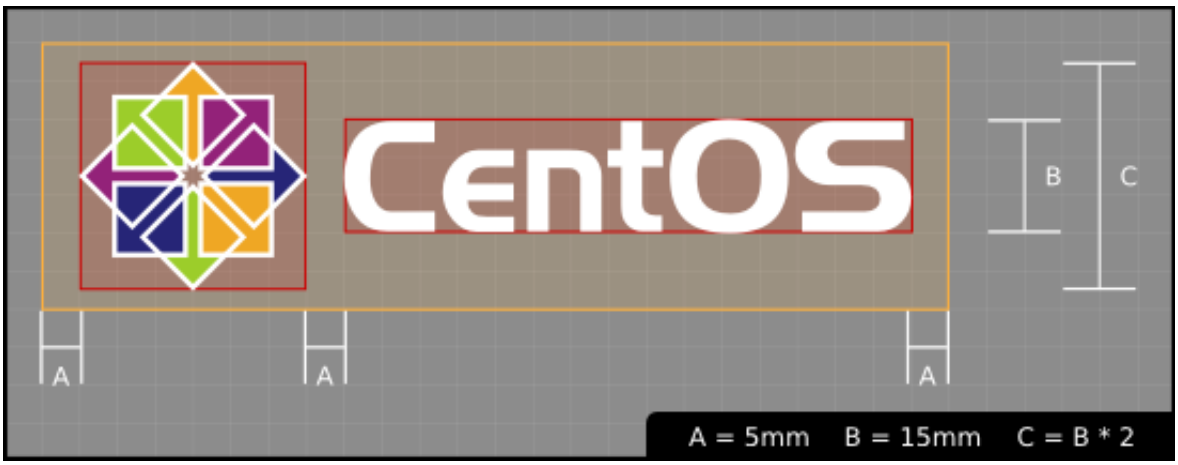
Description

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

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

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

with it on the left. The distance between '`CentOS`' word and phrase '`Community Enterprise Operating System`' have the size in points the phrase has.



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

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

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

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

Usage

See also

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

Goals

- ...

Description

- ...

Usage

- ...

See also

2.11 The 'trunk/Identity/Patterns' Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

2.12 The 'trunk/Identity/Themes' Directory

Goals

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

Description

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

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

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

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

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

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

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

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

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

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.

- See Section 2.13 [Directories trunk Identity Themes Models], page 25.
- See Section 2.28 [Directories trunk Identity Themes Motifs], page 32.

See also

- Section 2.4 [Directories trunk Identity], page 14.
- Section 2.3 [Directories trunk], page 14.

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

Goals

This section describes design models from The CentOS Themes.

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.

Default Design Model

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

Alternative Design Models

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

Usage

- See Section 2.14 [Directories trunk Identity Themes Models Default], page 26.

See also

- Section 2.12 [Directories trunk Identity Themes], page 24.
- Section 2.4 [Directories trunk Identity], page 14.
- Section 2.3 [Directories trunk], page 14.

2.14 The '`trunk/Identity/Themes/Models/Default`' Directory

Goals

This section describes the default design model of The CentOS Themes.

Description

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

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

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

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

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

How to remark the visual resemblance?
: Use a common artistic motifs as background for all CentOS Distribution images. See Section 2.28 [Directories trunk Identity Themes Motifs], page 32, for more information.

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

> **Important** Remarking the CentOS Release Schema inside each major release of CentOS Distribution —or similar visual manifestations— takes *high attention* inside

The CentOS Project corporate visual identity. It should be very clear for everyone which major release of CentOS Distribution is being used.

Usage

- See Section 2.16 [Directories trunk Identity Themes Models Default Distro], page 27.
- See Section 2.15 [Directories trunk Identity Themes Models Default Concept], page 27.

See also

- Section 2.12 [Directories trunk Identity Themes], page 24
- Section 2.13 [Directories trunk Identity Themes Models], page 25
- Section 2.28 [Directories trunk Identity Themes Motifs], page 32

2.15 The 'trunk/Identity/Themes/Models/Default/Concept' Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

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

Goals

This section organizes default design models for different major releases of CentOS Distribution.

Description

In order to better understatand how this visual manifestation is organized, it is necessary to consider what The CentOS Distribution is and how it is released.

The CentOS Distribution

The CentOS Distribution is an Enterprise-class Linux Distribution derived from sources freely provided to the public by a prominent North American Enterprise Linux vendor. The CentOS Distribution conforms fully with the upstream vendors redistribution policy and aims to be 100% binary compatible. (The CentOS Project mainly changes packages to remove upstream vendor branding and artwork.)

The CentOS Distribution is developed by a small but growing team of core developers. In turn the core developers are supported by an active user community including system administrators, network administrators, enterprise users, managers, core Linux contributors and Linux enthusiasts from around the world.

The CentOS Distribution Release Schema

The upstream vendor has released 4 versions of their EL (Enterprise Linux) product that The CentOS Project rebuilds the freely available SRPMS for. The upstream vendor releases security

updates as required by circumstances. The CentOS Project releases rebuilds of security updates as soon as possible. Usually within 24 hours (our stated goal is with 72 hours, but we are usually much faster).

The upstream vendor also releases numbered update sets for major versions of their EL product from 2 to 4 times per year. There are new ISOs from the upstream vendor provided for these update sets. Update sets will be completed as soon as possible after the upstream vendor releases their version ... generally within 2 weeks. The CentOS Project follows these conventions as well, so CentOS-3.9 correlates with EL 3 update 9 and CentOS-4.6 correlates with EL 4 update 6, CentOS-5.1 correlates to EL 5 update 1, etc.

One thing some people have problems understanding is that if you have any CentOS-3 product and update it, you will be updated to the latest CentOS-3.x version.

The same is true for CentOS-4 and CentOS-5. If you update any CentOS-4 product, you will be updated to the latest CentOS-4.x version, or to the latest CentOS-5.x version if you are updating a CentOS-5 system. This is exactly the same behavior as the upstream product. Let's assume that the latest EL4 product is update 6. If you install the upstream original EL4 CDs (the ones before any update set) and upgrade via `yum`, you will have latest update set installed (EL4 update 6 in our example). Since all updates within a major release (CentOS-2, CentOS-3, CentOS-4, CentOS-5) always upgrade to the latest version when updates are performed (thus mimicking upstream behavior), only the latest version is maintained in each main tree on The CentOS Mirrors (`http://mirrors.centos.org/`).

There is a CentOS Vault (`http://vault.centos.org/`) containing old CentOS trees. This vault is a picture of the older tree when it was removed from the main tree, and does not receive updates. It should only be used for reference.

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

Usage

Sometimes, between major releases, image files inside packages can be added, removed or just get the name changed. In order to describe such variations, the design models directory structure is organized in the same way the variations are introduced (i.e., through The CentOS Distribution Release Schema). So, each major release of The CentOS Distribution has its own design model directory structure.

When a new package/component is added to one or all the major releases of The CentOS Distribution, a design model directory structure for that component needs to be created. Later, it is filled up with related design models. Design models are created for each image file inside the component that need to be rebuilt in order to set the visual style and brand information correctly.

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

- See Section 2.17 [Directories trunk Identity Themes Models Default Distro 5], page 29.

See also

- Section 2.14 [Directories trunk Identity Themes Models Default], page 26.
- Section 2.13 [Directories trunk Identity Themes Models], page 25.

- Section 2.12 [Directories trunk Identity Themes], page 24.
- Section 2.4 [Directories trunk Identity], page 14.
- Section 2.3 [Directories trunk], page 14.

2.17 The 'trunk/Identity/Themes/Models/Default/Distro/5' Directory

Goals

- ...

Description

- ...

Usage

- See Section 2.26 [Directories trunk Identity Themes Models Default Distro 5 Syslinux], page 31.
- See Section 2.18 [Directories trunk Identity Themes Models Default Distro 5 Anaconda], page 29.
- See Section 2.25 [Directories trunk Identity Themes Models Default Distro 5 Rhgb], page 31.
- See Section 2.20 [Directories trunk Identity Themes Models Default Distro 5 Gdm], page 30.
- See Section 2.23 [Directories trunk Identity Themes Models Default Distro 5 Kdm], page 31.
- See Section 2.21 [Directories trunk Identity Themes Models Default Distro 5 Grub], page 30.
- See Section 2.22 [Directories trunk Identity Themes Models Default Distro 5 Gsplash], page 30.
- See Section 2.24 [Directories trunk Identity Themes Models Default Distro 5 Ksplash], page 31.

See also

- Section 2.16 [Directories trunk Identity Themes Models Default Distro], page 27.
- Section 2.14 [Directories trunk Identity Themes Models Default], page 26.
- Section 2.13 [Directories trunk Identity Themes Models], page 25.
- Section 2.12 [Directories trunk Identity Themes], page 24.
- Section 2.4 [Directories trunk Identity], page 14.
- Section 2.3 [Directories trunk], page 14.

2.18 The 'trunk/Identity/Themes/Models/Default/Distro/5/Anaconda' Directory

Goals

- ...

Description

Usage

See also

2.19 The ‘trunk/Identity/Themes/Models/Default/Distro/5/Firstboot’ Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

2.20 The ‘trunk/Identity/Themes/Models/Default/Distro/5/Gdm’ Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

2.21 The ‘trunk/Identity/Themes/Models/Default/Distro/5/Grub’ Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

2.22 The ‘trunk/Identity/Themes/Models/Default/Distro/5/Gsplash’ Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

2.23 The 'trunk/Identity/Themes/Models/Default/Distro/5/Kdm' Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

2.24 The 'trunk/Identity/Themes/Models/Default/Distro/5/Ksplash' Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

2.25 The 'trunk/Identity/Themes/Models/Default/Distro/5/Rhgb' Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

2.26 The 'trunk/Identity/Themes/Models/Default/Distro/5/Syslinux' Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

- ...

2.27 The '`trunk/Identity/Themes/Models/Default/Posters`' Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

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

Goals

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

- Organize CentOS themes' artistic motifs.

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
```

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/`).

See also

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

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

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

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

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

Each artistic motif directory represents just one unique artistic motif.

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

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

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

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

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

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

Goals

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

Description

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

The flame filter of Gimp can produce stunning, randomly generated fractal patterns. The flame filter of Gimp gives us a great oportunity to reduce the time used to produce new artistic motifs, because of its "randomly generated" nature. Once the artistic motif be created, it is propagated through all visual manifestations of CentOS Project corporate visual identity using the '`centos-art.sh`' script (see Section 2.36 [Directories trunk Scripts], 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 oursleves inside the monolithic conception we are trying to be attached to (i.e., one unique name, and one unique visual style for all visual manifestations).

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

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

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

> **Important** When we design background images, which are considered part of the same theme, it is essential to use the same design pattern always. This is what makes theme images to be visually connected among themeselves, and so, the reason we use to define the word "theme" as: a set of images visually connected among themeselves.

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

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

See also

- See Section 2.28 [Directories trunk Identity Themes Motifs], page 32.
- See Section 2.12 [Directories trunk Identity Themes], page 24.
- See Section 2.4 [Directories trunk Identity], page 14.
- See Section 2.3 [Directories trunk], page 14.

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

Goals

Description

- ...

Usage

- ...

See also

2.31 The 'trunk/Identity/Themes/Motifs/Pipes' Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

2.32 The 'trunk/Identity/Themes/Motifs/TreeFlower' Directory

Goals

Description

Usage

See also

2.33 The 'trunk/Identity/Webenv' Directory

Goals

- ...

Description

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

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

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

The CentOS project is attached to a monolithic corporate visual identity (see Section 2.4 [Directories trunk Identity], page 14), where all visual manifestations have one unique name

and one unique visual style. This way, the CentOS web environment has one unique name (the CentOS brand) and one unique visual style (the CentOS default theme) for all its visual manifestations, the web applications in this case.

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

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

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

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

Design model (without ads)

Design model (with ads)

HTML definitions

Controlling visual style

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

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

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

Producing visual style

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

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

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

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

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

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

Navigation

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

Development and release cycle

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

Download

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

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

Prepare

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

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

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

Customize

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

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

The command above creates the following structure:

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

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

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

Build packages

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

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

Release for testing

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

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

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

'webapp1-0.0.1-webenv-1.rpm', when it be fixed the new package will be named 'webapp1-0.0.1-webenv-2.rpm', and so on.

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

Release for production

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

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

The [webenv-test] repository

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

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

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
```

Priority configuration

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

Usage

- ...

See also

2.34 The 'trunk/Locales' Directory

Goals

The 'trunk/Locales' directory structure provides the localization work line and its main goal is provide the translation messages required to produce content in different languages.

Description

Translation messages inside the repository are stored as portable objects (e.g., .po, .pot) and machine objects (.mo) under '`trunk/Locales`' directory structure.

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

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

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

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

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

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

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

Usage

- ...

See also

Section 2.39 [Directories trunk Scripts Functions Locale], page 49

2.35 The '`trunk/Manual`' Directory

Goals

This '`trunk/Manual`' directory structure provides the documentation work line. The main goal of documentation work line is describe what each directory inside the CentOS Artwork Repository is for, the conceptual ideas behind them and, if possible, how automation scripts make use of them.

Description

- ...

Usage

- ...

See also

2.36 The 'trunk/Scripts' Directory

Goals

This section provides the automation work line. The automation work line exists to standardize content production in CentOS Artwork Repository. There is no need to type several tasks, time after time, if they can be programmed into just one executable script.

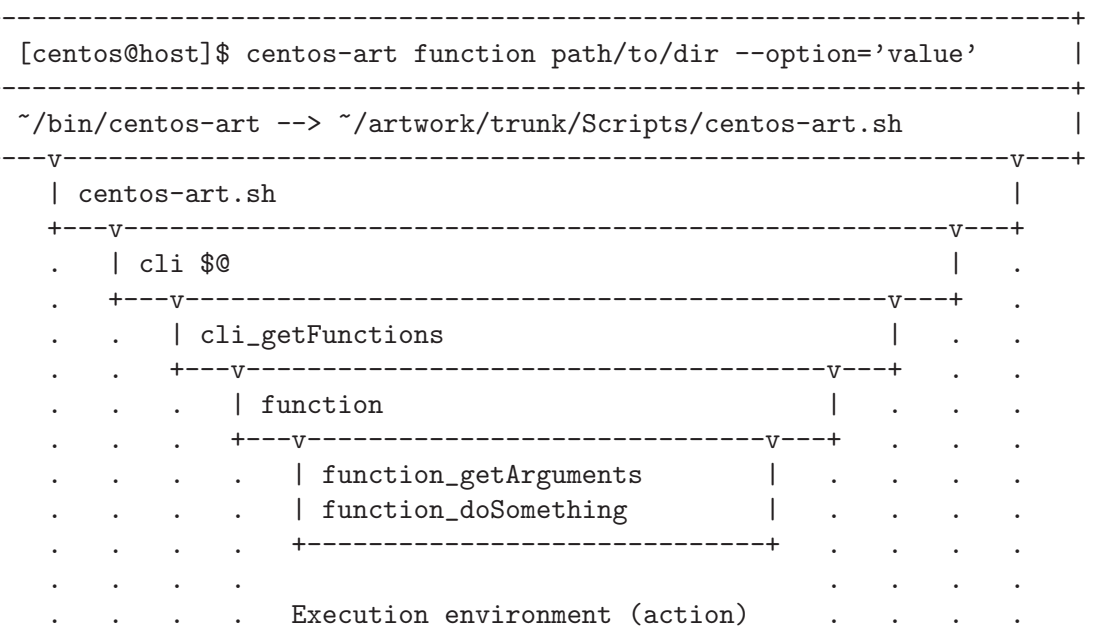
In this section you'll find how to organize and extend the `centos-art.sh` script, a bash scripts specially designed to automate most frequent tasks in the repository (e.g., image rendition, documenting directory structures, translating content, etc.). If you can't resist the idea of automating repeatable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

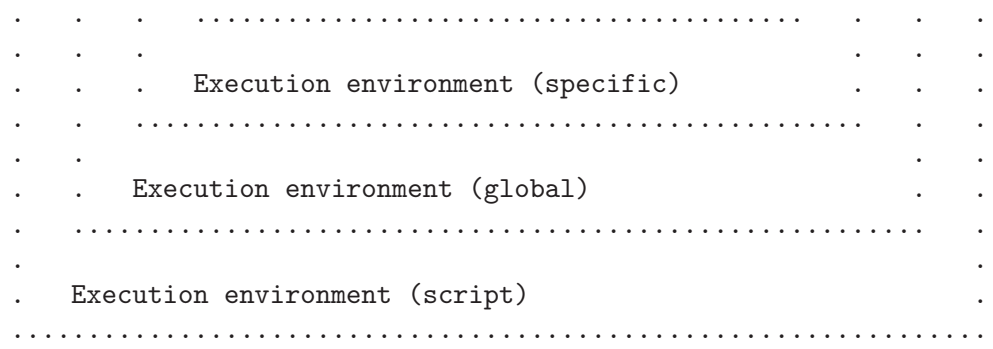
Description

The best way to understand the `centos-art.sh` script is studying and improving its source code. However, as start point, you may prefer to read an introductory resume before diving into the source code details. In this section we identify the different parts the `centos-art.sh` script is made of and how these parts interact one another.

Execution environments

The `centos-art.sh` script is basically made of four execution environments which are named *script*, *global*, *specific* and *action*. These execution environments are nested one into another and provide different definition levels for variables and functions. In this design, variables and functions defined in higher execution environments are available on lower execution environments, but variables and functions defined in lower execution environments are not available for higher execution enviroments.





The script execution environment exists to provide script definitions that can't be set anywhere else inside the script. Example of such definitions include initialization of internationalization through `gettext` program, script personal information and initialization of global functionalities.

The global execution environment exists to provide definitions that can't be set anywhere else inside the script. Example of such definitions include initialization of functionalities (e.g., `cli_printMessage`, `cli_getCurrentLocale`, `cli_checkFiles`, etc.) and variables (e.g., *FUNCNAM*, *FUNCDIR*, *FUNCDIRNAM*, *ARGUMENTS*, etc.) that can be both used on specific and action execution environments, only.

The specific execution environment exists to provide definitions that can't be set anywhere else inside the script. Example of such definitions include initialization of specifc functionalities (e.g., `render`, `help`, `locale`, etc.) and specific variables (*ACTIONNAM*, *ACTIONVAL*, etc.) that can be used on action execution environment only.

The action execution environment exists to perform the script actions themselves. It is here where we perform content rendition, content documentation, content localization and whatever action you plan for the `centos-art.sh` script to perform. For example, if you passed the `render` value as first argument to `centos-art.sh` command-line, the script performs the content rendition action through the `render` function which is defined in the '`render.sh`' file under '`trunk/Scripts/Functions/Render`' directory. Is there, inside `render` functionality were the action execution environment takes place exactly.

Command-line interface

When the `centos-art` command is executed in a bash terminal, the bash interpreter uses the `PATH` environment variable to find where such command is. In order to run the `centos-art`, it must exist either as a link to an executable file or an executable file by its own, in any of the paths provided by `PATH` environment variable. Otherwise, the bash interpreter will print an error message and prompt you back to type a valid command.

By default, after installing The CentOS Distribution, there is no `centos-art` command available in the `PATH` environment variable for you to execute. The `centos-art` command is made available in your workstation as result of executing the `prepare` functionality of `centos-art.sh` script (see Section 2.40 [Directories trunk Scripts Functions Prepare], page 50) which requires you had previously downloaded a working copy of CentOS Artwork Repository in your workstation.

When the `centos-art` is executed, the first positional parameter passed is required and represents the name of the function you want to perform (e.g., `render` for content rendition, `locale` for content localization, etc.). Beyond the first positional parameter you can provide either option or non-option parameters in no specific order. There are also, option parameters with arguments and without arguments. Frequently, non-option paramters are used to specify the path location inside the repository where the function will be performed in (e.g., the directory structure do you want to produce content for) and option parameters to specify how such functionality is performed (e.g., do you want to go quietly? do you want to do filtering? etc.).

```
    A         B         C               D           E
---------- ------- ----------- ---------------- -------
centos-art funcnam path/to/dir --filter='regex' --quiet
---------- ------- ----------- ---------------- -------

    A = The centos-art.sh script command-line.
    B = The centos-art.sh function name.
    C = Non-option parameter.
    D = Option parameter (with argument).
    E = Option parameter (without argument).
```

Parsing command-line options

The action of parsing options is performed through `getopt` and results particularly interesting. `getopt` breaks up (parse) options in command lines and checks for legal options using the GNU `getopt` routines to do this. One important consideration on `centos-art.sh` script design is that positional parameters are retrived in the `cli` function but parsed on each specific function, individually. There isn't a big parsing definition to cover all specific functions, but one parsing definitions for each specific functions.

Usage

- See Section 2.37 [Directories trunk Scripts Functions], page 44.

See also

- Section 2.3 [Directories trunk], page 14

2.37 The 'trunk/Scripts/Functions' Directory

Goals

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

Description

The specific functions of '`centos-art.sh`' script are designed with the "Software Toolbox" philosophy (See Info file '`coreutils.info`', node '`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/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/Functions/Render`'.

Creating the `greet` functionality

To better understand how to design specific functions for '`centos-art.sh`' script, let's create the `greet` functionality which only goal is to print out different kind of greetings to your screen. The `greet` functionality will be set using the follwiing directory structure:

```
trunk/Scripts/Functions/Greet   <-- The source location of greet function.
|-- greet_getArguments.sh       <-- Defines command-line interface.
|-- greet_sayGoodbye.sh         <-- Defines specific action.
|-- greet_sayHello.sh           <-- Defines specific action.
```

```
‘-- greet.sh                    <-- Defines function initialization.
```

The ‘greet.sh’ file contains the initialization script of greet functionality. It is the first file loaded from function source location by centos-art.sh script when it is executed using the greet functionality as first argument.

Inside ‘centos-art.sh’ script, as convenction, each function script has one top commentary, followed by one blank line, and then one function defintion below it only. The top commentary has the function description, one-line for copyright notice 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— and the Id keyword of Subversion which is later expanded by svn propset command. In our example, the top comment of 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., 675 Mass Ave, Cambridge, MA 02139, USA.
# ------------------------------------------------------------------------
# $Id$
# ------------------------------------------------------------------------

function greet {

    # Define command-line interface.
    greet_getArguments

    # Execute action name.
    if [[ $ACTIONNAM =~ "^${FUNCNAM}_[A-Za-z]+$" ]];then
        eval $ACTIONNAM
    else
        cli_printMessage "‘gettext "A valid action is required."‘" ’AsErrorLine’
        cli_printMessage "${FUNCDIRNAM}" ’AsToKnowMoreLine’
    fi

}
```

The first definition inside `greet` function is for variables that will be available along the whole execution environment of `greet` function. This time we didn't define any variable here so, we continued with definition of command-line interface, through `greet_getArguments` function.

The command-line interface of `greet` functionality defines how to interpret arguments passed from `centos-art.sh` script command-line. Inside `centos-art.sh` script, the interpretation of arguments passed through its command-line takes place by mean of `getopt` command and is written as the following code example describes:

```
function greet_getArguments {

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

    # Define long options we want to support.
    local ARGSL="hello:,bye:,quiet"

    # Redefine ARGUMENTS variable using getopt output.
    cli_doParseArguments

    # Redefine positional parameters using ARGUMENTS variable.
    eval set -- "$ARGUMENTS"

    # Look for options passed through command-line.
    while true; do

        case "$1" in

            --hello )
                ACTIONNAM="${FUNCNAM}_sayHello"
                ACTIONVAL="$2"
                shift 2
                ;;

            --bye )
                ACTIONNAM="${FUNCNAM}_sayGoodbye"
                ACTIONVAL="$2"
                shift 2
                ;;

            -- )
                # Remove the '--' argument from the list of arguments
                # in order for processing non-option arguments
                # correctly. At this point all option arguments have
                # been processed already but the '--' argument still
                # remains to mark ending of option arguments and
                # begining of non-option arguments. The '--' argument
                # needs to be removed here in order to avoid
                # centos-art.sh script to process it as a path inside
                # the repository, which obviously is not.
                shift 1
                break
                ;;
```

```
        esac
    done

    # Redefine ARGUMENTS variable using current positional parameters.
    cli_doParseArgumentsReDef "$@"

}
```

The `greet_sayHello` and `greet_sayGoodbye` 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_sayHello {

    cli_printMessage "`gettext "Hello"`, $ACTIONVAL"

}
```

The `greet_sayHello` function definition is stored in '`greet_sayHello.sh`' function script.

```
function greet_sayGoodbye {

    cli_printMessage "`gettext "Goodbye"`, $ACTIONVAL"

}
```

The `greet_sayGoodbye` function definition is stored in the '`greet_sayGoodbye.sh`' function script.

Executing the `greet` functionality

To execute 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 after it. 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. Likewise, if you need to change the way either the hello or goodbye messages are printed out, you can modifie the functions `greet_sayHello` and `greet_sayGoodbye`, respectively.

Documenting the `greet` functionality

Now that `greet` functionality works as we expect, it is time to document it. To document functionalities inside `centos-art.sh` script we use the function directory path as argument to the `help` functionality (see Section 2.38 [Directories trunk Scripts Functions Help], page 49) of '`centos-art.sh`' script, just as the following command illustrates:

```
centos-art help --edit trunk/Scripts/Functions/Greet
```

The function documentation helps to understand how the function really works and how it should be used. Also, when `centos-art.sh` script ends because an error, the documentation entry related to the functionality being currently executed is used as vehicle to communicate the user what is the correct way of using the functionality.

Localizing the greet functionality

Now that `greet` functionality has been documented, it is time to localize its output messages. Localizing specific functionalities of `centos-art.sh` script takes place as part of `centos-art.sh` script localization itself which is performed by applying the path '`trunk/Scripts`' to the `locale` functionality of `centos-art.sh` script.

As the `greet` functionality added new translatable strings to the `centos-art.sh` script, it is required to update the translation messages firstly, to add the new translatable strings from `greet` functionality to `centos-art.sh` script translation messages and then, edit the translation messages of `centos-art.sh` script to localize the new translatable strings that have been added. To achieve this, execute the following two commands:

```
centos-art locale --update trunk/Scripts
centos-art locale --edit trunk/Scripts
```

> **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, before executing the `locale` functionality of `centos-art.sh` script.

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

Extending the greet functionality

In the `greet` functionality we've described so far, we only use `cli_printMessage` 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 argument, you could use it to retrive a list of files from therein and process them. If the list of files turns too long or you just want to control which files to process, so you could add another argument in the form '`--filter='regex''` and reduce the list of files to process using a regular expression pattern.

In case you consider to extend the `greet` functionality to do something different but print out grettings, consider changing the function name from `greet` to something more appropriate, as well. The name change must be coherent with the actions the new function is designed to perform.

If you doubt what name is better for your functionality, write to centos-devel@centos.org mailing list, explain what your functionality intends to do and request suggestion about what name would be more appropriate for it. That would be also very convenient for you, in order to evaluate the purposes of your function and what the community thinks about it. It is a way for you to gather ideas that help you to write using the community feeling as base.

If your function passes the community evaluation, that is a good sign for you to start/keep writing it. However, if it doesn't, it is time for you to rethink what you are doing and ask again until it passes the community evaluation. You can considered you've passed the community evaluation when after proposing your idea, you get a considerable amount of possitve responses for what you are doing, specially if those responses come from community leaders.

It is very hard to do something useful for a community of people without any point of contact with that community you are trying to do things for. How could you know you are doing something that is needed if you don't know what the needs are? So, explore the community needs first, define them, work them out and repeat the process time after time, even when you might think the need has been already satisfied. At that point, surely, you'll find smaller needs that need to be satisfied, as well.

Conclusions

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

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

Usage

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

- See Section 2.38 [Directories trunk Scripts Functions Help], page 49.
- See Section 2.39 [Directories trunk Scripts Functions Locale], page 49.
- See Section 2.40 [Directories trunk Scripts Functions Prepare], page 50.
- See Section 2.41 [Directories trunk Scripts Functions Render], page 54.
- See Section 2.42 [Directories trunk Scripts Functions Tuneup], page 64.

See also

- Section 2.36 [Directories trunk Scripts], page 42
- Section 2.3 [Directories trunk], page 14

2.38 The '`trunk/Scripts/Functions/Help`' Directory

Goals

- ...

Description

- ...

Usage

- ...

See also

2.39 The '`trunk/Scripts/Functions/Locale`' Directory

Goals

- ...

Description

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

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

The *CLI_LANG* variable discribes the locale language used to output messages inside `centos-art` command. The locale language used inside `centos-art` command is taken from

the `LANG` environment variable. The *CLI_LANG* variable has the '`LL_CC`' format, where '`LL`' is a language code from the ISO-639 standard, and '`CC`' a country code from the ISO-3166 standard.

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

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

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

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

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

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

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

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.

See also

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

Goals

This section describes the `prepare` functionality of `centos-art.sh` script and the preliminar steps you need to follow in order to get your workstation ready for using the CentOS Artwork Repository.

Description

The `prepare` functionality of `centos-art.sh` script verifies all the information required by the working copy of CentOS Artwork Repository (e.g., packages of software, symbolic links, etc.) does exist in your workstation.

The `prepare` functionality of `centos-art.sh` script is part of the CentOS Artwork Repository. So, in order to execute the `prepare` functionality of `centos-art.sh` script you need to

have access to a working copy of CentOS Artwork Repository, first. Working copies of CentOS Artwork Repository are downloaded from the source repository and made available to you by mean of workstations. A workstation is a computer that you install and configure (prepare) to do something. In this case, you pick a computer and prepare it for working on the CentOS Artwork Repository.

Installing the workstation

Installing the workstation is the first step you need to perform for using the CentOS Artwork Repository. In this step you make your computer functional through an operating system. In this case, The Community Enterprise Operating System; which is also know as The CentOS Distribution or just CentOS, for short.

To install The CentOS Distribution you need to have the installation media somehow (e.g., CDs, DVDs, Pendrives, etc.). There are several different ways to perform the installation process of CentOS distribution, but generally, you put the installation media in your media reader, boot the computer from it, and follow the installer intructions. That simple.

If you don't have the installation media of CentOS distribution, you need to download the ISO files related to the media you plan to use (e.g., CD or DVD) and then create the installation media by yourself. The CentOS Distribution ISO files can be downloaded from http://mirrors.centos.org/ and, if you chosen CD or DVD as your prefered installation medium, you can burn the ISO files using the `K3B` application to create the installation media you'll use. Of course, in order to download the ISO files and create the installation media, you need to have an already installed CentOS workstation where you can realized all the work.

Configuring the workstation

Once you've installed the workstation and it is up and running, login as '`root`' user, create a username (e.g., '`centos`') and set a password for it. This is the username you must use for everyday work inside your working copy of the CentOS Artwork Repository.

> **Caution** Do not use ever the '`root`' username for your everyday work inside the working copy of CentOS Artwork Repository. It is dangerous and might provoke unreversable damages on your workstation.

Once you've created the username for everyday work, there are some environment variables that you can customize to fit your personal needs (e.g., the text editor you'll use, the language the automation scripts will use to print and translate output messages, the time correction for your location, etc.). To customize these variables you need to edit your profile file (i.e., '`~/.bash_profile`') and set the redefinition there. Notice that you may need to logout and then do login again in order for the new variable values to take effect.

Default text editor:

The default text editor information is contrlled by the `EDITOR` environment variable. The '`centos-art.sh`' script uses the default text editor to edit subversion pre-commit messages, translation files, documentation 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 the `EDITOR` environment variable, the '`centos-art.sh`' script uses '`/usr/bin/vim`' text editor, the one installed by default in The CentOS Distribution.

Default locale information:

The default locale information is controlled by the `LANG` environment variable. This variable is initially set in the configuration process of CentOS distribution installer, 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 determine what language to use for printing output messages. Moreover, the `locale` functionality uses the `LANG` to determine what translation messages to udpate or edit.

Default time zone representation:

The time zone representation is a time correction applied to the system time (stored in the BIOS clock) based on your country location. This correction is specially useful to distributed computers around the world that work together and need to be syncronized in time to know when things happened.

The CentOS Artwork Repository is made of one server and several workstations spread around the world. In order for all these workstations to know when changes in the server took place, it is required that all the workstations set their system clocks to use the same time information (i.e., UTC (Coordinated Universal Time)) and set the time correction for their countries in the operating system. Otherwise, it'd be hard to know when something exactly happened.

Generally, setting the time information is a straight-forward task and configuration tools provided by The CentOS Distribution do cover time correction for most of the countries around the world. However, if you need a time precision not provided by any of the date and time configuration tools provided by The CentOS Distribution then, you need to use the `TZ` environment variable to correct the time information by yourself. The format of `TZ` environment variable is described in '`tzset(3)`' manual page.

Downloading the working copy

Once you've configured the workstation, it is time to download the working copy of CentOS Artwork Repository.

To download the working copy of CentOS Artwork Repository you need to login as your everyday work username (e.g., '`centos`') and use the Subversion client to bring all the files you need to work with down from the source location of CentOS Artwork Repository (`https://projects.centos.org/svn/artwork/`) to your workstation, just as the following command describes:

```
svn co https://projects.centos.org/svn/artwork ~/
```

This command will create the working copy of CentOS Artwork Repository in your workstation, specifically in the '`/home/centos/artwork`' directory. Note that you only need to execute this command once. After that, to keep your working copy up to date, you use the Subversion `update` command instead.

> **Tip** In the condition that you don't have Subversion client installed in the workstation, then you can install it using the command:
>
> ```
> sudo yum install subversion
> ```

Configuring the working copy

Once you have a working copy of CentOS Artwork Repository in your workstation, you can go and run the `prepare` functionality of `centos-art.sh` script to realize the remaining configuration stuff.

Assuming this is the very first time you run the `centos-art.sh` script, you'll find that there is no `centos-art` command-line interface for it in your workstation. This is correct. In order to

have the `centos-art` command-line in your workstation, you need to run the `centos-art.sh` script using its absolute path:

```
~/artwork/trunk/Scripts/centos-art.sh prepare [OPTIONS]
```

Assuming you've already run the `prepare` functionality before, there is no need for you to use the absolute path again. Instead, you can use the `centos-art` command-line interface directly, as the following example describes:

```
centos-art.sh prepare [OPTIONS]
```

Notice that you can execute the `prepare` functionality more than once. This is specially useful to keep the link information syncronized. For example, considering you've added new brushes to or removed old brushes from your working copy of CentOS Artwork Repository, the link information related to those files need to be updated in the '`~/.gimp-2.2/brushes`' directory too, in a way that reflect the addition/deletion change that took place in your working copy. The same is true for fonts, patterns and palettes components.

Usage

Synopsis

`centos-art prepare [OPTIONS]`

Options

'`--packages`'

Verify existence of software packages and install them if they aren't installed yet.

This option provides the software required to manipulate files inside the repository (e.g., image files, documentation files, translation files, script files, etc.). Most of the software packages required by the CentOS Artwork Repository are available on The CentOS Distribution and can be installed using The CentOS Distribution installation media. The only exception is the '`inkscape`', the package you use to manipulate SVG (Scalable Vector Graphics) files. The '`inkscape`' package isn't inside The CentOS Distribution or any of The CentOS Project repositories neither, so you need to install it from a third party repositories like '`RPMForge`' or '`EPEL`'.

'`--link`'

Update connection between working copy and workstation through symbolic links.

This option creates the `centos-art` command-line interface of `centos-art.sh` script through a symbolic link. There is no need for you to type the full path to `centos-art.sh` script each time you need to execute it. Instead, you use the `centos-art` command which is much shorter and faster to type.

This option connects design compenents like fonts, brushes, patterns and palettes inside your working copy of CentOS Artwork Repository with programs like GIMP (GNU Image Manipulation Program) and Inkscape outside it. This way, all your modifications on these components will take place inside the repository and will be shared to all other working copies the next time you commit the changes up to source repository.

This option standardizes width, tabulation, indentation, and line numbering for text editors in your workstation. The configuration file where these definitions are set, is versioned inside your working copy and linked from the appropriate place in the workstation to make it valid to your default text editor.

'`--environment`'

Print the name and value of some of the environment variables used by `centos-art.sh` scripts.

‘`--quiet`’

Supress all output messages except errors. This option assumes an affirmative response to all questions so, take care when using it.

See also

- Section 2.37 [Directories trunk Scripts Functions], page 44
- Section 2.36 [Directories trunk Scripts], page 42
- Section 2.3 [Directories trunk], page 14
- The CentOS Repositories, to know how to configure third party repositories inside The CentOS Distribution.

2.41 The ‘`trunk/Scripts/Functions/Render`’ Directory

Rendition is the action through we produce content (images specially) inside The CentOS Artwork Repository. In order to produce content through rendition, the directory structure where that content is stored in must be renderable (i.e., the rendition action must be applied to a directory structure considered renderable). Presently, renderable directories are stored under ‘`trunk/Identity/Images`’ and ‘`trunk/Identity/Themes/Motifs`’ directories only.

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

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

Inside ‘`trunk/Identity`’ directory structure, you can find that base-rendition, post-rendition and last-rendition flows can be combined to build *directory-specific* rendition. The directory-specific rendition exists to process specific renderable directories in very specific ways. Using directory-specific rendition speeds up production of different components like Syslinux, Grub, Gdm, Kdm and Ksplash that require intermediate formats or even several independent files, in order for the final content to be created.

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).

Design template without translation

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

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

One renderable identity directory structure:

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

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

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

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

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

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

area you (as designer) want to export as PNG (Portable Network Graphic) image file.

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

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

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

See Section 2.4 [Directories trunk Identity], page 14, for more information.

One translation directory structure:

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

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

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

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

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

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

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

One pre-rendition configuration script:

In order to make an identity directory structure renderable, a pre-rendition configuration script should exist for it. The pre-rendition configuration script specifies what type of rendition does '`centos-art.sh`' will perform over the identity directory structure and how does it do that.

```
                                                | The bond component
                                                |----------------->|
trunk/Scripts/Bash/Functions/Render/Config/Identity/Path/To/Dir
```

```
'-- render.conf.sh
```

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

```
function render_loadConfig {

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

}
```

Since translation directory structure is empty, 'centos-art.sh' assumes a "design template without translation" configuration to produce untranslated images.

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

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

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

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

Design template with translation (one-to-one)

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

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

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

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

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

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

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

Design template with translation (optimized)

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

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

let us use as many assignment combinations as translation files and design templates we are working with.

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

For example, consider the following configuration:

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

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

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

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

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

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

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

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

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

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

```
function render_loadConfig {

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

    # Define matching list.
```

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

}
```

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

Design template with translation (optimized+flexibility)

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

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

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

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

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

For example, consider the following configuration:

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

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

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

```
‘-- list.svg
```

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

One entry under ‘trunk/Identity/Themes/Motifs/’:

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

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

One entry under ‘trunk/Translations/’:

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

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

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

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

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

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

```
function render_loadConfig {

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

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

    # Deifne theme model.
    THEMEMODEL='Default'

}
```

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

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.

Copying renderable directory structures

A renderable layout is formed by design models, design images, pre-rendition configuration scripts and translations files. This way, when we say to duplicate rendition stuff we are saying to duplicate these four directory structures (i.e., design models, design images, pre-rendition configuration scripts, and related translations files).

When we duplicate directories, inside ‘trunk/Identity’ directory structure, we need to be aware of renderable layout described above and the source location used to perform the duplication action. The source location is relevant to centos-art.sh script in order to determine the required auxiliar information inside directory structures that need to be copied too (otherwise

we may end up with orphan directory structures unable to be rendered, due the absence of required information).

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

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

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

Design templates + No translation:

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

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

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

Renderable layout 2:

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

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

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

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

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

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

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

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

- When translations and/or pre-rendition configuration scripts are found inside the source directory structure, the centos-art.sh script loads common auxiliar directories first and later

specific auxiliar directories. This way, identity rendition of source locations can be customized idividually over the base of common default values.

- The specific auxiliar directories are optional.

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

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

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

Usage

- ...

See also

2.42 The 'trunk/Scripts/Functions/Tuneup' Directory

Goals

- ...

Description

Shell Script Files

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

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

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

```
13| function doSomething {
14|
15| }
```

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

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

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

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

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

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

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

```
28| }
```

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

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

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

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

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

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
                         675 Mass Ave, Cambridge, MA 02139, 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-devel@centos-art.sh` mailing list.

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

SVG Files

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

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

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

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

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

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

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

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

XHTML Files

- ...

Usage

- ...

See also

- ...

Index

List of Figures