



Red Hat Enterprise Linux 8

Customizing Anaconda

Changing the installer appearance and creating custom add-ons on Red Hat Enterprise Linux 8

Red Hat Enterprise Linux 8 Customizing Anaconda

Changing the installer appearance and creating custom add-ons on Red Hat Enterprise Linux 8

Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Anaconda is the installer used by Red Hat Enterprise Linux, Fedora, and their derivatives. This document contains information necessary for customizing it.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	3
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	4
CHAPTER 1. INTRODUCTION TO ANACONDA CUSTOMIZATION	5
1.1. INTRODUCTION TO ANACONDA CUSTOMIZATION	5
1.2. PERFORMING THE PRE-CUSTOMIZATION TASKS	5
1.2.1. Working with ISO images	5
1.2.2. Downloading RH boot images	5
1.2.3. Extracting Red Hat Enterprise Linux boot images	6
CHAPTER 2. CUSTOMIZING THE BOOT MENU	7
2.1. CUSTOMIZING THE BOOT MENU	7
2.2. SYSTEMS WITH BIOS FIRMWARE	7
2.3. SYSTEMS WITH UEFI FIRMWARE	10
CHAPTER 3. BRANDING AND CHROMING THE GRAPHICAL USER INTERFACE	12
3.1. CUSTOMIZING GRAPHICAL ELEMENTS	12
3.2. CUSTOMIZING THE PRODUCT NAME	13
3.3. CUSTOMIZING THE DEFAULT CONFIGURATION	14
3.3.1. Configuring the default configuration files	14
3.3.2. Configuring the product configuration files	19
3.3.3. Configuring the custom configuration files	19
CHAPTER 4. DEVELOPING INSTALLER ADD-ONS	21
4.1. INTRODUCTION TO ANACONDA AND ADD-ONS	21
4.2. ANACONDA ARCHITECTURE	21
4.3. ANACONDA USER INTERFACE	23
4.4. COMMUNICATION ACROSS ANACONDA THREADS	24
4.5. ANACONDA MODULES AND D-BUS LIBRARY	24
4.6. THE HELLO WORLD ADDON EXAMPLE	25
4.7. ANACONDA ADD-ON STRUCTURE	25
4.8. ANACONDA SERVICES AND CONFIGURATION FILES	26
4.9. DEPLOYING AND TESTING AN ANACONDA ADD-ON	27
CHAPTER 5. COMPLETING POST CUSTOMIZATION TASKS	29
5.1. CREATING A PRODUCT IMG FILE	29
5.2. CREATING CUSTOM BOOT IMAGES	31

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages:
 1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
 2. Use your mouse cursor to highlight the part of text that you want to comment on.
 3. Click the **Add Feedback** pop-up that appears below the highlighted text.
 4. Follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
 1. Go to the [Bugzilla](#) website.
 2. As the Component, use **Documentation**.
 3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
 4. Click **Submit Bug**.

CHAPTER 1. INTRODUCTION TO ANACONDA CUSTOMIZATION

1.1. INTRODUCTION TO ANACONDA CUSTOMIZATION

The Red Hat Enterprise Linux and Fedora installation program, **Anaconda**, brings many improvements in its most recent versions. One of these improvements is enhanced customizability. You can now write add-ons to extend the base installer functionality, and change the appearance of the graphical user interface.

This document will explain how to customize the following:

- Boot menu - pre-configured options, color scheme and background
- Appearance of the graphical interface - logo, backgrounds, product name
- Installer functionality - add-ons which can enhance the installer by adding new Kickstart commands and new screens in the graphical and textual user interfaces

Also note that this document applies only to Red Hat Enterprise Linux 8 and Fedora 17 and later.



IMPORTANT

Procedures described in this book are written for Red Hat Enterprise Linux 8 or a similar system. On other systems, the tools and applications used (such as **genisoimage** for creating custom ISO images) may be different, and procedures may need to be adjusted.

1.2. PERFORMING THE PRE-CUSTOMIZATION TASKS

1.2.1. Working with ISO images

In this section, you will learn how to:

- Extract a Red Hat ISO.
- Create a new boot image containing your customizations.

1.2.2. Downloading RH boot images

Before you begin to customize the installer, download the Red Hat-provided boot images. You can obtain Red Hat Enterprise Linux 8 boot media from the [Red Hat Customer Portal](#) after login to your account.

**NOTE**

- Your account must have sufficient entitlements to download Red Hat Enterprise Linux 8 images.
- You must download either the **Binary DVD** or **Boot ISO** image and can use any of the image variants (Server or ComputeNode).
- You cannot customize the installer using the other available downloads, such as the KVM Guest Image or Supplementary DVD; other available downloads, such as the **KVM Guest Image** or **Supplementary DVD**.

For more information about the Binary DVD and Boot ISO downloads, see the [Red Hat Enterprise Linux 8 Performing an Advanced RHEL Installation](#).

1.2.3. Extracting Red Hat Enterprise Linux boot images

Perform the following procedure to extract the contents of a boot image.

Procedure

1. Ensure that the directory `/mnt/iso` exists and nothing is currently mounted there.
2. Mount the downloaded image.

```
# mount -t iso9660 -o loop path/to/image.iso /mnt/iso
```

Where `path/to/image.iso` is the path to the downloaded boot image.

3. Create a working directory where you want to place the contents of the ISO image.

```
$ mkdir /tmp/ISO
```

4. Copy all contents of the mounted image to your new working directory. Make sure to use the `-p` option to preserve file and directory permissions and ownership.

```
# cp -pRf /mnt/iso /tmp/ISO
```

5. Unmount the image.

```
# umount /mnt/iso
```

Additional resources

- For detailed download instructions and description of the Binary DVD and Boot ISO downloads, see the [Red Hat Enterprise Linux 8](#).

CHAPTER 2. CUSTOMIZING THE BOOT MENU

This section provides information about what the Boot menu customization is, and how to customize it.

Prerequisites:

For information about downloading and extracting Boot images, see [Section 1.2.3, “Extracting Red Hat Enterprise Linux boot images”](#)

The Boot menu customization involves the following high-level tasks:

1. Complete the prerequisites.
2. Customize the Boot menu.
3. Create a custom Boot image.

2.1. CUSTOMIZING THE BOOT MENU

The *Boot menu* is the menu which appears after you boot your system using an installation image. Normally, this menu allows you to choose between options such as **Install Red Hat Enterprise Linux**, **Boot from local drive** or **Rescue an installed system**. To customize the Boot menu, you can:

- Customize the default options.
- Add more options.
- Change the visual style (color and background).

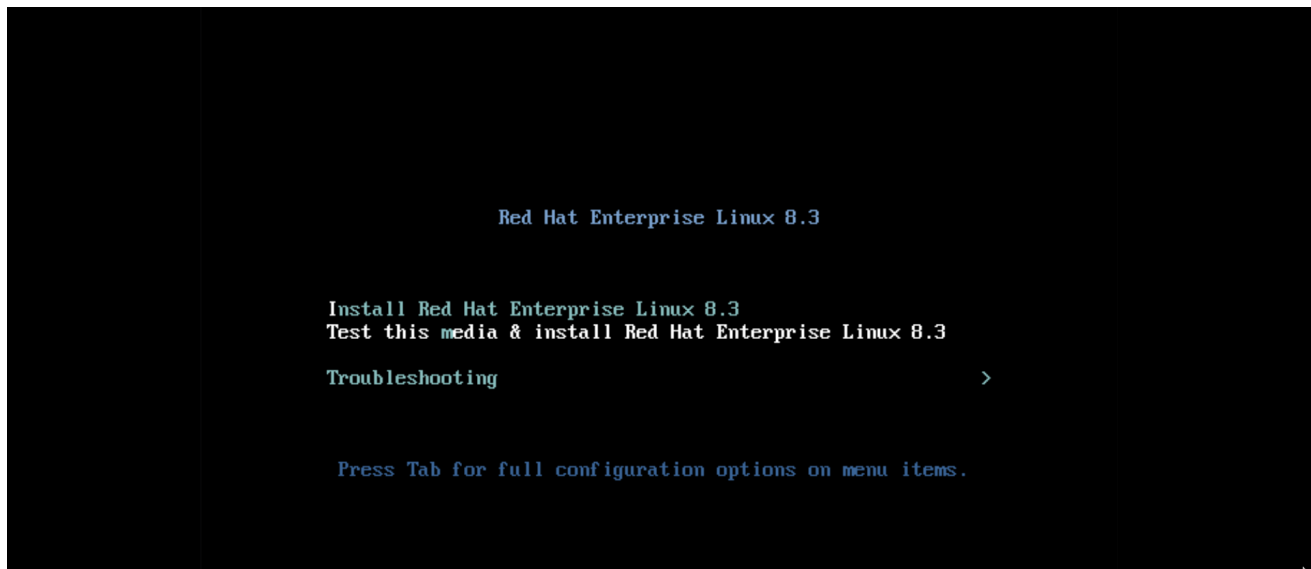
An installation media consists of **ISOLINUX** and **GRUB2** boot loaders. The **ISOLINUX** boot loader is used on systems with BIOS firmware, and the **GRUB2** boot loader is used on systems with UEFI firmware. Both the boot loaders are present on all Red Hat images for AMD64 and Intel 64 systems.

Customizing the boot menu options can especially be useful with Kickstart. Kickstart files must be provided to the installer before the installation begins. Normally, this is done by manually editing one of the existing boot options to add the **inst.ks=** boot option. You can add this option to one of the pre-configured entries, if you edit boot loader configuration files on the media.

2.2. SYSTEMS WITH BIOS FIRMWARE

The **ISOLINUX** boot loader is used on systems with BIOS firmware.

Figure 2.1. ISOLINUX Boot Menu



The `isolinux/isolinux.cfg` configuration file on the boot media contains directives for setting the color scheme and the menu structure (entries and submenus).

In the configuration file, the default menu entry for Red Hat Enterprise Linux, **Test this media & Install Red Hat Enterprise Linux 8**, is defined in the following block:

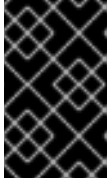
```
label check
  menu label Test this ^media & install Red Hat Enterprise Linux 8.3.0.
  menu default
  kernel vmlinuz
  append initrd=initrd.img inst.stage2=hd:LABEL=RHEL-8-2-0-BaseOS-x86_64 rd.live.check
  quiet
```

Where:

- **menu label** - determines how the entry will be named in the menu. The `^` character determines its keyboard shortcut (the **m** key).
- **menu default** - provides a default selection, even though it is not the first option in the list.
- **kernel** - loads the installer kernel. In most cases it should not be changed.
- **append** - contains additional kernel options. The **initrd=** and **inst.stage2** options are mandatory; you can add others. For information about the options that are applicable to **Anaconda** refer to [Red Hat Enterprise Linux 8 Performing a Standard RHEL Installation Guide](#).

One of the notable options is **inst.ks=**, which allows you to specify a location of a Kickstart file. You can place a Kickstart file on the boot ISO image and use the **inst.ks=** option to specify its location; for example, you can place a **kickstart.ks** file into the image's root directory and use **inst.ks=hd:LABEL=RHEL-8-2-0-BaseOS-x86_64:/kickstart.ks**.

You can also use **dracut** options which are listed on the **dracut.cmdline(8)** man page.



IMPORTANT

When using a disk label to refer to a certain drive (as seen in the `inst.stage2=hd:LABEL=RHEL-8-2-0-BaseOS-x86_64` option above), replace all spaces with `\x20`.

Other important options which are not included in the menu entry definition are:

- **timeout** - determines the time for which the boot menu is displayed before the default menu entry is automatically used. The default value is **600**, which means the menu is displayed for 60 seconds. Setting this value to **0** disables the timeout option.



NOTE

Setting the timeout to a low value such as **1** is useful when performing a headless installation. This helps to avoid the default timeout to finish.

- **menu begin** and **menu end** - determines a start and end of a *submenu* block, allowing you to add additional options such as troubleshooting and grouping them in a submenu. A simple submenu with two options (one to continue and one to go back to the main menu) looks similar to the following:

```

menu begin ^Troubleshooting
  menu title Troubleshooting

  label rescue
  menu label ^Rescue a Red Hat Enterprise Linux system
  kernel vmlinuz
  append initrd=initrd.img inst.stage2=hd:LABEL=RHEL-8-2-0-BaseOS-x86_64 rescue quiet

  menu separator

  label returntomain
  menu label Return to ^main menu
  menu exit

menu end

```

The submenu entry definitions are similar to normal menu entries, but grouped between **menu begin** and **menu end** statements. The **menu exit** line in the second option exits the submenu and returns to the main menu.

- **menu background** - the menu background can either be a solid color (see **menu color** below), or an image in a PNG, JPEG or LSS16 format. When using an image, make sure that its dimensions correspond to the resolution set using the **set resolution** statement. Default dimensions are 640x480.
- **menu color** - determines the color of a menu element. The full format is:

```
menu color element ansi foreground background shadow
```

Most important parts of this command are:

- *element* - determines which element the color will apply to.

- *foreground* and *background* - determine the actual colors. The colors are described using an **#AARRGGBB** notation in hexadecimal format determines opacity:
- **00** for fully transparent.
- **ff** for fully opaque.
- **menu help *textfile*** - creates a menu entry which, when selected, displays a help text file.

Additional resources

- For a complete list of **ISOLINUX** configuration file options, see the [Syslinux Wiki](#).

2.3. SYSTEMS WITH UEFI FIRMWARE

The **GRUB2** boot loader is used on systems with UEFI firmware.

The **EFI/BOOT/grub.cfg** configuration file on the boot media contains a list of preconfigured menu entries and other directives which controls the appearance and the Boot menu functionality.

In the configuration file, the default menu entry for Red Hat Enterprise Linux (**Test this media & install Red Hat Enterprise Linux 8.3.0**) is defined in the following block:

```
menuentry 'Test this media & install Red Hat Enterprise Linux 8.3' --class fedora --class gnu-linux
--class gnu --class os {
    linuxefi /images/pxeboot/vmlinuz inst.stage2=hd:LABEL=RHEL-8-3-0-BaseOS-x86_64
    rd.live.check quiet
    initrdefi /images/pxeboot/initrd.img
}
```

Where:

- **menuentry** - Defines the title of the entry. It is specified in single or double quotes (' or "). You can use the **--class** option to group menu entries into different *classes*, which can then be styled differently using **GRUB2** themes.



NOTE

As shown in the above example, you must enclose each menu entry definition in curly braces ({}).

- **linuxefi** - Defines the kernel that boots (**/images/pxeboot/vmlinuz** in the above example) and the other additional options, if any. You can customize these options to change the behavior of the boot entry. For details about the options that are applicable to **Anaconda**, refer to [Red Hat Enterprise Linux 8 Performing an advanced RHEL installation](#).

One of the notable options is **inst.ks=**, which allows you to specify a location of a Kickstart file. You can place a Kickstart file on the boot ISO image and use the **inst.ks=** option to specify its location; for example, you can place a **kickstart.ks** file into the image's root directory and use **inst.ks=hd:LABEL=RHEL-8-2-0-BaseOS-x86_64:/kickstart.ks**.

You can also use **dracut** options which are listed on the **dracut.cmdline(8)** man page.



IMPORTANT

When using a disk label to refer to a certain drive (as seen in the `inst.stage2=hd:LABEL=RHEL-8-2-0-BaseOS-x86_64` option above), replace all spaces with `\x20`.

- **initrdefi** - location of the initial RAM disk (initrd) image to be loaded.

Other options used in the **grub.cfg** configuration file are:

- **set timeout** - determines how long is the boot menu displayed before the default menu entry is automatically used. The default value is **60**, which means the menu is displayed for 60 seconds. Setting this value to **-1** disables the timeout completely.



NOTE

Setting the timeout to **0** is useful when performing a headless installation, because this setting immediately activates the default boot entry.

- **submenu** - A *submenu* block allows you to create a sub-menu and group some entries under it, instead of displaying them in the main menu. The **Troubleshooting** submenu in the default configuration contains entries for rescuing an existing system. The title of the entry is in single or double quotes (' or ").

The **submenu** block contains one or more **menuentry** definitions as described above, and the entire block is enclosed in curly braces ({}). For example:

```

submenu 'Submenu title' {
  menuentry 'Submenu option 1' {
    linuxefi /images/vmlinuz inst.stage2=hd:LABEL=RHEL-8-2-0-BaseOS-x86_64
    xdriver=vesa nomodeset quiet
    initrdefi /images/pxeboot/initrd.img
  }
  menuentry 'Submenu option 2' {
    linuxefi /images/vmlinuz inst.stage2=hd:LABEL=RHEL-8-2-0-BaseOS-x86_64 rescue quiet
    initrdefi /images/initrd.img
  }
}

```

- **set default** - Determines the default entry. The entry numbers start from **0**. If you want to make the *third* entry the default one, use **set default=2** and so on.
- **theme** - determines the directory which contains **GRUB2** theme files. You can use the themes to customize visual aspects of the boot loader - background, fonts, and colors of specific elements.

Additional resources

- For additional information about customizing the boot menu, see the [GNU GRUB Manual 2.00](#).
- For more general information about **GRUB2**, see the [Red Hat Enterprise Linux 8 Managing, monitoring and updating the kernel](#).

CHAPTER 3. BRANDING AND CHROMING THE GRAPHICAL USER INTERFACE

The customization of Anaconda user interface may include the customization of graphical elements and the customization of product name.

This section provides information about how to customize the graphical elements and the product name.

Prerequisites

1. You have downloaded and extracted the ISO image.
2. You have created your own branding material.

For information about downloading and extracting boot images, see [Section 1.2.3, “Extracting Red Hat Enterprise Linux boot images”](#)

The user interface customization involves the following high-level tasks:

1. Complete the prerequisites.
2. Create custom branding material (if you plan to customize the graphical elements)
3. Customize the graphical elements (if you plan to customize it)
4. Customize the product name (if you plan to customize it)
5. Create a product.img file
6. Create a custom Boot image



NOTE

To create the custom branding material, first refer to the default graphical element files type and dimensions. You can accordingly create the custom material. Details about default graphical elements are available in the sample files that are provided in the [Section 3.1, “Customizing graphical elements”](#) section.

3.1. CUSTOMIZING GRAPHICAL ELEMENTS

To customize the graphical elements, you can modify or replace the customisable elements with the custom branded material, and update the container files.

The customisable graphical elements of the installer are stored in the `/usr/share/anaconda/pixmaps/` directory in the installer runtime file system. This directory contains the following customisable files:

```
pixmaps
├── anaconda-password-show-off.svg
├── anaconda-password-show-on.svg
├── right-arrow-icon.png
├── sidebar-bg.png
├── sidebar-logo.png
└── topbar-bg.png
```


Additionally, the `/usr/share/anaconda/` directory contains a CSS stylesheet named `anaconda-gtk.css`, which determines the file names and parameters of the main UI elements - the logo and the backgrounds for the sidebar and top bar. The file has the following contents that can be customized as per your requirement:

```
/* theme colors/images */

@define-color product_bg_color @redhat;

/* logo and sidebar classes */

.logo-sidebar {
    background-image: url('/usr/share/anaconda/pixmaps/sidebar-bg.png');
    background-color: @product_bg_color;
    background-repeat: no-repeat;
}

/* Add a logo to the sidebar */

.logo {
    background-image: url('/usr/share/anaconda/pixmaps/sidebar-logo.png');
    background-position: 50% 20px;
    background-repeat: no-repeat;
    background-color: transparent;
}

/* This is a placeholder to be filled by a product-specific logo. */

.product-logo {
    background-image: none;
    background-color: transparent;
}

AnacondaSpokeWindow #nav-box {
    background-color: @product_bg_color;
    background-image: url('/usr/share/anaconda/pixmaps/topbar-bg.png');
    background-repeat: no-repeat;
    color: white;
}
```

The most important part of the CSS file is the way in which it handles scaling based on resolution. The PNG image backgrounds do not scale, they are always displayed in their true dimensions. Instead, the backgrounds have a transparent background, and the stylesheet defines a matching background color on the `@define-color` line. Therefore, the background *images* "fade" into the background *color*, which means that the backgrounds work on all resolutions without a need for image scaling.

You could also change the `background-repeat` parameters to tile the background, or, if you are confident that every system you will be installing on will have the same display resolution, you can use background images which fill the entire bar.

Any of the files listed above can be customized. Once you do so, follow the instructions in Section 2.2, "Creating a product.img File" to create your own product.img with custom graphics, and then Section 2.3, "Creating Custom Boot Images" to create a new bootable ISO image with your changes included.

3.2. CUSTOMIZING THE PRODUCT NAME

To customize the product name, you must create a custom **.buildstamp file**. To do so, create a new file **.buildstamp.py** with the following content:

```
[Main]
Product=My Distribution
Version=8.3
BugURL=https://bugzilla.redhat.com/
IsFinal=True
UUID=202007011344.x86_64
[Compose]
Lorax=28.14.49-1
```

Change *My Distribution* to the name which you want to display in the installer.

After you create the custom **.buildstamp** file, follow the steps in [Section 5.1, “Creating a product img file”](#) section to create a new **product.img** file containing your customizations, and the [Section 5.2, “Creating custom boot images”](#) section to create a new bootable ISO file with your changes included.

3.3. CUSTOMIZING THE DEFAULT CONFIGURATION

You can create your own configuration file and use it to customize the configuration of the installer.

3.3.1. Configuring the default configuration files

You can write the Anaconda configuration files in the **.ini** file format. The Anaconda configuration file consists of sections, options and comments. Each section is defined by a **[section]** header, the comments starting with a **#** character and the keys to define the **options**. The resulting configuration file is processed with the **configparser** configuration file parser.

The default configuration file, located at **/etc/anaconda/anaconda.conf**, contains the documented sections and options that are supported. The file provides a full default configuration of the installer. You can modify the configuration of the product configuration files from **/etc/anaconda/product.d/** and the custom configuration files from **/etc/anaconda/conf.d/**.

The following configuration file describes the default configuration of RHEL 8.3:

```
[Anaconda]
# Run Anaconda in the debugging mode.
debug = False

# Enable Anaconda addons.
addons_enabled = True

# List of enabled Anaconda Dbus modules.
kickstart_modules =
    org.fedoraproject.Anaconda.Modules.Timezone
    org.fedoraproject.Anaconda.Modules.Network
    org.fedoraproject.Anaconda.Modules.Localization
    org.fedoraproject.Anaconda.Modules.Security
    org.fedoraproject.Anaconda.Modules.Users
    org.fedoraproject.Anaconda.Modules.Payloads
    org.fedoraproject.Anaconda.Modules.Storage
    org.fedoraproject.Anaconda.Modules.Services
    org.fedoraproject.Anaconda.Modules.Subscription
```

```
[Installation System]
# Should the installer show a warning about enabled SMT?
can_detect_enabled_smt = True

[Installation Target]
# Type of the installation target.
type = HARDWARE

# A path to the physical root of the target.
physical_root = /mnt/sysimage

# A path to the system root of the target.
system_root = /mnt/sysroot

[Network]
# Network device to be activated on boot if none was configured so.
# Valid values:
#
# NONE          No device
# DEFAULT_ROUTE_DEVICE  A default route device
# FIRST_WIRED_WITH_LINK  The first wired device with link
#
default_on_boot = DEFAULT_ROUTE_DEVICE

[Payload]
# Default package environment.
default_environment =

# List of ignored packages.
ignored_packages =
    ntfsprogs
    btrfs-progs
    dmraid

# Enable installation of latest updates.
enable_updates = False

# List of .treeinfo variant types to enable.
# Valid items:
#
# addon
# optional
# variant
#
enabled_repositories_from_treeinfo = addon optional variant

# Enable installation from the closest mirror.
enable_closest_mirror = False

# Default installation source.
# Valid values:
#
# CLOSEST_MIRROR  Use closest public repository mirror.
```

```
# CDN          Use Content Delivery Network (CDN).
#
default_source = CDN

# Enable ssl verification for all HTTP connection
verify_ssl = True

[Security]
# Enable SELinux usage in the installed system.
# Valid values:
#
# -1 The value is not set.
# 0 SELinux is disabled.
# 1 SELinux is enabled.
#
selinux = -1

[Bootloader]
# Type of the bootloader.
# Supported values:
#
# DEFAULT Choose the type by platform.
# EXTLINUX Use extlinux as the bootloader.
#
type = DEFAULT

# Name of the EFI directory.
efi_dir = redhat

# Hide the GRUB menu.
menu_auto_hide = False

# Are non-iBFT iSCSI disks allowed?
nonibft_iscsi_boot = False

# Arguments preserved from the installation system.
preserved_arguments =
  cio_ignore rd.znet rd_ZNET zfcplib.allow_lun_scan
  speakup_synth apic noapic apm ide noht acpi video
  pci nodmraid nompath nomodeset noiswmd fips selinux
  biosdevname ipv6.disable net.ifnames net.ifnames.prefix
  nosmt

[Storage]
# Enable dmraid usage during the installation.
dmraid = True

# Enable iBFT usage during the installation.
ibft = True

# Do you prefer creation of GPT disk labels?
gpt = False
```

```
# Tell multipathd to use user friendly names when naming devices during the installation.
multipath_friendly_names = True

# Do you want to allow imperfect devices (for example, degraded mdraid array devices)?
allow_imperfect_devices = False

# Default file system type. Use whatever Blivet uses by default.
file_system_type = xfs

# Default partitioning.
# Valid values:
#
# SERVER      Choose partitioning for servers.
# WORKSTATION Choose partitioning for workstations.
# VIRTUALIZATION Choose partitioning for virtualizations.
#
default_partitioning = WORKSTATION

# Default partitioning scheme.
# Valid values:
#
# PLAIN      Create standard partitions.
# BTRFS      Use the Btrfs scheme.
# LVM        Use the LVM scheme.
# LVM_THINP  Use LVM Thin Provisioning.
#
default_scheme = LVM

# Default version of LUKS.
# Valid values:
#
# luks1      Use version 1 by default.
# luks2      Use version 2 by default.
#
luks_version = luks2

[Storage Constraints]

# Minimal size of the total memory.
min_ram = 320 MiB

# Minimal size of the available memory for LUKS2.
luks2_min_ram = 128 MiB

# Should we recommend specifying a swap partition?
swap_is_recommended = True

# Recommended minimal sizes of partitions.
# Specify a mount point and a size on each line.
min_partition_sizes =
 /    250 MiB
 /usr 250 MiB
 /tmp  50 MiB
 /var 384 MiB
 /home 100 MiB
```

```
/boot 200 MiB

# Required minimum sizes of partitions.
# Specify a mount point and a size on each line.
req_partition_sizes =

# Allowed device types of the / partition if any.
# Valid values:
#
# LVM    Allow LVM.
# MD     Allow RAID.
# PARTITION Allow standard partitions.
# BTRFS  Allow Btrfs.
# DISK   Allow disks.
# LVM_THINP Allow LVM Thin Provisioning.
#
root_device_types =

# Mount points that must be on a linux file system.
# Specify a list of mount points.
must_be_on_linuxfs = / /var /tmp /usr /home /usr/share /usr/lib

# Paths that must be directories on the / file system.
# Specify a list of paths.
must_be_on_root = /bin /dev /sbin /etc /lib /root /mnt lost+found /proc

# Paths that must NOT be directories on the / file system.
# Specify a list of paths.
must_not_be_on_root =

[User Interface]
# The path to a custom stylesheet.
custom_stylesheet =

# The path to a directory with help files.
help_directory = /usr/share/anaconda/help/rhel

# Default help pages for TUI, GUI and Live OS.
default_help_pages =
    rhel_help_placeholder.txt
    rhel_help_placeholder.xml
    rhel_help_placeholder.xml

# A list of spokes to hide in UI.
# FIXME: Use other identification then names of the spokes.
hidden_spokes =

[License]
# A path to EULA (if any)
#
# If the given distribution has an EULA & feels the need to
# tell the user about it fill in this variable by a path
# pointing to a file with the EULA on the installed system.
#
```

```
# This is currently used just to show the path to the file to
# the user at the end of the installation.
eula = /usr/share/redhat-release/EULA
```

3.3.2. Configuring the product configuration files

The product configuration files have one or two extra sections that identify the product. The **[Product]** section specifies the product name of a product. The **[Base Product]** section specifies the product name of a base product if any. For example, Red Hat Enterprise Linux is a base product of Red Hat Virtualization.

The installer loads configuration files of the base products before it loads the configuration file of the specified product. For example, it will first load the configuration for Red Hat Enterprise Linux and then the configuration for Red Hat Virtualization.

See an example of the product configuration file for Red Hat Virtualization:

```
[Product]
product_name = Red Hat Virtualization

[Base Product]
product_name = Red Hat Enterprise Linux

[Storage]
default_scheme = LVM_THINP
default_partitioning = VIRTUALIZATION

[Storage Constraints]
root_device_types = LVM_THINP
must_not_be_on_root = /var
req_partition_sizes =
/var 10 GiB
/boot 1 GiB

[User Interface]
help_directory = /usr/share/anaconda/help/rhv
```

To customize the installer configuration for your product, you must create a product configuration file. Create a new file named **my-distribution.conf**, with content similar to the example above. Change *product_name* in the **[Product]** section to the name of your product, for example My Distribution. The product name should be the same as the name used in the **.buildstamp** file.

After you create the custom configuration file, follow the steps in [Section 5.1, “Creating a product img file”](#) section to create a **new product.img** file containing your customizations, and the [Section 5.2, “Creating custom boot images”](#) to create a new bootable ISO file with your changes included.

3.3.3. Configuring the custom configuration files

To customize the installer configuration independently of the product name, you must create a custom configuration file. To do so, create a new file named **100-my-configuration.conf** with the content similar to the example in [Section 3.3.2, “Configuring the product configuration files”](#) and omit the **[Product]** and **[Base Product]** sections.

After you create the custom configuration file, follow the steps in [Section 5.1, “Creating a product image file”](#) section to create a **new product.img** file containing your customizations, and the [Section 5.2, “Creating custom boot images”](#) to create a new bootable ISO file with your changes included.

CHAPTER 4. DEVELOPING INSTALLER ADD-ONS

This section provides details about Anaconda and its architecture, and how to develop your own add-ons. The details about Anaconda and its architecture helps you to understand Anaconda backend and various plug points for the add-ons to work. It also helps to accordingly develop the add-ons.

4.1. INTRODUCTION TO ANACONDA AND ADD-ONS

Anaconda is the operating system installer used in Fedora, Red Hat Enterprise Linux, and their derivatives. It is a set of Python modules and scripts together with some additional files like **Gtk** widgets (written in C), **systemd** units, and **dracut** libraries. Together, they form a tool that allows users to set parameters of the resulting (target) system and then set up this system on a machine. The installation process has four major steps:

1. Prepare installation destination (usually disk partitioning)
2. Install package and data
3. Install and configure boot loader
4. Configure newly installed system

Using Anaconda enables you to install Fedora, Red Hat Enterprise Linux, and their derivatives, in the following three ways:

Using graphical user interface (GUI):

This is the most common installation method. The interface allows users to install the system interactively with little or no configuration required before starting the installation. This method covers all common use cases, including setting up complicated partitioning layouts.

The graphical interface supports remote access over **VNC**, which allows you to use the GUI even on systems with no graphics cards or attached monitor.

Using text user interface (TUI):

The TUI works similar to a monochrome line printer, which allows it to work on serial consoles that do not support cursor movement, colors and other advanced features. The text mode is limited and allows you to customize only the most common options, such as network settings, language options or installation (package) source; advanced features such as manual partitioning are not available in this interface.

Using Kickstart file:

A Kickstart file is a plain text file with shell-like syntax that can contain data to drive the installation process. A Kickstart file allows you to partially or completely automate the installation. A set of commands which configures all required areas is necessary to completely automate the installation. If one or more commands are missed, the installation requires interaction.

Apart from automation of the installer itself, Kickstart files can contain custom scripts that are run at specific moments during the installation process.

4.2. ANACONDA ARCHITECTURE

Anaconda is a set of Python modules and scripts. It also uses several external packages and libraries. The major components of this toolset include the following packages:

- **pykickstart** - parses and validates the Kickstart files. Also, provides data structure that stores values that drives the installation.
- **yum** - the package manager that installs packages and resolves dependencies
- **blivet** - handles all activities related to storage management
- **pyanaconda** - contains the user interface and modules for **Anaconda**, such as keyboard and timezone selection, network configuration, and user creation. Also provides various utilities to perform system-oriented functions
- **python-meh** - contains an exception handler that gathers and stores additional system information in case of a crash and passes this information to the **libreport** library, which itself is a part of the [ABRT Project](#)
- **dbus** - enables communication between the **D-Bus** library with modules of anaconda and with external components
- **python-simpleline** - text UI framework library to manage user interaction in the **Anaconda** text mode
- **gtk** - the Gnome toolkit library for creating and managing GUI

Apart from the division into packages previously mentioned, **Anaconda** is internally divided into the user interface and a set of modules that run as separate processes and communicate using the **D-Bus** library. These modules are:

- **Boss** - manages the internal module discovery, lifecycle, and coordination
- **Localization** - manages locales
- **Network** - handles network
- **Payloads** - handles data for installation in different formats, such as **rpm**, **ostree**, **tar** and other installation formats. Payloads manage the sources of data for installation; sources can vary in format such as CD-ROM, HDD, NFS, URLs, and other sources
- **Security** - manages security related aspects
- **Services** - handles services
- **Storage** - manages storage using **blivet**
- **Subscription** - handles the **subscription-manager** tool and Insights.
- **Timezone** - deals with time, date, zones, and time synchronization.
- **Users** - creates users and groups.

Each module declares which parts of Kickstart it handles, and has methods to apply the configuration from Kickstart to the installation environment and to the installed system.

The Python code portion of Anaconda (**pyanaconda**) starts as a “main” process that owns the user interface. Any Kickstart data you provide are parsed using the **pykickstart** module and the **Boss** module is started, it discovers all other modules, and starts them. Main process then sends Kickstart data to the modules according to their declared capabilities. Modules process the data, apply the configuration to

the installation environment, and the UI validates if all required choices have been made. If not, you must supply the data in an interactive installation mode. Once all required choices have been made, the installation can start - the modules write data to the installed system.

4.3. ANACONDA USER INTERFACE

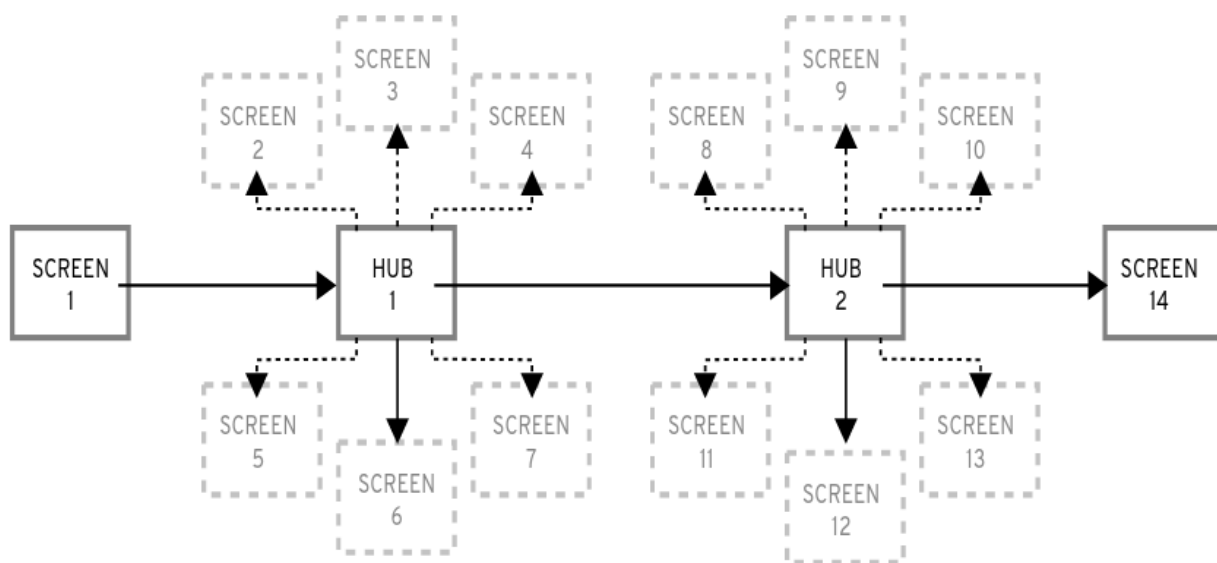
The Anaconda user interface (UI) has a non-linear structure, also known as hub and spoke model.

The advantages of **Anaconda** hub and spoke model are:

- Flexibility to follow the installer screens.
- Flexibility to retain the default settings.
- Provides an overview of the configured values.
- Supports extensibility. You can add hubs without the need to reorder anything and can resolve some complex ordering dependencies.
- Supports installation in graphical and text mode.

The following diagram shows the installer layout and the possible interactions between *hubs* and *spokes* (screens):

Figure 4.1. Hub and spoke model



In the diagram, screens 2-13 are called *normal spokes*, and screens 1 and 14 are *standalone spokes*. Standalone spokes are the screens that can be used before or after the standalone spoke or hub. For example, the **Welcome** screen at the beginning of the installation which prompts you to choose your language for the rest of the installation.



NOTE

- The **Installation Summary** is the only hub in Anaconda. It shows a summary of configured options before the installation begins

Each spoke has the following predefined *properties* that reflect the hub.

- **ready** - states whether or not you can visit a spoke. For example, when the installer is configuring a package source, the spoke is colored in gray, and you cannot access it until the configuration is complete.
- **completed** - marks whether or not is the spoke complete (all required values are set).
- **mandatory** - determines whether you *must* visit the spoke before continuing the installation; for example, you must visit the **Installation Destination** spoke, even if you want to use automatic disk partitioning
- **status** - provides a short summary of values configured within the spoke (displayed under the spoke name in the hub)

To make the user interface clearer, spokes are grouped together into *categories*. For example, the **Localization** category groups together spokes for keyboard layout selection, language support and time zone settings.

Each spoke contains UI controls which display and allow you to modify values from one or more modules. The same applies to spokes that add-ons provide.

4.4. COMMUNICATION ACROSS ANACONDA THREADS

Some of the actions that you need to perform during the installation process may take a long time. For example, scanning disks for existing partitions or downloading package metadata. To prevent you from waiting and remaining responsive, **Anaconda** runs these actions in separate threads.

The **Gtk** toolkit does not support element changes from multiple threads. The main event loop of **Gtk** runs in the main thread of the **Anaconda** process. Therefore, all actions pertaining to the GUI must be performed in the main thread. To do so, use **GLib.idle_add**, which is not always easy or desired. Several helper functions and decorators that are defined in the **pyanaconda.ui.gui.utils** module may add to the difficulty.

The **@gtk_action_wait** and **@gtk_action_nowait** decorators change the decorated function or method in such a way that when this function or method is called, it is automatically queued into **Gtk**'s main loop that runs in the main thread. The return value is either returned to the caller or dropped, respectively.

In a spoke and hub communication, a spoke announces when it is ready and is not blocked. The **hubQ** message queue handles this function, and periodically checks the main event loop. When a spoke becomes accessible, it sends a message to the queue announcing the change and that it should no longer be blocked.

The same applies in a situation where a spoke needs to refresh its status or complete a flag. The **Configuration and Progress** hub has a different queue called **progressQ** which serves as a medium to transfer installation progress updates.

These mechanisms are also used for the text-based interface. In the text mode, there is no main loop, but the keyboard input takes most of the time.

4.5. ANACONDA MODULES AND D-BUS LIBRARY

Anaconda's modules run as independent processes. To communicate with these processes via their **D-Bus** API, use the **dbus** library.

Calls to methods via **D-Bus` API are asynchronous, but with the `dbus** library you can convert them to synchronous method calls in Python. You can also write either of the following programs:

- program with asynchronous calls and return handlers
- A program with synchronous calls that makes the caller wait until the call is complete.

For more information about threads and communication, see [Section 4.4, “Communication across Anaconda threads”](#).

Additionally, Anaconda uses Task objects running in modules. Tasks have a **D-Bus** API and methods that are automatically executed in additional threads. To successfully run the tasks, use the **sync_run_task** and **async_run_task** helper functions.

4.6. THE HELLO WORLD ADDON EXAMPLE

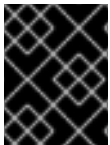
Anaconda developers publish an example addon called “Hello World”, available on GitHub: <https://github.com/rhinstaller/hello-world-anaconda-addon/> The descriptions in further sections are reproduced in this.

4.7. ANACONDA ADD-ON STRUCTURE

An **Anaconda** add-on is a Python package that contains a directory with an **__init__.py** and other source directories (subpackages). Because Python allows you to import each package name only once, specify a unique name for the package top-level directory. You can use an arbitrary name, because add-ons are loaded regardless of their name - the only requirement is that they must be placed in a specific directory.

The suggested naming convention for add-ons is similar to Java packages or D-Bus service names.

To make the directory name a unique identifier for a Python package, prefix the add-on name with the reversed domain name of your organization, using underscores (**_**) instead of dots. For example, **com_example_hello_world**.



IMPORTANT

Make sure to create an **__init__.py** file in each directory. Directories missing this file are considered as invalid Python packages.

When writing an add-on, ensure the following:

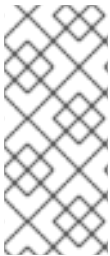
- Support for each interface (graphical interface and text interface) is available in a separate subpackage and these subpackages are named **gui** for the graphical interface and **tui** for the text-based interface.
- The **gui** and **tui** packages contain a **spokes** subpackage. ^[1]
- Modules contained in the packages have an arbitrary name.
- The **gui/** and **tui/** directories contain Python modules with any name.
- There is a service that performs the actual work of the addon. This service can be written in Python or any other language.
- The service implements support for D-Bus and Kickstart.
- The addon contains files that enable automatic startup of the service.

Following is a sample directory structure for an add-on which supports every interface (Kickstart, GUI and TUI):

Example 4.1. Sample Add-on Structure

```
com_example_hello_world
├── gui
│   ├── init.py
│   └── spokes
│       └── init.py
└── tui
    ├── init.py
    ├── spokes
    └── init.py
```

Each package must contain at least one module with an arbitrary name defining the classes that are inherited from one or more classes defined in the API.



NOTE

For all add-ons, follow Python's [PEP 8](#) and [PEP 257](#) guidelines for docstring conventions. There is no consensus on the format of the actual content of docstrings in **Anaconda**; the only requirement is that they are human-readable. If you plan to use auto-generated documentation for your add-on, docstrings should follow the guidelines for the toolkit you use to accomplish this.

You can include a category subpackage if an add-on needs to define a new category, but this is not recommended.

4.8. ANACONDA SERVICES AND CONFIGURATION FILES

Anaconda services and configuration files are included in `data/` directory. These files are required to start the add-ons service and to configure D-Bus.

Following are some examples of Anaconda Hello World add-on:

Example 4.2. Example of `addon-name.conf`:

```
<!DOCTYPE busconfig PUBLIC
"-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>
  <policy user="root">
    <allow own="org.fedoraproject.Anaconda.Addons.HelloWorld"/>
    <allow send_destination="org.fedoraproject.Anaconda.Addons.HelloWorld"/>
  </policy>
  <policy context="default">
    <deny own="org.fedoraproject.Anaconda.Addons.HelloWorld"/>
    <allow send_destination="org.fedoraproject.Anaconda.Addons.HelloWorld"/>
  </policy>
</busconfig>
```

This file must be placed in the `/usr/share/anaconda/dbus/confs/` directory in the installation environment. The string `org.fedoraproject.Anaconda.Addons.HelloWorld` must correspond to the location of add-on's service on D-Bus.

Example 4.3. Example of `addon-name.service`:

```
[D-BUS Service]
# Start the org.fedoraproject.Anaconda.Addons.HelloWorld service.
# Runs org_fedora_hello_world/service/main.py
Name=org.fedoraproject.Anaconda.Addons.HelloWorld
Exec=/usr/libexec/anaconda/start-module org_fedora_hello_world.service
User=root
```

This file must be placed in the `/usr/share/anaconda/dbus/services/` directory in the installation environment. The string `org.fedoraproject.Anaconda.Addons.HelloWorld` must correspond to the location of add-on's service on D-Bus. The value on the line starting with `Exec=` must be a valid command that starts the service in the installation environment.

4.9. DEPLOYING AND TESTING AN ANACONDA ADD-ON

You can deploy and test your own Anaconda add-on into the installation environment. To do so, follow the steps:

Prerequisites

- You created an Add-on.
- You have access to your **D-Bus** files.

Procedure

1. Create a directory **DIR** at the place of your preference.
2. Add the **Add-on** python files into `DIR/usr/share/anaconda/addons/`.
3. Copy your **D-Bus** service file into `DIR/usr/share/anaconda/dbus/services/`.
4. Copy your **D-Bus** service configuration file to `/usr/share/anaconda/dbus/confs/`.
5. Create the `updates` image.
Access the **DIR** directory:

```
cd DIR
```

Locate the `updates` image.

```
find . | cpio -c -o | pigz -9cv > DIR/updates.img
```

6. Extract the contents of the ISO boot image.
7. Use the resulting **updates** image:
 - a. Add the **updates.img** file into the **images** directory of your unpacked ISO contents.

- b. Repack the image.
- c. Set up a web server to provide the **updates**.img file to the Anaconda installer via HTTP.
- d. Load **updates**.img file at boot time by adding the following specification to the boot options.

```
inst.updates=http://your-server/whatever/updates.img to boot options.
```

For specific instructions on unpacking an existing boot image, creating a **product.img** file and repackaging the image, see [Section 1.2.3, "Extracting Red Hat Enterprise Linux boot images"](#) .

[1] The **gui** package may also contain a **categories** subpackage if the add-on needs to define a new category, but this is not recommended.

CHAPTER 5. COMPLETING POST CUSTOMIZATION TASKS

To complete the customizations made, perform the following tasks:

- Create a `product.img` image file (applies only for graphical customizations).
- Create a custom boot image.

This section provides information about how to create a `product.img` image file and to create a custom boot image.

5.1. CREATING A PRODUCT IMG FILE

A **product.img** image file is an archive containing new installer files that replace the existing ones at runtime.

During a system boot, **Anaconda** loads the `product.img` file from the `images/` directory on the boot media. It then uses the files that are present in this directory to replace identically named files in the installer's file system. The files when replaced customizes the installer (for example, for replacing default images with custom ones).

Note: The **product.img** image must contain a directory structure identical to the installer. For more information about the installer directory structure, see table, [Table 5.1, "Installer directory structure and custom contents"](#)

Table 5.1. Installer directory structure and custom contents

Type of custom content	File system location
Pixmaps (logo, sidebar, top bar, and so on.)	<code>/usr/share/anaconda/pixmaps/</code>
GUI stylesheet	<code>/usr/share/anaconda/anaconda-gtk.css</code>
Anaconda add-ons	<code>/usr/share/anaconda/addons/</code>
Product configuration files	<code>/etc/anaconda/product.d/</code>
Custom configuration files	<code>/etc/anaconda/conf.d/</code>
Anaconda Dbus service conf files	<code>/usr/share/anaconda/dbus/confs/</code>
Anaconda Dbus service files	<code>/usr/share/anaconda/dbus/services/</code>

The procedure below explains how to create a **product.img** file.

Procedure

1. Navigate to a working directory such as `/tmp`, and create a subdirectory named **product/**:

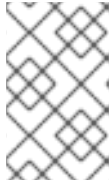
```
$ cd /tmp
```

2. Create a subdirectory `product/`

```
$ mkdir product/
```

3. Create a directory structure identical to the location of the file you want to replace. For example, if you want to test an add-on that is present in the `/usr/share/anaconda/addons` directory on the installation system, create the same structure in your working directory:

```
$ mkdir -p product/usr/share/anaconda/addons
```



NOTE

To view the installer's runtime file, boot the installation and switch to virtual console 1 (**Ctrl+Alt+F1**) and then switch to the second **tmux** window (**Ctrl+b 2**). A shell prompt that can be used to browse a file system opens.

4. Place your customized files (in this example, custom add-on for **Anaconda**) into the newly created directory:

```
$ cp -r ~/path/to/custom/addon/ product/usr/share/anaconda/addons/
```

5. Repeat steps 3 and 4 (create a directory structure and place the custom files into it) for every file you want to add to the installer.
6. Create a **.buildstamp** file in the root of the directory. The **.buildstamp** file describes the system version, the product and several other parameters. The following is an example of a **.buildstamp** file from Red Hat Enterprise Linux 8.3:

```
[Main]
Product=Red Hat Enterprise Linux
Version=8.3
BugURL=https://bugzilla.redhat.com/
IsFinal=True
UUID=202007011344.x86_64
[Compose]
Lorax=28.14.49-1
```

The **IsFinal** parameter, specifies whether the image is for a release (GA) version of the product (**True**), or a pre-release such as Alpha, Beta, or an internal milestone (**False**).

7. Navigate to the `product/` directory, and create the `product.img` archive:

```
$ cd product
```

```
$ find . | cpio -c -o | gzip -9cv > ../product.img
```

This creates a `product.img` file one level above the `product/` directory.

8. Move the `product.img` file to the `images/` directory of the extracted ISO image.

The `product.img` file is now created and the customizations that you want to make are placed in the respective directories.



NOTE

Instead of adding the **product.img** file on the boot media, you can place this file into a different location and use the **inst.updates=** boot option at the boot menu to load it. In that case, the image file can have any name, and it can be placed in any location (USB flash drive, hard disk, HTTP, FTP or NFS server), as long as this location is reachable from the installation system.

See the [Anaconda Boot Options](#) for more information about Anaconda boot options.

5.2. CREATING CUSTOM BOOT IMAGES

After you customize the boot images and the GUI layout, create a new image that includes the changes you made.

To create custom boot images, follow the procedure below.

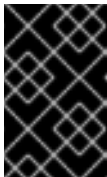
Procedure

1. Make sure that all of your changes are included in the working directory. For example, if you are testing an add-on, make sure to place the **product.img** in the **images/** directory.
2. Make sure your current working directory is the top-level directory of the extracted ISO image - e.g. **/tmp/ISO/iso**.
3. Create a new ISO image using the **genisoimage**:

```
# genisoimage -U -r -v -T -J -joliet-long -V "RHEL-8.2 Server.x86_64" -volset "RHEL-8.2
Server.x86_64" -A "RHEL-8.2 Server.x86_64" -b isolinux/isolinux.bin -c isolinux/boot.cat -no-
emul-boot -boot-load-size 4 -boot-info-table -eltorito-alt-boot -e images/efiboot.img -no-emul-
boot -o ../NEWISO.iso .
```

In the above example:

- Make sure that the values for **-V**, **-volset**, and **-A** options match the image's boot loader configuration, if you are using the **LABEL=** directive for options that require a location to load a file on the same disk. If your boot loader configuration (**isolinux/isolinux.cfg** for BIOS and **EFI/BOOT/grub.cfg** for UEFI) uses the **inst.stage2=LABEL=disk_label** stanza to load the second stage of the installer from the same disk, then the disk labels must match.



IMPORTANT

In boot loader configuration files, replace all spaces in disk labels with **\x20**. For example, if you create an ISO image with a **RHEL 8.2** label, boot loader configuration should use **RHEL\x207.1**.

- Replace the value of the **-o** option (**-o ../NEWISO.iso**) with the file name of your new image. The value in the example creates the **NEWISO.iso** file in the directory *above* the current one. For more information about this command, see the **genisoimage(1)** man page.

4. Implant an MD5 checksum into the image. Note that without an MD5 checksum, the image verification check might fail (the **rd.live.check** option in the boot loader configuration) and the installation can hang.

```
# implantisomd5 ../NEWISO.iso
```

In the above example, replace `../NEWISO.iso` with the file name and the location of the ISO image that you have created in the previous step.

You can now write the new ISO image to physical media or a network server to boot it on physical hardware, or you can use it to start installing a virtual machine.

Additional resources

- For instructions on preparing boot media or network server, see [Performing an advanced RHEL Installation](#).
- For instructions on creating virtual machines with ISO images, see [Configuring and Managing Virtualization](#).