Red Hat Enterprise Linux 9

Personnaliser Anaconda

Modifier l'apparence du programme d'installation et créer des modules complémentaires personnalisés sur Red Hat Enterprise Linux

Last Updated: 2024-02-19

Red Hat Enterprise Linux 9 Personnaliser Anaconda

Modifier l'apparence du programme d'installation et créer des modules complémentaires personnalisés sur Red Hat Enterprise Linux

Notice légale

Résumé

Anaconda est le programme d'installation utilisé par Red Hat Enterprise Linux. Vous pouvez personnaliser Anaconda pour étendre ses capacités lorsque vous installez RHEL dans votre environnement.

Table des matières

RENDRE L'OPEN SOURCE PLUS INCLUSIF

Red Hat s'engage à remplacer les termes problématiques dans son code, sa documentation et ses propriétés Web. Nous commençons par ces quatre termes : master, slave, blacklist et whitelist. En raison de l'ampleur de cette entreprise, ces changements seront mis en œuvre progressivement au cours de plusieurs versions à venir. Pour plus de détails, voir le message de notre directeur technique Chris Wright.

FOURNIR UN RETOUR D'INFORMATION SUR LA DOCUMENTATION DE RED HAT

Nous apprécions vos commentaires sur notre documentation. Faites-nous savoir comment nous pouvons l'améliorer.

Soumettre des commentaires sur des passages spécifiques

1. Consultez la documentation au format **Multi-page HTML** et assurez-vous que le bouton **Feedback** apparaît dans le coin supérieur droit après le chargement complet de la page.
2. Utilisez votre curseur pour mettre en évidence la partie du texte que vous souhaitez commenter.
3. Cliquez sur le bouton **Add Feedback** qui apparaît près du texte en surbrillance.
4. Ajoutez vos commentaires et cliquez sur **Submit**.

Soumettre des commentaires via Bugzilla (compte requis)

1. Connectez-vous au site Web de Bugzilla.
2. Sélectionnez la version correcte dans le menu **Version**.
3. Saisissez un titre descriptif dans le champ **Summary**.
4. Saisissez votre suggestion d'amélioration dans le champ **Description**. Incluez des liens vers les parties pertinentes de la documentation.
5. Cliquez sur **Submit Bug**.

CHAPITRE 1. INTRODUCTION À LA PERSONNALISATION D'ANACONDA

1.1. INTRODUCTION À LA PERSONNALISATION D'ANACONDA

Le programme d'installation de Red Hat Enterprise Linux et Fedora, **Anaconda**apporte de nombreuses améliorations dans ses versions les plus récentes. L'une de ces améliorations est une meilleure personnalisation. Vous pouvez désormais écrire des modules complémentaires pour étendre les fonctionnalités de l'installateur de base et modifier l'apparence de l'interface utilisateur graphique.

Ce document explique comment personnaliser les éléments suivants :

- Menu de démarrage - options préconfigurées, schéma de couleurs et arrière-plan
- Apparence de l'interface graphique - logo, arrière-plans, nom du produit
- Fonctionnalité de l'installateur - modules complémentaires qui peuvent améliorer l'installateur en ajoutant de nouvelles commandes Kickstart et de nouveaux écrans dans les interfaces utilisateur graphiques et textuelles

Notez également que ce document ne s'applique qu'à Red Hat Enterprise Linux 8 et Fedora 17 et versions ultérieures.

IMPORTANT

Les procédures décrites dans ce livre sont écrites pour Red Hat Enterprise Linux 9 ou un système similaire. Sur d'autres systèmes, les outils et les applications utilisés (tels que **genisoimage** pour la création d'images ISO personnalisées) peuvent être différents et les procédures peuvent devoir être ajustées.

CHAPITRE 2. EXÉCUTION DES TÂCHES DE PRÉ-PERSONNALISATION

2.1. TRAVAILLER AVEC DES IMAGES ISO

Dans cette section, vous apprendrez à :

- Extraire une ISO Red Hat.
- Créez une nouvelle image de démarrage contenant vos personnalisations.

2.2. TÉLÉCHARGEMENT DES IMAGES DE DÉMARRAGE DE RH

Avant de commencer à personnaliser le programme d'installation, téléchargez les images de démarrage fournies par Red Hat. Vous pouvez obtenir les supports de démarrage de Red Hat Enterprise Linux 9 à partir du portail client de Red Hat après vous être connecté à votre compte.

NOTE

- Votre compte doit avoir suffisamment de droits pour télécharger les images de Red Hat Enterprise Linux 9.
- Vous devez télécharger l'image **Binary DVD** ou **Boot ISO** et pouvez utiliser n'importe quelle variante de l'image (serveur ou ComputeNode).
- Vous ne pouvez pas personnaliser le programme d'installation en utilisant les autres téléchargements disponibles, tels que l'image KVM Guest ou le DVD supplémentaire ; les autres téléchargements disponibles, tels que **KVM Guest Image** ou **Supplementary DVD**.

Pour plus d'informations sur les téléchargements du DVD binaire et de l'ISO de démarrage, voir Red Hat Enterprise Linux 9 Effectuer une installation avancée de RHEL 9.

2.3. EXTRACTION DES IMAGES DE DÉMARRAGE DE RED HAT ENTERPRISE LINUX

Effectuez la procédure suivante pour extraire le contenu d'une image de démarrage.

Procédure

1. Assurez-vous que le répertoire ***/mnt/iso*** existe et que rien n'y est actuellement monté.
2. Monter l'image téléchargée.

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

   Où *path/to/image.iso* est le chemin d'accès à l'image de démarrage téléchargée.
3. Créez un répertoire de travail dans lequel vous souhaitez placer le contenu de l'image ISO.

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

4. Copiez tout le contenu de l'image montée dans votre nouveau répertoire de travail. Veillez à utiliser l'option **-p** pour préserver les autorisations et la propriété des fichiers et des répertoires.

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

5. Démonter l'image.

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

Ressources supplémentaires

- Pour des instructions de téléchargement détaillées et une description des téléchargements du DVD binaire et de l'ISO de démarrage, consultez le site Web de Red Hat Enterprise Linux 9 .

CHAPITRE 3. PERSONNALISER LE MENU DE DÉMARRAGE

Cette section fournit des informations sur la personnalisation du menu Boot et sur la manière de le personnaliser.

Prérequis :

Pour plus d'informations sur le téléchargement et l'extraction des images de démarrage, voir Extraction des images de démarrage de Red Hat Enterprise Linux

La personnalisation du menu de démarrage implique les tâches de haut niveau suivantes :

1. Remplir les conditions préalables.
2. Personnaliser le menu de démarrage.
3. Créer une image de démarrage personnalisée.

3.1. PERSONNALISER LE MENU DE DÉMARRAGE

Le site *Boot menu* est le menu qui apparaît lorsque vous démarrez votre système à l'aide d'une image d'installation. Normalement, ce menu vous permet de choisir entre des options telles que **Install Red Hat Enterprise Linux**, **Boot from local drive** ou **Rescue an installed system**. Pour personnaliser le menu de démarrage, vous pouvez :

- Personnaliser les options par défaut.
- Ajouter des options.
- Modifier le style visuel (couleur et arrière-plan).

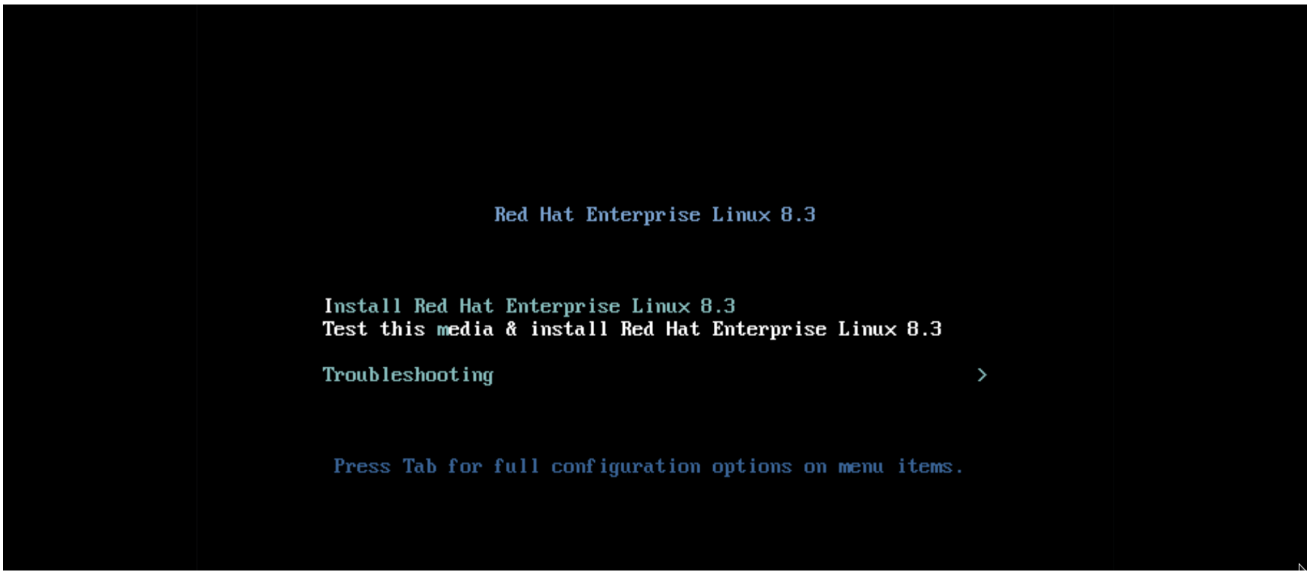
Un média d'installation se compose des chargeurs de démarrage **ISOLINUX** et **GRUB2**. Le chargeur de démarrage **ISOLINUX** est utilisé sur les systèmes dotés d'un microprogramme BIOS, et le chargeur de démarrage **GRUB2** est utilisé sur les systèmes dotés d'un microprogramme UEFI. Les deux chargeurs de démarrage sont présents sur toutes les images Red Hat pour les systèmes AMD64 et Intel 64.

La personnalisation des options du menu de démarrage peut être particulièrement utile avec Kickstart. Les fichiers Kickstart doivent être fournis à l'installateur avant le début de l'installation. Normalement, cela se fait en modifiant manuellement l'une des options de démarrage existantes pour ajouter l'option de démarrage **inst.ks=**. Vous pouvez ajouter cette option à l'une des entrées préconfigurées si vous modifiez les fichiers de configuration du chargeur de démarrage sur le support.

3.2. SYSTÈMES AVEC MICROLOGICIEL BIOS

Le chargeur d'amorçage **ISOLINUX** le chargeur de démarrage est utilisé sur les systèmes dotés d'un microprogramme BIOS.

Figure 3.1. Menu de démarrage d'ISOLINUX



Le fichier de configuration ***isolinux/isolinux.cfg*** sur le support d'amorçage contient des directives pour définir la palette de couleurs et la structure du menu (entrées et sous-menus).

Dans le fichier de configuration, l'entrée de menu par défaut de Red Hat Enterprise Linux, **Test this media & Install Red Hat Enterprise Linux 9**, est définie dans le bloc suivant :

```
label check
  menu label Test this ^media & install Red Hat Enterprise Linux 9.
  menu default
  kernel vmlinuz
  append initrd=initrd.img inst.stage2=hd:LABEL=RHEL-9-BaseOS-x86_64 rd.live.check
quiet
```

Où ?

- **menu label** - détermine comment l'entrée sera nommée dans le menu. Le caractère **^** détermine son raccourci clavier (la touche **m**).
- **menu default** - fournit une sélection par défaut, même s'il ne s'agit pas de la première option de la liste.
- **kernel** - charge le noyau d'installation. Dans la plupart des cas, il ne doit pas être modifié.
- **append** - contient des options supplémentaires pour le noyau. Les options **initrd=** et **inst.stage2** sont obligatoires ; vous pouvez en ajouter d'autres.
  Pour plus d'informations sur les options applicables à l'installation de RHEL 9, reportez-vous au Guide d'installation de Red Hat Enterprise Linux 9 **Anaconda** reportez-vous au Guide d'installation standard de Red Hat Enterprise Linux 9 Performing a standard RHEL 9 installation Guide.

  L'une des options notables est **inst.ks=**, qui vous permet de spécifier l'emplacement d'un fichier Kickstart. Vous pouvez placer un fichier Kickstart sur l'image ISO de démarrage et utiliser l'option inst.ks= pour spécifier son emplacement ; par exemple, vous pouvez placer un fichier **kickstart.ks** dans le répertoire racine de l'image et utiliser **inst.ks=hd:LABEL=RHEL-9-BaseOS-x86_64:/kickstart.ks**.

  Vous pouvez également utiliser les options de **dracut** qui sont listées sur la page de manuel **dracut.cmdline(7)**.

IMPORTANT

Lorsque vous utilisez une étiquette de disque pour faire référence à un certain lecteur (comme dans l'option **inst.stage2=hd:LABEL=RHEL-9-BaseOS-x86_64** ci-dessus), remplacez tous les espaces par **\x20**.

Les autres options importantes qui ne sont pas incluses dans la définition de l'entrée de menu sont les suivantes :

- **timeout** - détermine la durée d'affichage du menu de démarrage avant l'utilisation automatique de l'entrée de menu par défaut. La valeur par défaut est **600**, ce qui signifie que le menu est affiché pendant 60 secondes. La définition de cette valeur à **0** désactive l'option de délai d'attente.

  NOTE

  Il est utile de fixer le délai d'attente à une valeur faible, telle que **1**, lorsque vous effectuez une installation sans tête. Cela permet d'éviter le délai d'attente par défaut.

- **menu begin** et **menu end** - déterminent le début et la fin d'un bloc *submenu*, ce qui vous permet d'ajouter des options supplémentaires telles que le dépannage et de les regrouper dans un sous-menu. Un sous-menu simple avec deux options (une pour continuer et une pour revenir au menu principal) ressemble à ce qui suit :

```
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-9-BaseOS-x86_64 rescue quiet

menu separator

label returntomain
  menu label Return to ^main menu
  menu exit

menu end
```

  Les définitions des entrées de sous-menu sont similaires aux entrées de menu normales, mais elles sont regroupées entre les instructions **menu begin** et **menu end**. La ligne **menu exit** de la deuxième option permet de quitter le sous-menu et de revenir au menu principal.

- **menu background** - l'arrière-plan du menu peut être soit une couleur unie (voir **menu color** ci-dessous), soit une image au format PNG, JPEG ou LSS16. Si vous utilisez une image, assurez-vous que ses dimensions correspondent à la résolution définie à l'aide de l'instruction **set resolution**. Les dimensions par défaut sont 640x480.

- **menu color** - détermine la couleur d'un élément de menu. Le format complet est le suivant :

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

  Les éléments les plus importants de cette commande sont les suivants :

- *element* - détermine l'élément auquel la couleur s'appliquera.
- *foreground* et *background* - déterminent les couleurs réelles.
  Les couleurs sont décrites à l'aide d'une ***#AARRGGBB*** en format hexadécimal détermine l'opacité :
- **00** pour une transparence totale.
- **ff** pour une opacité totale.
- **menu help *textfile*** - crée une entrée de menu qui, lorsqu'elle est sélectionnée, affiche un fichier texte d'aide.

Ressources supplémentaires

- Pour une liste complète des **ISOLINUX** options du fichier de configuration, voir le Wiki Syslinux.

3.3. SYSTÈMES AVEC MICROLOGICIEL UEFI

Le chargeur de démarrage **GRUB2** le chargeur de démarrage est utilisé sur les systèmes dotés d'un microprogramme UEFI.

Le fichier de configuration **EFI/BOOT/grub.cfg** sur le support de démarrage contient une liste d'entrées de menu préconfigurées et d'autres directives qui contrôlent l'apparence et la fonctionnalité du menu de démarrage.

Dans le fichier de configuration, l'entrée de menu par défaut pour Red Hat Enterprise Linux (**Test this media & install Red Hat Enterprise Linux 9**) est définie dans le bloc suivant :

```
menuentry 'Test this media & install Red Hat Enterprise Linux 9' --class fedora --class gnu-linux --class gnu --class os {
   linuxefi /images/pxeboot/vmlinuz inst.stage2=hd:LABEL=RHEL-9-BaseOS-x86_64 rd.live.check quiet
   initrdefi /images/pxeboot/initrd.img
}
```

Où ?

- **menuentry** - Définit le titre de l'entrée. Il est spécifié entre guillemets simples ou doubles (**'** ou **"**). Vous pouvez utiliser l'option **--class** pour regrouper les entrées de menu dans différents *classes*, qui peuvent ensuite être stylisés différemment à l'aide de thèmes **GRUB2** thèmes.

  NOTE

  Comme le montre l'exemple ci-dessus, chaque définition d'entrée de menu doit être entourée d'accolades (**{}**).

- **linuxefi** - Définit le noyau qui démarre (**/images/pxeboot/vmlinuz** dans l'exemple ci-dessus) et les autres options supplémentaires, le cas échéant.
  Vous pouvez personnaliser ces options pour modifier le comportement de l'entrée de démarrage. Pour plus de détails sur les options applicables à **Anaconda**pour plus d'informations sur les options applicables à RHEL 9, reportez-vous à Red Hat Enterprise Linux 9 Performing an advanced RHEL 9 installation.

L'une des options notables est **inst.ks=**, qui vous permet de spécifier l'emplacement d'un fichier Kickstart. Vous pouvez placer un fichier Kickstart sur l'image ISO de démarrage et utiliser l'option inst.ks= pour spécifier son emplacement ; par exemple, vous pouvez placer un fichier **kickstart.ks** dans le répertoire racine de l'image et utiliser **inst.ks=hd:LABEL=RHEL-9-BaseOS-x86_64:/kickstart.ks**.

Vous pouvez également utiliser les options de **dracut** qui sont listées sur la page de manuel **dracut.cmdline(7)**.

> IMPORTANT
>
> Lorsque vous utilisez une étiquette de disque pour faire référence à un certain lecteur (comme dans l'option **inst.stage2=hd:LABEL=RHEL-9-BaseOS-x86_64** ci-dessus), remplacez tous les espaces par **\x20**.

- **initrdefi** - emplacement de l'image du disque RAM initial (initrd) à charger.

Les autres options utilisées dans le fichier de configuration **grub.cfg** sont les suivantes :

- **set timeout** - détermine la durée d'affichage du menu de démarrage avant que l'entrée de menu par défaut ne soit automatiquement utilisée. La valeur par défaut est **60**, ce qui signifie que le menu est affiché pendant 60 secondes. En définissant cette valeur sur **-1**, vous désactivez complètement le délai d'attente.

> NOTE
>
> Le réglage du délai d'attente sur **0** est utile lors d'une installation sans tête, car il active immédiatement l'entrée de démarrage par défaut.

- **submenu** - Le bloc *submenu* permet de créer un sous-menu et d'y regrouper certaines entrées, au lieu de les afficher dans le menu principal. Dans la configuration par défaut, le sous-menu **Troubleshooting** contient des entrées permettant de sauver un système existant.
  Le titre de l'entrée est entre guillemets simples ou doubles (**'** ou **"**).

  Le bloc **submenu** contient une ou plusieurs définitions **menuentry** comme décrit ci-dessus, et le bloc entier est entouré d'accolades (**{}**). Par exemple :

```
submenu 'Submenu title' {
  menuentry 'Submenu option 1' {
    linuxefi /images/vmlinuz inst.stage2=hd:LABEL=RHEL-9-BaseOS-x86_64 xdriver=vesa
nomodeset quiet
    initrdefi /images/pxeboot/initrd.img
  }
  menuentry 'Submenu option 2' {
    linuxefi /images/vmlinuz inst.stage2=hd:LABEL=RHEL-9-BaseOS-x86_64 rescue quiet
    initrdefi /images/initrd.img
  }
}
```

- **set default** - Détermine l'entrée par défaut. Les numéros d'entrée commencent à **0**. Si vous voulez que l'entrée *third* soit l'entrée par défaut, utilisez **set default=2** et ainsi de suite.

- **theme** - détermine le répertoire qui contient les fichiers **GRUB2** les fichiers de thème. Vous pouvez utiliser les thèmes pour personnaliser les aspects visuels du chargeur de démarrage - arrière-plan, polices et couleurs d'éléments spécifiques.

Ressources supplémentaires

- Pour plus d'informations sur la personnalisation du menu de démarrage, voir le manuel GNU GRUB 2.00.
- Pour plus d'informations générales sur **GRUB2**, voir la page *Red Hat Enterprise Linux 9 Managing, monitoring and updating the kernel*.

CHAPITRE 4. MARQUAGE ET CHROMAGE DE L'INTERFACE UTILISATEUR GRAPHIQUE

La personnalisation de l'interface utilisateur d'Anaconda peut inclure la personnalisation des éléments graphiques et la personnalisation du nom du produit.

Cette section fournit des informations sur la manière de personnaliser les éléments graphiques et le nom du produit.

Conditions préalables

1. Vous avez téléchargé et extrait l'image ISO.
2. Vous avez créé votre propre matériel de marque.

Pour plus d'informations sur le téléchargement et l'extraction des images de démarrage, voir Extraction des images de démarrage de Red Hat Enterprise Linux

La personnalisation de l'interface utilisateur implique les tâches de haut niveau suivantes :

1. Remplir les conditions préalables.
2. Créer du matériel de marque personnalisé (si vous prévoyez de personnaliser les éléments graphiques)
3. Personnaliser les éléments graphiques (si vous envisagez de le faire)
4. Personnaliser le nom du produit (si vous envisagez de le personnaliser)
5. Créer un fichier product.img
6. Créer une image de démarrage personnalisée

> **NOTE**
>
> Pour créer un matériau de marquage personnalisé, reportez-vous d'abord au type et aux dimensions des fichiers d'éléments graphiques par défaut. Vous pouvez ensuite créer le matériau personnalisé. Les détails concernant les éléments graphiques par défaut sont disponibles dans les fichiers d'exemple fournis dans la section Personnaliser les éléments graphiques.

4.1. PERSONNALISATION DES ÉLÉMENTS GRAPHIQUES

Pour personnaliser les éléments graphiques, vous pouvez modifier ou remplacer les éléments personnalisables par le matériel de marque personnalisé et mettre à jour les fichiers de conteneur.

Les éléments graphiques personnalisables du programme d'installation sont stockés dans le répertoire **/usr/share/anaconda/pixmaps/** du système de fichiers d'exécution du programme d'installation. Ce répertoire contient les fichiers personnalisables suivants :

```
pixmaps
├─ anaconda-password-show-off.svg
├─ anaconda-password-show-on.svg
├─ right-arrow-icon.png
```

```
├── sidebar-bg.png
├── sidebar-logo.png
└── topbar-bg.png
```

En outre, le répertoire **/usr/share/anaconda/** contient une feuille de style CSS nommée **anaconda-gtk.css**, qui détermine les noms de fichiers et les paramètres des principaux éléments de l'interface utilisateur - le logo et les arrière-plans de la barre latérale et de la barre supérieure. Le fichier contient les éléments suivants, qui peuvent être personnalisés en fonction de vos besoins :

```
/* 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;
}
```

La partie la plus importante du fichier CSS est la manière dont il gère la mise à l'échelle en fonction de la résolution. Les images d'arrière-plan au format PNG ne sont pas mises à l'échelle, elles sont toujours affichées dans leurs dimensions réelles. Au lieu de cela, les arrière-plans ont un fond transparent et la feuille de style définit une couleur d'arrière-plan correspondante sur la ligne **@define-color**. Par conséquent, l'arrière-plan *images* "fade" dans l'arrière-plan *color*, ce qui signifie que les arrière-plans fonctionnent sur toutes les résolutions sans qu'il soit nécessaire de mettre l'image à l'échelle.

Vous pouvez également modifier les paramètres de **background-repeat** pour créer des tuiles sur l'arrière-plan ou, si vous êtes certain que tous les systèmes que vous installerez auront la même résolution d'affichage, vous pouvez utiliser des images d'arrière-plan qui remplissent l'ensemble de la barre.

Tous les fichiers énumérés ci-dessus peuvent être personnalisés. Une fois que vous l'avez fait, suivez les instructions de la section 2.2, "Création d'un fichier product.img" pour créer votre propre fichier product.img avec des graphiques personnalisés, puis de la section 2.3, "Création d'images d'amorçage personnalisées" pour créer une nouvelle image ISO amorçable avec vos modifications.

4.2. PERSONNALISER LE NOM DU PRODUIT

Pour personnaliser le nom du produit, vous devez créer un fichier personnalisé **.buildstamp file**. Pour ce faire, créez un nouveau fichier **.buildstamp.py** avec le contenu suivant :

```
[Main]
Product=My Distribution
Version=9
BugURL=https://bugzilla.redhat.com/
IsFinal=True
UUID=202007011344.x86_64
[Compose]
Lorax=28.14.49-1
```

Remplacez *My Distribution* par le nom que vous souhaitez afficher dans le programme d'installation.

Après avoir créé le fichier .buildstamp personnalisé, suivez les étapes de la section Création d'un fichier product.im g pour créer un nouveau fichier product.img contenant vos personnalisations, et de la section Création d'images de démarrage personnalisées pour créer un nouveau fichier ISO amorçable contenant vos modifications.

4.3. PERSONNALISATION DE LA CONFIGURATION PAR DÉFAUT

Vous pouvez créer votre propre fichier de configuration et l'utiliser pour personnaliser la configuration du programme d'installation.

4.3.1. Configuration des fichiers de configuration par défaut

Vous pouvez écrire les fichiers de configuration Anaconda au format **.ini**. Le fichier de configuration Anaconda se compose de sections, d'options et de commentaires. Chaque section est définie par un en-tête **[section]**, les commentaires commençant par un caractère **#** et les clés pour définir les options **options**. Le fichier de configuration résultant est traité avec l'analyseur de fichiers de configuration **configparser**.

Le fichier de configuration par défaut, situé à l'adresse **/etc/anaconda/anaconda.conf**, contient les sections documentées et les options prises en charge. Ce fichier fournit une configuration complète par défaut du programme d'installation. Vous pouvez modifier la configuration des fichiers de configuration du produit à partir de **/etc/anaconda/product.d/** et des fichiers de configuration personnalisés à partir de **/etc/anaconda/conf.d/**.

Le fichier de configuration suivant décrit la configuration par défaut de RHEL 9 :

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

# Enable Anaconda addons.
# This option is deprecated and will be removed in the future.
# addons_enabled = True
```

```
# List of enabled Anaconda DBus modules.
# This option is deprecated and will be removed in the future.
# kickstart_modules =

# List of Anaconda DBus modules that can be activated.
# Supported patterns: MODULE.PREFIX., MODULE.NAME activatable_modules =
org.fedoraproject.Anaconda.Modules.
    org.fedoraproject.Anaconda.Addons.*

# List of Anaconda DBus modules that are not allowed to run.
# Supported patterns: MODULE.PREFIX., MODULE.NAME forbidden_modules = # List of
Anaconda DBus modules that can fail to run. # The installation won't be aborted because of
them. # Supported patterns: MODULE.PREFIX., MODULE.NAME
optional_modules =
    org.fedoraproject.Anaconda.Modules.Subscription
    org.fedoraproject.Anaconda.Addons.*


[Installation System]

# Should the installer show a warning about enabled SMT?
can_detect_enabled_smt = False


[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

# Should we install the network configuration?
can_configure_network = True


[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 = NONE


[Payload]
# Default package environment.
default_environment =

# List of ignored packages.
ignored_packages =
```

```
# Names of repositories that provide latest updates.
updates_repositories =

# 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 = True

# Default installation source.
# Valid values:
#
#    CLOSEST_MIRROR  Use closest public repository mirror.
#    CDN             Use Content Delivery Network (CDN).
#
default_source = CLOSEST_MIRROR

# Enable ssl verification for all HTTP connection
verify_ssl = True

# GPG keys to import to RPM database by default.
# Specify paths on the installed system, each on a line.
# Substitutions for $releasever and $basearch happen automatically.
default_rpm_gpg_keys =

[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 = default

# 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 zfcp.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 =

# Default partitioning.
# Specify a mount point and its attributes on each line.
#
# Valid attributes:
#
#   size <SIZE>     The size of the mount point.
#   min <MIN_SIZE> The size will grow from MIN_SIZE to MAX_SIZE.
#   max <MAX_SIZE> The max size is unlimited by default.
#   free <SIZE>     The required available space.
#
default_partitioning =
    /     (min 1 GiB, max 70 GiB)
    /home (min 500 MiB, free 50 GiB)

# 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 to specify a swap partition?
swap_is_recommended = False

# 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 minimal 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 =

# Mount points that are recommended to be reformatted.
#
# It will be recommended to create a new file system on a mount point
# that has an allowed prefix, but doesn't have a blocked one.
# Specify lists of mount points.
reformat_allowlist = /boot /var /tmp /usr
reformat_blocklist = /home /usr/local /opt /var/www


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

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

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

# Should the UI allow to change the configured root account?
can_change_root = False

# Should the UI allow to change the configured user accounts?
can_change_users = False

# Define the default password policies.
# Specify a policy name and its attributes on each line.
#
# Valid attributes:
#
#   quality <NUMBER>  The minimum quality score (see libpwquality).
#   length <NUMBER>   The minimum length of the password.
#   empty             Allow an empty password.
#   strict            Require the minimum quality.
#
password_policies =
    root (quality 1, length 6)
    user (quality 1, length 6, empty)
    luks (quality 1, length 6)


[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 =
```

4.3.2. Configuration des fichiers de configuration du produit

Les fichiers de configuration des produits comportent une ou deux sections supplémentaires qui identifient le produit. La section **[Product]** indique le nom d'un produit. La section **[Base Product]** spécifie le nom d'un produit de base, le cas échéant. Par exemple, Red Hat Enterprise Linux est un produit de base de Red Hat Virtualization.

Le programme d'installation charge les fichiers de configuration des produits de base avant de charger le fichier de configuration du produit spécifié. Par exemple, il chargera d'abord la configuration de Red Hat Enterprise Linux, puis celle de Red Hat Virtualization.

Voir un exemple de fichier de configuration de produit pour Red Hat Enterprise Linux :

```
# Anaconda configuration file for Red Hat Enterprise Linux.

[Product]
product_name = Red Hat Enterprise Linux

[Installation System]

# Show a warning if SMT is enabled.
can_detect_enabled_smt = True

[Network]
default_on_boot = DEFAULT_ROUTE_DEVICE

[Payload]
ignored_packages =
    ntfsprogs
    btrfs-progs
    dmraid

enable_closest_mirror = False
default_source = CDN

[Bootloader]
efi_dir = redhat

[Storage]
file_system_type = xfs
default_partitioning =
    /     (min 1 GiB, max 70 GiB)
    /home (min 500 MiB, free 50 GiB)
    swap

[Storage Constraints]
swap_is_recommended = True

[User Interface]

help_directory = /usr/share/anaconda/help/rhel

[License]
eula = /usr/share/redhat-release/EULA
```

Voir un exemple de fichier de configuration de produit pour Red Hat Virtualization :

```
# Anaconda configuration file for Red Hat Virtualization.

[Product]
product_name = Red Hat Virtualization (RHVH)

[Base Product]
product_name = Red Hat Enterprise Linux

[Storage]
default_scheme = LVM_THINP
default_partitioning =
    /            (min 6 GiB)
    /home          (size 1 GiB)
    /tmp           (size 1 GiB)
    /var           (size 15 GiB)
    /var/crash     (size 10 GiB)
    /var/log       (size 8 GiB)
    /var/log/audit (size 2 GiB)
    swap

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

Pour personnaliser la configuration du programme d'installation pour votre produit, vous devez créer un fichier de configuration du produit. Créez un nouveau fichier nommé **my-distribution.conf**, dont le contenu est similaire à l'exemple ci-dessus. Remplacez *product_name* dans la section **[Product]** par le nom de votre produit, par exemple Ma distribution. Le nom du produit doit être le même que celui utilisé dans le fichier **.buildstamp**.

Après avoir créé le fichier de configuration personnalisé, suivez les étapes de la section Création d'un fichier product.img pour créer un fichier **new product.img** contenant vos personnalisations, et de la section Création d'images de démarrage personnalisées pour créer un nouveau fichier ISO amorçable contenant vos modifications.

4.3.3. Configuration des fichiers de configuration personnalisés

Pour personnaliser la configuration du programme d'installation indépendamment du nom du produit, vous devez créer un fichier de configuration personnalisé. Pour ce faire, créez un nouveau fichier nommé **100-my-configuration.conf** dont le contenu est similaire à l'exemple de la section Configuration des fichiers de configuration par défaut , en omettant les sections **[Product]** et **[Base Product]**.

Après avoir créé le fichier de configuration personnalisé, suivez les étapes de la section Création d'un fichier product.img pour créer un fichier **new product.img** contenant vos personnalisations, et de la section Création d'images de démarrage personnalisées pour créer un nouveau fichier ISO amorçable contenant vos modifications.

CHAPITRE 5. DÉVELOPPEMENT DE MODULES D'INSTALLATION

Cette section fournit des détails sur Anaconda et son architecture, et sur la manière de développer vos propres modules complémentaires. Les détails sur Anaconda et son architecture vous aident à comprendre le backend d'Anaconda et les différents points de connexion pour que les add-ons fonctionnent. Cela vous aidera également à développer vos propres add-ons.

5.1. INTRODUCTION À ANACONDA ET AUX MODULES COMPLÉMENTAIRES

Anaconda est le programme d'installation du système d'exploitation utilisé dans Fedora, Red Hat Enterprise Linux et leurs dérivés. Il s'agit d'un ensemble de modules et de scripts Python ainsi que de quelques fichiers supplémentaires tels que **Gtk** widgets (écrits en C), **systemd** units et **dracut** libraries. Ensemble, ils forment un outil qui permet aux utilisateurs de définir les paramètres du système résultant (cible), puis d'installer ce système sur une machine. Le processus d'installation comporte quatre étapes principales :

1. Préparer la destination de l'installation (généralement le partitionnement du disque)
2. Installer le paquet et les données
3. Installation et configuration du chargeur de démarrage
4. Configurer le système nouvellement installé

L'utilisation d'Anaconda vous permet d'installer Fedora, Red Hat Enterprise Linux et leurs dérivés, de trois manières différentes :

Using graphical user interface (GUI):

Il s'agit de la méthode d'installation la plus courante. L'interface permet aux utilisateurs d'installer le système de manière interactive, avec peu ou pas de configuration requise avant de commencer l'installation. Cette méthode couvre tous les cas d'utilisation courants, y compris la mise en place de configurations de partitionnement complexes.

L'interface graphique prend en charge l'accès à distance via **VNC**, ce qui vous permet d'utiliser l'interface graphique même sur des systèmes dépourvus de cartes graphiques ou de moniteurs connectés.

Using text user interface (TUI):

L'interface utilisateur fonctionne comme une imprimante monochrome, ce qui lui permet de fonctionner sur des consoles série qui ne prennent pas en charge le déplacement du curseur, les couleurs et d'autres fonctions avancées. Le mode texte est limité et ne permet de personnaliser que les options les plus courantes, telles que les paramètres réseau, les options linguistiques ou la source d'installation (paquet) ; les fonctions avancées telles que le partitionnement manuel ne sont pas disponibles dans cette interface.

Using Kickstart file:

Un fichier Kickstart est un fichier texte brut dont la syntaxe s'apparente à celle d'un shell et qui peut contenir des données destinées à piloter le processus d'installation. Un fichier Kickstart vous permet d'automatiser partiellement ou totalement l'installation. Un ensemble de commandes configurant toutes les zones requises est nécessaire pour automatiser complètement l'installation. Si une ou plusieurs commandes sont manquantes, l'installation nécessite une interaction.

Outre l'automatisation du programme d'installation lui-même, les fichiers Kickstart peuvent contenir des scripts personnalisés qui sont exécutés à des moments précis du processus d'installation.

5.2. ANACONDA ARCHITECTURE

Anaconda est un ensemble de modules et de scripts Python. Il utilise également plusieurs paquets et bibliothèques externes. Les principaux composants de cet ensemble d'outils comprennent les paquets suivants :

- **pykickstart** - analyse et valide les fichiers Kickstart. Il fournit également une structure de données qui stocke les valeurs qui pilotent l'installation.
- **dnf** - le gestionnaire de paquets qui installe les paquets et résout les dépendances
- **blivet** - s'occupe de toutes les activités liées à la gestion du stockage
- **pyanaconda** - contient l'interface utilisateur et les modules pour **Anaconda**il fournit également divers utilitaires permettant d'exécuter des fonctions orientées système, telles que la sélection du clavier et du fuseau horaire, la configuration du réseau et la création d'utilisateurs. Il fournit également divers utilitaires permettant d'exécuter des fonctions orientées système
- **python-meh** - contient un gestionnaire d'exception qui recueille et stocke des informations système supplémentaires en cas de crash et transmet ces informations à la bibliothèque **libreport**, qui fait elle-même partie du projet ABRT
- **dasbus** - permet la communication entre la bibliothèque **D-Bus** et les modules d'anaconda ainsi qu'avec des composants externes
- **python-simpleline** - bibliothèque de l'interface utilisateur texte pour gérer l'interaction de l'utilisateur dans le mode texte **Anaconda**
- **gtk** - la bibliothèque de la boîte à outils Gnome pour la création et la gestion d'interfaces graphiques

Outre la division en paquets mentionnée précédemment, **Anaconda** est divisé en interne entre l'interface utilisateur et un ensemble de modules qui s'exécutent en tant que processus distincts et communiquent à l'aide de la bibliothèque **D-Bus**. Ces modules sont les suivants

- **Boss** - gère la découverte, le cycle de vie et la coordination des modules internes
- **Localization** - gère les locales
- **Network** - gère le réseau
- **Payloads** - gère les données à installer dans différents formats, tels que **rpm**, **ostree**, **tar** et d'autres formats d'installation. Les charges utiles gèrent les sources de données pour l'installation ; les sources peuvent varier en format comme CD-ROM, disque dur, NFS, URL, et autres sources
- **Security** - gère les aspects liés à la sécurité
- **Services** - services de manutention
- **Storage** - gère le stockage à l'aide de **blivet**
- **Subscription** - gère l'outil **subscription-manager** et Insights.

- **Timezone** - traite de l'heure, de la date, des zones et de la synchronisation de l'heure.
- **Users** - crée des utilisateurs et des groupes.

Chaque module déclare les parties de Kickstart qu'il gère et dispose de méthodes pour appliquer la configuration de Kickstart à l'environnement d'installation et au système installé.

La partie du code Python d'Anaconda (**pyanaconda**) démarre en tant que processus "principal" qui possède l'interface utilisateur. Toutes les données Kickstart que vous fournissez sont analysées à l'aide du module **pykickstart** et le module **Boss** est démarré, il découvre tous les autres modules et les démarre. Le processus principal envoie ensuite les données Kickstart aux modules en fonction de leurs capacités déclarées. Les modules traitent les données, appliquent la configuration à l'environnement d'installation et l'interface utilisateur valide si tous les choix requis ont été faits. Si ce n'est pas le cas, vous devez fournir les données dans un mode d'installation interactif. Une fois que tous les choix requis ont été faits, l'installation peut commencer - les modules écrivent des données sur le système installé.

5.3. INTERFACE UTILISATEUR D'ANACONDA

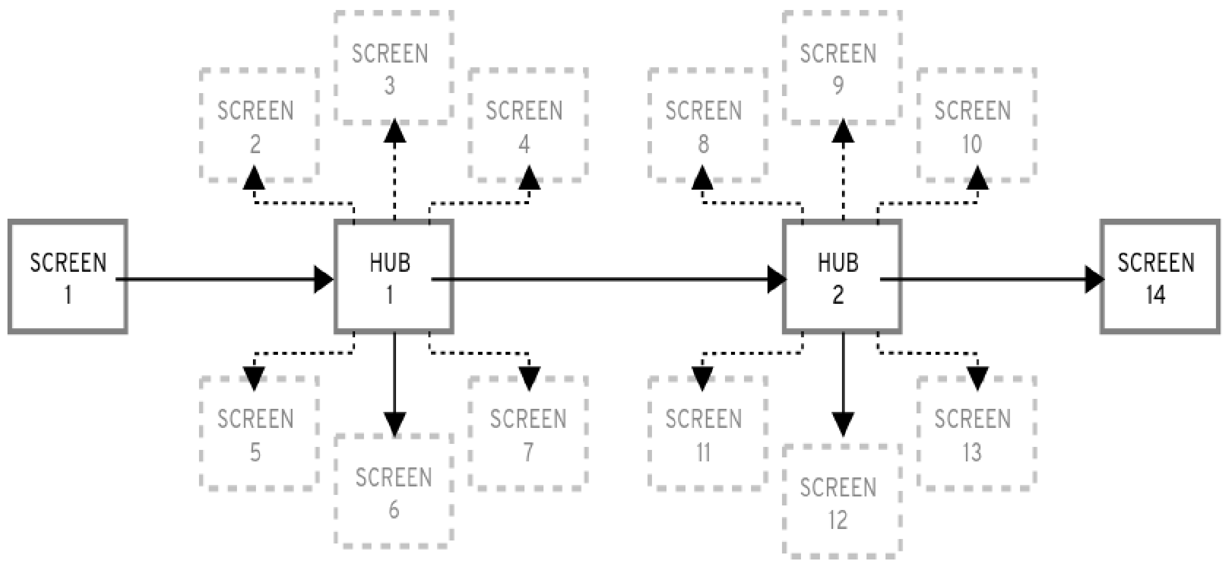
L'interface utilisateur (IU) d'Anaconda a une structure non linéaire, également connue sous le nom de modèle "hub and spoke".

Les avantages du modèle **Anaconda** modèle en étoile sont les suivants :

- Flexibilité pour suivre les écrans de l'installateur.
- Possibilité de conserver les paramètres par défaut.
- Fournit une vue d'ensemble des valeurs configurées.
- Favorise l'extensibilité. Vous pouvez ajouter des concentrateurs sans avoir à réorganiser quoi que ce soit et vous pouvez résoudre certaines dépendances complexes en matière d'ordre.
- Prise en charge de l'installation en mode graphique et en mode texte.

Le diagramme suivant montre la disposition de l'installateur et les interactions possibles entre *hubs* et *spokes* (écrans) :

Figure 5.1. Modèle de moyeu et de rayon



Dans le diagramme, les écrans 2 à 13 sont appelés *normal spokes*, et les écrans 1 et 14 sont *standalone spokes*. Les rayons autonomes sont les écrans qui peuvent être utilisés avant ou après le rayon ou le concentrateur autonome. Par exemple, l'écran **Welcome** au début de l'installation qui vous invite à choisir votre langue pour le reste de l'installation.

NOTE

- Le site **Installation Summary** est le seul hub d'Anaconda. Il affiche un résumé des options configurées avant le début de l'installation

Chaque rayon dispose des *properties* prédéfinis suivants qui reflètent le hub.

- **ready** - indique si vous pouvez ou non visiter un rayon. Par exemple, lorsque le programme d'installation est en train de configurer une source de paquets, le rayon est coloré en gris et vous ne pouvez pas y accéder tant que la configuration n'est pas terminée.
- **completed** - indique si le rayon est complet ou non (toutes les valeurs requises sont définies).
- **mandatory** - détermine si vous *must* devez visiter le rayon avant de poursuivre l'installation ; par exemple, vous devez visiter le rayon **Installation Destination**, même si vous voulez utiliser le partitionnement automatique du disque
- **status** - fournit un bref résumé des valeurs configurées dans le rayon (affiché sous le nom du rayon dans le hub)

Pour rendre l'interface utilisateur plus claire, les rayons sont regroupés dans la catégorie *categories*. Par exemple, la catégorie **Localization** regroupe les rayons relatifs à la sélection de la disposition du clavier, à la prise en charge de la langue et aux paramètres du fuseau horaire.

Chaque rayon contient des contrôles d'interface utilisateur qui affichent et permettent de modifier les valeurs d'un ou de plusieurs modules. Il en va de même pour les rayons fournis par les modules complémentaires.

5.4. COMMUNICATION ENTRE LES FILS D'ANACONDA

Certaines des actions que vous devez effectuer au cours du processus d'installation peuvent prendre beaucoup de temps. Par exemple, l'analyse des disques à la recherche de partitions existantes ou le téléchargement des métadonnées des paquets. Pour vous éviter d'attendre et rester réactif, **Anaconda** exécute ces actions dans des threads distincts.

La boîte à outils **Gtk** ne prend pas en charge les changements d'éléments à partir de plusieurs fils d'exécution. La boucle événementielle principale de **Gtk** s'exécute dans le thread principal du processus **Anaconda** processus. Par conséquent, toutes les actions relatives à l'interface graphique doivent être effectuées dans le thread principal. Pour ce faire, il faut utiliser **GLib.idle_add**, ce qui n'est pas toujours facile ou souhaitable. Plusieurs fonctions d'aide et décorateurs définis dans le module **pyanaconda.ui.gui.utils** peuvent ajouter à la difficulté.

Les décorateurs **@gtk_action_wait** et **@gtk_action_nowait** modifient la fonction ou la méthode décorée de telle sorte que lorsque cette fonction ou cette méthode est appelée, elle est automatiquement mise en file d'attente dans la boucle principale de Gtk qui s'exécute dans le fil d'exécution principal. La valeur de retour est soit renvoyée à l'appelant, soit abandonnée, respectivement.

Dans une communication entre un rayon et un moyeu, un rayon annonce qu'il est prêt et qu'il n'est pas bloqué. La file d'attente de messages **hubQ** gère cette fonction et vérifie périodiquement la boucle d'événements principale. Lorsqu'un rayon devient accessible, il envoie un message à la file d'attente annonçant le changement et le fait qu'il ne doit plus être bloqué.

Il en va de même lorsqu'un rayon doit actualiser son statut ou compléter un drapeau. Le hub **Configuration and Progress** dispose d'une file d'attente différente, appelée **progressQ**, qui sert à transférer les mises à jour de l'état d'avancement de l'installation.

Ces mécanismes sont également utilisés pour l'interface textuelle. En mode texte, il n'y a pas de boucle principale, mais la saisie au clavier prend la majeure partie du temps.

5.5. MODULES ANACONDA ET BIBLIOTHÈQUE D-BUS

Les modules d'Anaconda fonctionnent comme des processus indépendants. Pour communiquer avec ces processus via leur API **D-Bus**, utilisez la bibliothèque **dasbus**.

Les appels aux méthodes via l'API **D-Bus** sont asynchrones, mais la bibliothèque **dasbus** vous permet de les convertir en appels de méthodes synchrones en Python. Vous pouvez également écrire l'un des programmes suivants :

- programme avec des appels asynchrones et des gestionnaires de retour
- Un programme avec des appels synchrones qui fait attendre l'appelant jusqu'à ce que l'appel soit terminé.

Pour plus d'informations sur les threads et la communication, voir Communication entre les threads Anaconda.

En outre, Anaconda utilise des objets Task fonctionnant dans des modules. Les tâches ont une API **D-Bus** et des méthodes qui sont automatiquement exécutées dans des threads supplémentaires. Pour exécuter les tâches avec succès, utilisez les fonctions d'aide **sync_run_task** et **async_run_task**.

5.6. L'EXEMPLE DE L'ADDON HELLO WORLD

Les développeurs d'Anaconda publient un exemple d'addon appelé "Hello World", disponible sur GitHub : https://github.com/rhinstaller/hello-world-anaconda-addon/ Les descriptions des sections suivantes sont reproduites dans ce document.

5.7. STRUCTURE DE L'EXTENSION ANACONDA

Un add-on **Anaconda** est un paquetage Python qui contient un répertoire avec **__init__.py** et d'autres répertoires sources (sous-paquetages). Comme Python ne permet d'importer qu'une seule fois le nom de chaque paquetage, il convient de spécifier un nom unique pour le répertoire de premier niveau du paquetage. Vous pouvez utiliser un nom arbitraire, car les modules complémentaires sont chargés quel que soit leur nom - la seule exigence est qu'ils doivent être placés dans un répertoire spécifique.

La convention de dénomination suggérée pour les modules complémentaires est similaire aux paquets Java ou aux noms de services D-Bus.

Pour que le nom du répertoire soit un identifiant unique pour un paquetage Python, préfixez le nom du module d'extension par le nom de domaine inversé de votre organisation, en utilisant des traits de soulignement (_) au lieu de points. Par exemple, **com_example_hello_world**.

IMPORTANT

Veillez à créer un fichier **__init__.py** dans chaque répertoire. Les répertoires ne contenant pas ce fichier sont considérés comme des paquets Python non valides.

Lors de la rédaction d'un add-on, veillez aux points suivants :

- La prise en charge de chaque interface (interface graphique et interface textuelle) est disponible dans un sous-paquet distinct, nommé **gui** pour l'interface graphique et **tui** pour l'interface textuelle.
- Les paquets **gui** et **tui** contiennent un sous-paquet **spokes**. [1]
- Les modules contenus dans les paquets ont un nom arbitraire.
- Les répertoires **gui**/ et **tui**/ contiennent des modules Python de n'importe quel nom.
- Il existe un service qui effectue le travail réel de l'addon. Ce service peut être écrit en Python ou dans tout autre langage.
- Le service prend en charge D-Bus et Kickstart.
- L'addon contient des fichiers qui permettent le démarrage automatique du service.

Voici un exemple de structure de répertoire pour un module complémentaire qui prend en charge toutes les interfaces (Kickstart, GUI et TUI) :

Exemple 5.1. Exemple de structure de l'extension

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

Chaque paquetage doit contenir au moins un module avec un nom arbitraire définissant les classes héritées d'une ou plusieurs classes définies dans l'API.

NOTE

Pour tous les modules complémentaires, suivez les directives PEP 8 et PEP 257 de Python pour les conventions des chaînes de documentation. Il n'y a pas de consensus sur le format du contenu réel des chaînes de documentation en **Anaconda**; la seule exigence est qu'elles soient lisibles par l'homme. Si vous prévoyez d'utiliser une documentation générée automatiquement pour votre module complémentaire, les chaînes de documentation doivent suivre les directives de la boîte à outils que vous utilisez pour ce faire.

Vous pouvez inclure un sous-paquetage de catégorie si un module complémentaire doit définir une nouvelle catégorie, mais cela n'est pas recommandé.

5.8. SERVICES ET FICHIERS DE CONFIGURATION D'ANACONDA

Les services Anaconda et les fichiers de configuration sont inclus dans le répertoire data/. Ces fichiers sont nécessaires pour démarrer le service add-ons et pour configurer D-Bus.

Voici quelques exemples du module complémentaire Anaconda Hello World :

Exemple 5.2. Exemple de *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>
```

Ce fichier doit être placé dans le répertoire **/usr/share/anaconda/dbus/confs/** de l'environnement d'installation. La chaîne **org.fedoraproject.Anaconda.Addons.HelloWorld** doit correspondre à l'emplacement du service de l'addon sur D-Bus.

Exemple 5.3. Exemple de *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
```

Ce fichier doit être placé dans le répertoire **/usr/share/anaconda/dbus/services/** de l'environnement d'installation. La chaîne **org.fedoraproject.Anaconda.Addons.HelloWorld** doit correspondre à l'emplacement du service de l'addon sur D-Bus. La valeur de la ligne commençant par **Exec=** doit être une commande valide qui démarre le service dans l'environnement d'installation.

5.9. CARACTÉRISTIQUES DE BASE DU MODULE COMPLÉMENTAIRE GUI

De même que pour la prise en charge de Kickstart dans les modules complémentaires, la prise en charge de l'interface graphique exige que chaque partie du module complémentaire contienne au moins un module avec une définition d'une classe héritée d'une classe particulière définie par l'API. Pour la prise en charge du module complémentaire graphique, la seule classe que vous devez ajouter est la classe **NormalSpoke**, définie dans **pyanaconda.ui.gui.spokes**, comme classe pour le type d'écran normal. Pour en savoir plus, voir l'interface utilisateur d'Anaconda.

Pour mettre en œuvre une nouvelle classe héritée de **NormalSpoke**, vous devez définir les attributs de classe suivants, requis par l'API :

- **builderObjects** - liste tous les objets de premier niveau du fichier **.glade** du rayon qui doivent être exposés au rayon avec leurs objets enfants (récursivement). Si tout doit être exposé à spoke, ce qui n'est pas recommandé, la liste doit être vide.
- **mainWidgetName** - contient l'identifiant du widget de la fenêtre principale (Add Link) tel qu'il est défini dans le fichier **.glade**.
- **uiFile** - contient le nom du fichier **.glade**.
- **category** - contient la classe de la catégorie à laquelle appartient le rayon.
- **icon** - contient l'identifiant de l'icône qui sera utilisée pour le rayon sur le hub.
- **title** - définit le titre qui sera utilisé pour le rayon sur le concentrateur.

5.10. AJOUT DE LA PRISE EN CHARGE DE L'INTERFACE UTILISATEUR GRAPHIQUE (GUI) ADD-ON

Cette section décrit comment ajouter un support à l'interface utilisateur graphique (GUI) de votre module complémentaire en effectuant les étapes de haut niveau suivantes :

1. Définir les attributs requis pour la classe Normalspoke
2. Définir les méthodes **__init__** et **initialize**
3. Définir les méthodes **refresh**, **apply** et **execute**
4. Définir les propriétés **status** et **ready**, **completed** et **mandatory**

Conditions préalables

- Votre module complémentaire inclut la prise en charge de Kickstart. Voir la structure des modules complémentaires Anaconda.
- Installez les paquets anaconda-widgets et anaconda-widgets-devel, qui contiennent des widgets Gtk spécifiques à **Anaconda**, tels que **SpokeWindow**.

Procédure

- Créez les modules suivants avec toutes les définitions requises pour ajouter la prise en charge de l'interface utilisateur graphique (GUI) Add-on, conformément aux exemples suivants.

Exemple 5.4. Définition des attributs requis pour la classe Normalspoke :

```
# will never be translated
_ = lambda x: x
N_ = lambda x: x
```

```
# the path to addons is in sys.path so we can import things from org_fedora_hello_world
from org_fedora_hello_world.gui.categories.hello_world import HelloWorldCategory
from pyanaconda.ui.gui.spokes import NormalSpoke

# export only the spoke, no helper functions, classes or constants
all = ["HelloWorldSpoke"]

class HelloWorldSpoke(FirstbootSpokeMixIn, NormalSpoke):
    """
    Class for the Hello world spoke. This spoke will be in the Hello world
    category and thus on the Summary hub. It is a very simple example of a unit
    for the Anaconda's graphical user interface. Since it is also inherited form
    the FirstbootSpokeMixIn, it will also appear in the Initial Setup (successor
    of the Firstboot tool).

    :see: pyanaconda.ui.common.UIObject
    :see: pyanaconda.ui.common.Spoke
    :see: pyanaconda.ui.gui.GUIObject
    :see: pyanaconda.ui.common.FirstbootSpokeMixIn
    :see: pyanaconda.ui.gui.spokes.NormalSpoke

    """

    # class attributes defined by API #

    # list all top-level objects from the .glade file that should be exposed
    # to the spoke or leave empty to extract everything
    builderObjects = ["helloWorldSpokeWindow", "buttonImage"]

    # the name of the main window widget
    mainWidgetName = "helloWorldSpokeWindow"

    # name of the .glade file in the same directory as this source
    uiFile = "hello_world.glade"

    # category this spoke belongs to
    category = HelloWorldCategory

    # spoke icon (will be displayed on the hub)
    # preferred are the -symbolic icons as these are used in Anaconda's spokes
    icon = "face-cool-symbolic"

    # title of the spoke (will be displayed on the hub)
    title = N_("_HELLO WORLD")
```

L'attribut **__all__** exporte la classe **spoke**, suivie des premières lignes de sa définition, y compris les définitions des attributs mentionnés précédemment dans les caractéristiques de base du module d'extension GUI. Les valeurs de ces attributs font référence aux widgets définis dans le fichier **com_example_hello_world/gui/spokes/hello.glade**. Deux autres attributs notables sont présents :

- **category**dont la valeur est importée de la classe **HelloWorldCategory** du module **com_example_hello_world.gui.categories**. L'attribut **HelloWorldCategory** indique que le chemin d'accès aux modules complémentaires se trouve dans **sys.path**, de sorte que les valeurs peuvent être importées à partir du paquet **com_example_hello_world**. L'attribut **category** fait partie du nom **N_ function**, qui marque la chaîne pour la traduction, mais renvoie la version non traduite de la chaîne, car la traduction a lieu à un stade ultérieur.

- **title**qui contient un trait de soulignement dans sa définition. Le trait de soulignement de l'attribut **title** marque le début du titre lui-même et permet d'atteindre le rayon en utilisant le raccourci clavier **Alt H**.

Ce qui suit généralement l'en-tête de la définition de la classe et les définitions de la classe **attributes** est le constructeur qui initialise une instance de la classe. Dans le cas des objets de l'interface graphique Anaconda, il existe deux méthodes d'initialisation d'une nouvelle instance : la méthode **__init__** et la méthode **initialize**.

La raison pour laquelle il existe deux fonctions de ce type est que les objets GUI peuvent être créés en mémoire à un moment donné et entièrement initialisés à un autre moment, l'initialisation de **spoke** pouvant prendre beaucoup de temps. Par conséquent, la méthode **__init__** ne doit appeler que la méthode **__init__** du parent et, par exemple, initialiser les attributs non GUI. D'autre part, la méthode **initialize** qui est appelée lors de l'initialisation de l'interface graphique de l'installateur doit terminer l'initialisation complète des rayons.

Dans l'exemple **Hello World add-on**, définissez ces deux méthodes comme suit. Notez le nombre et la description des arguments transmis à la méthode **__init__**.

Exemple 5.5. Définition des méthodes__init__ et initialize :

```
def __init__(self, data, storage, payload):
    """
    :see: pyanaconda.ui.common.Spoke.init
    :param data: data object passed to every spoke to load/store data
    from/to it
    :type data: pykickstart.base.BaseHandler
    :param storage: object storing storage-related information
    (disks, partitioning, bootloader, etc.)
    :type storage: blivet.Blivet
    :param payload: object storing packaging-related information
    :type payload: pyanaconda.packaging.Payload

    """

    NormalSpoke.init(self, data, storage, payload)
    self._hello_world_module = HELLO_WORLD.get_proxy()

def initialize(self):
    """
    The initialize method that is called after the instance is created.
    The difference between init and this method is that this may take
    a long time and thus could be called in a separate thread.
    :see: pyanaconda.ui.common.UIObject.initialize
    """
    NormalSpoke.initialize(self)
    self._entry = self.builder.get_object("textLines")
    self._reverse = self.builder.get_object("reverseCheckButton")
```

Le paramètre data transmis à la méthode **__init__** est la représentation arborescente en mémoire du fichier Kickstart où toutes les données sont stockées. Dans l'une des méthodes **__init__** des ancêtres, il est stocké dans l'attribut **self.data**, ce qui permet à toutes les autres méthodes de la classe de lire et de modifier la structure.

NOTE

Le module **storage object** n'est plus utilisable depuis RHEL9. Si votre module complémentaire doit interagir avec la configuration du stockage, utilisez le module **Storage DBus**.

La classe HelloWorldData ayant déjà été définie dans l'exemple du module complémentaire Hello World, il existe déjà un sous-arbre dans self.data pour ce module complémentaire. Sa racine, une instance de la classe, est disponible à l'adresse **self.data.addons.com_example_hello_world**.

Une autre action de l'ancêtre **__init__** est d'initialiser une instance de GtkBuilder avec le fichier **spokeâs .glade** et de le stocker en tant que **self.builder**. La méthode **initialize** l'utilise pour obtenir le fichier **GtkTextEntry** utilisé pour afficher et modifier le texte de la section don du fichier kickstart.

Les méthodes **__init__** et **initialize** sont toutes deux importantes lors de la création du rayon. Cependant, le rôle principal du rayon est d'être visité par un utilisateur qui souhaite modifier ou revoir les valeurs du rayon. Pour ce faire, trois autres méthodes sont disponibles :

- **refresh** - appelée lorsque le rayon est sur le point d'être visité ; cette méthode rafraîchit l'état du rayon, principalement ses éléments d'interface utilisateur, pour s'assurer que les données affichées correspondent aux structures de données internes et, par conséquent, pour s'assurer que les valeurs actuelles stockées dans la structure self.data sont affichées.
- **apply** - appelée lorsque le rayon est quitté et utilisée pour stocker les valeurs des éléments de l'interface utilisateur dans la structure **self.data**.
- **execute** - appelé lorsque les utilisateurs quittent le rayon et utilisé pour effectuer tout changement d'exécution basé sur le nouvel état du rayon.

Ces fonctions sont mises en œuvre dans l'exemple de module complémentaire Hello World de la manière suivante :

Exemple 5.6. Définition des méthodes d'actualisation, d'application et d'exécution

```
def refresh(self):
    """
    The refresh method that is called every time the spoke is displayed.
    It should update the UI elements according to the contents of
    internal data structures.
    :see: pyanaconda.ui.common.UIObject.refresh
    """
    lines = self._hello_world_module.Lines
    self._entry.get_buffer().set_text("".join(lines))
    reverse = self._hello_world_module.Reverse
    self._reverse.set_active(reverse)

def apply(self):
    """
    The apply method that is called when user leaves the spoke. It should
    update the D-Bus service with values set in the GUI elements.
    """
```

```
        buf = self._entry.get_buffer()
        text = buf.get_text(buf.get_start_iter(),
                            buf.get_end_iter(),
                            True)
        lines = text.splitlines(True)
        self._hello_world_module.SetLines(lines)

        self._hello_world_module.SetReverse(self._reverse.get_active())

    def execute(self):
        """
        The execute method that is called when the spoke is exited. It is
        supposed to do all changes to the runtime environment according to
        the values set in the GUI elements.

        """

        # nothing to do here
        pass
```

Vous pouvez utiliser plusieurs méthodes supplémentaires pour contrôler l'état des rayons :

- **ready** - détermine si le rayon est prêt à être visité ; si la valeur est "False", le rayon **spoke** n'est pas accessible, par exemple, le rayon **Package Selection** avant qu'une source de paquets ne soit configurée.
- **completed** - détermine si le rayon a été achevé.
- **mandatory** - détermine si le rayon est obligatoire ou non, par exemple le rayon **Installation Destination**, qui doit toujours être visité, même si vous voulez utiliser le partitionnement automatique.

Tous ces attributs doivent être déterminés de manière dynamique en fonction de l'état actuel du processus d'installation.

Vous trouverez ci-dessous un exemple de mise en œuvre de ces méthodes dans le module complémentaire Hello World, qui exige qu'une certaine valeur soit définie dans l'attribut text de la classe **HelloWorldData**:

Exemple 5.7. Définir les méthodes "prêt", "complété" et "obligatoire

```
@property
def ready(self):
    """
    The ready property reports whether the spoke is ready, that is, can be visited
    or not. The spoke is made (in)sensitive based on the returned value of the ready
    property.

    :rtype: bool

    """

    # this spoke is always ready
    return True
```

```
@property
def mandatory(self):
    """
    The mandatory property that tells whether the spoke is mandatory to be
    completed to continue in the installation process.

    :rtype: bool

    """

    # this is an optional spoke that is not mandatory to be completed
    return False
```

Une fois ces propriétés définies, le rayon peut contrôler son accessibilité et son exhaustivité, mais il ne peut pas fournir un résumé des valeurs configurées à l'intérieur - vous devez visiter le rayon pour voir comment il est configuré, ce qui peut ne pas être souhaité. C'est pourquoi il existe une propriété supplémentaire appelée **status**. Cette propriété contient une seule ligne de texte avec un bref résumé des valeurs configurées, qui peut ensuite être affiché dans le hub sous le titre du rayon.

La propriété "status" est définie comme suit dans l'exemple de module complémentaire **Hello World**:

Exemple 5.8. Définition de la propriété status

```
@property
def status(self):
    """
    The status property that is a brief string describing the state of the
    spoke. It should describe whether all values are set and if possible
    also the values themselves. The returned value will appear on the hub
    below the spoke's title.
    :rtype: str
    """
    lines = self._hello_world_module.Lines
    if not lines:
        return _("No text added")
    elif self._hello_world_module.Reverse:
        return _("Text set with {} lines to reverse").format(len(lines))
    else:
        return _("Text set with {} lines").format(len(lines))
```

Après avoir défini toutes les propriétés décrites dans les exemples, le module complémentaire permet d'afficher une interface utilisateur graphique (GUI) ainsi que Kickstart.

NOTE

L'exemple présenté ici est très simple et ne contient aucun contrôle ; des connaissances en programmation Python Gtk sont nécessaires pour développer un rayon fonctionnel et interactif dans l'interface graphique.

Une restriction notable est que chaque rayon doit avoir sa propre fenêtre principale - une instance du widget **SpokeWindow**. Ce widget, ainsi que d'autres widgets spécifiques à Anaconda, se trouvent dans le paquetage **anaconda-widgets**. Vous trouverez d'autres fichiers nécessaires au développement de modules complémentaires avec prise en charge de l'interface graphique, tels que les définitions de **Glade**, dans le paquetage **anaconda-widgets-devel**.

Une fois que votre module de support de l'interface graphique contient toutes les méthodes nécessaires, vous pouvez continuer avec la section suivante pour ajouter le support de l'interface utilisateur textuelle, ou vous pouvez continuer avec Déployer et tester un module complémentaire Anaconda et tester le module complémentaire.

5.11. FONCTIONNALITÉS AVANCÉES DE L'INTERFACE GRAPHIQUE DU MODULE COMPLÉMENTAIRE

Le paquetage **pyanaconda** contient plusieurs fonctions d'aide et d'utilité, ainsi que des constructions qui peuvent être utilisées par les hubs et les spokes. La plupart d'entre elles se trouvent dans le paquetage **pyanaconda.ui.gui.utils**.

L'exemple de module complémentaire **Hello World** démontre l'utilisation du gestionnaire de contenu **englightbox** qui **Anaconda** utilise également. Ce gestionnaire de contenu peut placer une fenêtre dans une boîte lumineuse pour augmenter sa visibilité et la mettre en évidence pour empêcher les utilisateurs d'interagir avec la fenêtre sous-jacente. Pour démontrer cette fonction, l'exemple de module complémentaire contient un bouton qui ouvre une nouvelle fenêtre de dialogue ; le dialogue lui-même est un HelloWorldDialog spécial héritant de la classe GUIObject, qui est définie dans pyanaconda.ui.gui.*init*.

La classe dialog définit la méthode run qui exécute et détruit un dialogue Gtk interne accessible par l'attribut self.window, qui est rempli à l'aide d'un attribut de classe mainWidgetName ayant la même signification. Par conséquent, le code définissant le dialogue est très simple, comme le montre l'exemple suivant :

Exemple 5.9. Définition d'un dialogue de boîte de dialogue

```
# every GUIObject gets ksdata in init
dialog = HelloWorldDialog(self.data)

# show dialog above the lightbox
with self.main_window.enlightbox(dialog.window):
    dialog.run()
```

Le code de l'exemple **Defining an englightbox Dialog** crée une instance du dialogue et utilise ensuite le gestionnaire de contexte enlightbox pour exécuter le dialogue dans un cadre lumineux. Le gestionnaire de contexte a une référence à la fenêtre du rayon et n'a besoin que de la fenêtre du dialogue pour instancier le cadre lumineux pour le dialogue.

Une autre fonction utile fournie par **Anaconda** est la possibilité de définir un rayon qui apparaîtra à la fois pendant l'installation et après le premier redémarrage. L'utilitaire **Initial Setup** est décrit dans Adding support for the Add-on graphical user interface (GUI) . Pour qu'un rayon soit disponible à la fois dans Anaconda et dans Initial Setup, il doit hériter de la classe spéciale **FirstbootSpokeMixIn**, également connue sous le nom de **mixin**, en tant que première classe héritée définie dans le module **pyanaconda.ui.common**.

Pour qu'un rayon soit disponible dans **Anaconda** et le mode de reconfiguration de l'Initial Setup, il doit hériter de la classe spéciale **FirstbootSpokeMixIn**, également connue sous le nom de **mixin**, en tant que première classe héritée définie dans le module **pyanaconda.ui.common**.

Si vous souhaitez qu'un rayon donné ne soit disponible que dans la configuration initiale, ce rayon doit hériter de la classe **FirstbootOnlySpokeMixIn**.

Pour qu'un rayon soit toujours disponible à la fois dans **Anaconda** et Initial Setup, le rayon doit redéfinir la méthode **should_run**, comme le montre l'exemple suivant :

Exemple 5.10. Redéfinir la méthode should_run

```
@classmethod
   def should_run(cls, environment, data):
   """Run this spoke for Anaconda and Initial Setup"""
   return True
```

Le paquetage **pyanaconda** offre de nombreuses fonctionnalités plus avancées, telles que les décorateurs **@gtk_action_wait** et **@gtk_action_nowait**, mais elles sortent du cadre de ce guide. Pour plus d'exemples, reportez-vous aux sources du programme d'installation.

5.12. CARACTÉRISTIQUES DE BASE DU MODULE COMPLÉMENTAIRE TUI

Anaconda prend également en charge une interface textuelle (TUI). Cette interface est plus limitée dans ses capacités, mais sur certains systèmes, elle peut être le seul choix pour une installation interactive. Pour plus d'informations sur les différences entre l'interface textuelle et l'interface graphique et sur les limites de l'interface utilisateur, voir Introduction à Anaconda et aux modules complémentaires.

NOTE

Pour ajouter la prise en charge de l'interface texte dans votre module complémentaire, créez un nouvel ensemble de sous-paquetages dans le répertoire tui, comme décrit dans la structure du module complémentaire Anaconda .

La prise en charge du mode texte dans le programme d'installation est basée sur la bibliothèque **simpleline**, qui ne permet qu'une interaction très simple avec l'utilisateur. L'interface en mode texte :

- Ne prend pas en charge le déplacement du curseur - au lieu de cela, elle agit comme une imprimante de ligne.
- Ne prend pas en charge les améliorations visuelles, telles que l'utilisation de couleurs ou de polices différentes, par exemple.

En interne, la boîte à outils **simpleline** comporte trois classes principales : **App**, **UIScreen** et **Widget**. Les widgets sont des unités contenant des informations à imprimer sur l'écran. Ils sont placés sur des écrans UIScreens qui sont commutés par une instance unique de la classe App. En plus des éléments de base, **hubs**, **spoke`s and `dialogs** contiennent tous divers widgets d'une manière similaire à l'interface graphique.

Les classes les plus importantes pour un module complémentaire sont **NormalTUISpoke** et diverses autres classes définies dans le paquetage **pyanaconda.ui.tui.spokes**. Toutes ces classes sont basées sur la classe **TUIObject**, qui est elle-même un équivalent de la classe **GUIObject** dont il est question dans les fonctionnalités avancées de l'interface utilisateur graphique du module complémentaire .

Chaque rayon de l'interface graphique est une classe Python héritant de la classe **NormalTUISpoke**, qui surcharge les arguments spéciaux et les méthodes définis par l'API. L'interface texte étant plus simple que l'interface graphique, il n'y a que deux arguments de ce type :

- **title** - détermine le titre du rayon, de la même manière que l'argument title dans l'interface graphique.
- **category** - détermine la catégorie du rayon sous forme de chaîne de caractères ; le nom de la catégorie n'est affiché nulle part, il n'est utilisé que pour le regroupement.

NOTE

L'interface utilisateur gère les catégories différemment de l'interface graphique. Il est recommandé d'assigner une catégorie préexistante à vos nouveaux rayons. La création d'une nouvelle catégorie nécessiterait de patcher Anaconda, et n'apporterait que peu d'avantages.

Chaque rayon est également censé remplacer plusieurs méthodes, à savoir ***init***, **initialize**, **refresh**, **refresh**, **apply**, **execute**, **input**, **prompt**, et **properties** (**ready**, **completed**, **mandatory**, et **status**).

Ressources supplémentaires

- Voir Ajouter un support pour l'interface graphique des modules complémentaires .

5.13. DÉFINITION D'UNE BRANCHE SIMPLE DE L'INTERFACE UTILISATEUR

L'exemple suivant montre la mise en œuvre d'une interface utilisateur textuelle (TUI) simple dans l'exemple de module complémentaire Hello World :

Conditions préalables

- Vous avez créé un nouvel ensemble de sous-paquets dans le répertoire tui comme décrit dans la structure des modules complémentaires Anaconda.

Procédure

- Créez des modules avec toutes les définitions nécessaires pour ajouter la prise en charge de l'interface utilisateur textuelle (TUI), conformément aux exemples suivants :

Exemple 5.11. Définition d'une branche simple de l'interface utilisateur

```
def __init__(self, *args, **kwargs):
    """
    Create the representation of the spoke.

    :see: simpleline.render.screen.UIScreen
    """
    super().__init__(*args, **kwargs)
    self.title = N_("Hello World")
    self._hello_world_module = HELLO_WORLD.get_proxy()
    self._container = None
    self._reverse = False
```

```
        self._lines = ""

    def initialize(self):
        """
        The initialize method that is called after the instance is created.
        The difference between __init__ and this method is that this may take
        a long time and thus could be called in a separated thread.

        :see: pyanaconda.ui.common.UIObject.initialize
        """
        # nothing to do here
        super().initialize()

    def setup(self, args=None):
        """
        The setup method that is called right before the spoke is entered.
        It should update its state according to the contents of DBus modules.

        :see: simpleline.render.screen.UIScreen.setup
        """
        super().setup(args)

        self._reverse = self._hello_world_module.Reverse
        self._lines = self._hello_world_module.Lines

        return True

    def refresh(self, args=None):
        """
        The refresh method that is called every time the spoke is displayed.
        It should generate the UI elements according to its state.

        :see: pyanaconda.ui.common.UIObject.refresh
        :see: simpleline.render.screen.UIScreen.refresh
        """
        super().refresh(args)

        self._container = ListColumnContainer(
            columns=1
        )
        self._container.add(
            CheckboxWidget(
                title="Reverse",
                completed=self._reverse
            ),
            callback=self._change_reverse
        )
        self._container.add(
            EntryWidget(
                title="Hello world text",
                value="".join(self._lines)
            ),
            callback=self._change_lines
        )

        self.window.add_with_separator(self._container)
```

```
def _change_reverse(self, data):
    """
    Callback when user wants to switch checkbox.
    Flip state of the "reverse" parameter which is boolean.
    """
    self._reverse = not self._reverse

def _change_lines(self, data):
    """
    Callback when user wants to input new lines.
    Show a dialog and save the provided lines.
    """
    dialog = Dialog("Lines")
    result = dialog.run()
    self._lines = result.splitlines(True)

def input(self, args, key):
    """
    The input method that is called by the main loop on user's input.

    * If the input should not be handled here, return it.
    * If the input is invalid, return InputState.DISCARDED.
    * If the input is handled and the current screen should be refreshed,
      return InputState.PROCESSED_AND_REDRAW.
    * If the input is handled and the current screen should be closed,
      return InputState.PROCESSED_AND_CLOSE.

    :see: simpleline.render.screen.UIScreen.input
    """
    if self._container.process_user_input(key):
        return InputState.PROCESSED_AND_REDRAW

    if key.lower() == Prompt.CONTINUE:
        self.apply()
        self.execute()
        return InputState.PROCESSED_AND_CLOSE

    return super().input(args, key)

def apply(self):
    """
    The apply method is not called automatically for TUI. It should be called
    in input() if required. It should update the contents of internal data
    structures with values set in the spoke.
    """
    self._hello_world_module.SetReverse(self._reverse)
    self._hello_world_module.SetLines(self._lines)

def execute(self):
    """
    The execute method is not called automatically for TUI. It should be called
    in input() if required. It is supposed to do all changes to the runtime
    environment according to the values set in the spoke.
```

```
"""
# nothing to do here
pass
```

NOTE

Il n'est pas nécessaire de surcharger la méthode ***init*** si elle n'appelle que la méthode de l'ancêtre ***init*** mais les commentaires de l'exemple décrivent de manière compréhensible les arguments transmis aux constructeurs des classes parlantes.

Dans l'exemple précédent :

- La méthode **setup** définit une valeur par défaut pour l'attribut interne du rayon à chaque entrée, qui est ensuite affichée par la méthode **refresh**, mise à jour par la méthode **input** et utilisée par la méthode **apply** pour mettre à jour les structures de données internes.
- La méthode **execute** a le même objectif que la méthode équivalente dans l'interface graphique ; dans ce cas, la méthode n'a pas d'effet.
- La méthode **input** est spécifique à l'interface texte ; il n'y a pas d'équivalent dans Kickstart ou GUI. Les méthodes **input** sont responsables de l'interaction avec l'utilisateur.
- La méthode **input** traite la chaîne saisie et prend des mesures en fonction de son type et de sa valeur. L'exemple ci-dessus demande une valeur quelconque et la stocke en tant qu'attribut interne (clé). Dans les modules complémentaires plus complexes, vous devez généralement effectuer des actions non triviales, telles que l'analyse des lettres en tant qu'actions, la conversion des nombres en nombres entiers, l'affichage d'écrans supplémentaires ou le basculement de valeurs booléennes.
- La valeur **return** de la classe d'entrée doit être soit l'énumération **InputState**, soit la chaîne **input** elle-même, au cas où cette entrée devrait être traitée par un écran différent. Contrairement au mode graphique, les méthodes **apply** et **execute** ne sont pas appelées automatiquement lorsqu'on quitte le rayon ; elles doivent être appelées explicitement depuis la méthode input. Il en est de même pour la fermeture (masquage) de l'écran du rayon : elle doit être appelée explicitement depuis la méthode **close**.

Pour afficher un autre écran, par exemple si vous avez besoin d'informations supplémentaires qui ont été saisies dans un autre rayon, vous pouvez instancier un autre **TUIObject** et utiliser **ScreenHandler.push_screen_modal()** pour l'afficher.

En raison des restrictions imposées par l'interface textuelle, les rayons de l'interface utilisateur ont tendance à avoir une structure très similaire, qui consiste en une liste de cases à cocher ou d'entrées qui doivent être cochées ou décochées et remplies par l'utilisateur.

5.14. UTILISATION DE NORMALTUISPOKE POUR DÉFINIR UN SPOKE D'INTERFACE TEXTE

L'exemple de la définition d'une interface utilisateur simple a montré une façon d'implémenter une interface utilisateur dont les méthodes gèrent l'impression et le traitement des données disponibles et fournies. Cependant, il existe une autre façon d'y parvenir en utilisant la classe **Normal EditTUISpoke** du paquetage **pyanaconda.ui.tui.spokes**. En héritant de cette classe, vous pouvez mettre en œuvre un rayon d'interface utilisateur typique en spécifiant uniquement les champs et les attributs qui doivent être définis dans ce rayon. L'exemple suivant en est la démonstration :

Conditions préalables

- Vous avez ajouté un nouvel ensemble de sous-paquetages dans le répertoire **TUI**, comme décrit dans la structure des modules complémentaires d'Anaconda.

Procédure

- Créez des modules avec toutes les définitions requises pour ajouter la prise en charge de l'interface utilisateur textuelle (TUI) du module complémentaire, conformément aux exemples suivants.

Exemple 5.12. Utilisation de NormalTUISpoke pour définir un spoke d'interface texte

```
class HelloWorldEditSpoke(NormalTUISpoke):
    """Example class demonstrating usage of editing in TUI"""

    category = HelloWorldCategory

    def init(self, data, storage, payload):
        """
        :see: simpleline.render.screen.UIScreen
        :param data: data object passed to every spoke to load/store data
                     from/to it
        :type data: pykickstart.base.BaseHandler
        :param storage: object storing storage-related information
                        (disks, partitioning, bootloader, etc.)
        :type storage: blivet.Blivet
        :param payload: object storing packaging-related information
        :type payload: pyanaconda.packaging.Payload
        """
        NormalTUISpoke.init(self, data, storage, payload)

        self.title = N_("Hello World Edit")
        self._container = None
        # values for user to set
        self._checked = False
        self._unconditional_input = ""
        self._conditional_input = ""

    def refresh(self, args=None):
        """
        The refresh method that is called every time the spoke is displayed.
        It should update the UI elements according to the contents of
        self.data.
        :see: pyanaconda.ui.common.UIObject.refresh
        :see: simpleline.render.screen.UIScreen.refresh
        :param args: optional argument that may be used when the screen is
                     scheduled
        :type args: anything
        """
        super().refresh(args)
        self._container = ListColumnContainer(columns=1)

        # add ListColumnContainer to window (main window container)
        # this will automatically add numbering and will call callbacks when required
        self.window.add(self._container)
```

```
            self._container.add(CheckboxWidget(title="Simple checkbox", completed=self._checked),
                        callback=self._checkbox_called)
            self._container.add(EntryWidget(title="Unconditional text input",
                                 value=self._unconditional_input),
                        callback=self._get_unconditional_input)

            # show conditional input only if the checkbox is checked
            if self._checked:
                self._container.add(EntryWidget(title="Conditional password input",
                                     value="Password set" if self._conditional_input
                                     else ""),
                            callback=self._get_conditional_input)

            self._window.add_separator()


        @property
        def completed(self):
            # completed if user entered something non-empty to the Conditioned input
            return bool(self._conditional_input)
        @property
        def status(self):
            return "Hidden input %s" % ("entered" if self._conditional_input
                                 else "not entered")

        def apply(self):
            # nothing needed here, values are set in the self.args tree
            pass
```

5.15. DÉPLOYER ET TESTER UN MODULE COMPLÉMENTAIRE ANACONDA

Vous pouvez déployer et tester votre propre module complémentaire Anaconda dans l'environnement d'installation. Pour ce faire, suivez les étapes suivantes :

Conditions préalables

- Vous avez créé un Add-on.
- Vous avez accès à vos fichiers **D-Bus**.

Procédure

1. Créez un répertoire ***DIR*** à l'endroit de votre choix.
2. Ajoutez les fichiers python de **Add-on** dans ***DIR*/usr/share/anaconda/addons/**.
3. Copiez votre fichier de service **D-Bus** dans ***DIR*/usr/share/anaconda/dbus/services/**.
4. Copiez votre fichier de configuration du service **D-Bus** sur **/usr/share/anaconda/dbus/confs/**.
5. Créer l'image *updates* image.

Accédez au répertoire ***DIR*** l'annuaire :

```
cd DIR
```

Localisez l'image *updates* l'image.

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

6. Extraire le contenu de l'image de démarrage ISO.

7. Utiliser l'image résultante ***updates*** l'image obtenue :

   a. Ajoutez le fichier ***updates*.img** dans le répertoire **images** de votre contenu ISO déballé.

   b. Reconditionner l'image.

   c. Configurez un serveur web pour fournir le fichier ***updates***.img au programme d'installation d'Anaconda via HTTP.

   d. Charger le fichier ***updates***.img au moment du démarrage en ajoutant la spécification suivante aux options de démarrage.

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

Pour obtenir des instructions spécifiques sur le déballage d'une image de démarrage existante, la création d'un fichier **product.img** et le reconditionnement de l'image, reportez-vous à Extraction d'images de démarrage Red Hat Enterprise Linux.

---

[1] Le paquet **gui** peut également contenir un sous-paquet **categories** si le module complémentaire doit définir une nouvelle catégorie, mais cela n'est pas recommandé.

CHAPITRE 6. RÉALISATION DES TÂCHES DE PERSONNALISATION DES POSTES

Pour compléter les personnalisations effectuées, effectuez les tâches suivantes :

- Créer un fichier image product.img (ne s'applique qu'aux personnalisations graphiques).
- Créer une image de démarrage personnalisée.

Cette section explique comment créer un fichier image product.img et créer une image de démarrage personnalisée.

6.1. CRÉATION D'UN FICHIER PRODUCT.IMG

Un fichier image **product.img** est une archive contenant de nouveaux fichiers d'installation qui remplacent les fichiers existants au moment de l'exécution.

Pendant le démarrage du système, **Anaconda** charge le fichier product.img à partir du répertoire images/ du support de démarrage. Il utilise ensuite les fichiers présents dans ce répertoire pour remplacer les fichiers portant un nom identique dans le système de fichiers du programme d'installation. Les fichiers remplacés personnalisent le programme d'installation (par exemple, pour remplacer les images par défaut par des images personnalisées).

Remarque : l'image **product.img** doit contenir une structure de répertoire identique à celle du programme d'installation. Pour plus d'informations sur la structure du répertoire du programme d'installation, voir le tableau ci-dessous.

Tableau 6.1. Structure du répertoire d'installation et contenu personnalisé

Type de contenu personnalisé	Emplacement du système de fichiers
Pixmaps (logo, barre latérale, barre supérieure, etc.)	**/usr/share/anaconda/pixmaps/**
Feuille de style de l'interface graphique	**/usr/share/anaconda/anaconda-gtk.css**
Compléments d'Anaconda	**/usr/share/anaconda/addons/**
Fichiers de configuration du produit	**/etc/anaconda/product.d/**
Fichiers de configuration personnalisés	**/etc/anaconda/conf.d/**
Fichiers de configuration du service DBus Anaconda	**/usr/share/anaconda/dbus/confs/**
Fichiers de service Anaconda DBus	**/usr/share/anaconda/dbus/services/**

La procédure ci-dessous explique comment créer un fichier **product.img**.

Procédure

1. Accédez à un répertoire de travail tel que **/tmp**, et créez un sous-répertoire nommé **product/**:

```
$ cd /tmp
```

2. Créer un sous-répertoire product/

```
$ mkdir product/
```

3. Créez une structure de répertoire identique à l'emplacement du fichier que vous souhaitez remplacer. Par exemple, si vous souhaitez tester un module complémentaire présent dans le répertoire **/usr/share/anaconda/addons** sur le système d'installation, créez la même structure dans votre répertoire de travail :

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

> **NOTE**
>
> Pour afficher le fichier d'exécution du programme d'installation, démarrez l'installation et passez à la console virtuelle 1 (**Ctrl**+**Alt**+**F1**), puis passez à la deuxième fenêtre **tmux** fenêtre (**Ctrl**+**b**+**2**). Une invite de l'interpréteur de commandes permettant de parcourir un système de fichiers s'ouvre.

4. Placez vos fichiers personnalisés (dans cet exemple, le module d'extension personnalisé pour **Anaconda**) dans le répertoire nouvellement créé :

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

5. Répétez les étapes 3 et 4 (créer une structure de répertoire et y placer les fichiers personnalisés) pour chaque fichier que vous souhaitez ajouter au programme d'installation.

6. Créez un fichier **.buildstamp** à la racine du répertoire. Le fichier **.buildstamp** décrit la version du système, le produit et plusieurs autres paramètres. L'exemple suivant est un fichier **.buildstamp** de Red Hat Enterprise Linux 8.4 :

```
[Main]
Product=Red Hat Enterprise Linux
Version=8.4
BugURL=https://bugzilla.redhat.com/
IsFinal=True
UUID=202007011344.x86_64
[Compose]
Lorax=28.14.49-1
```

Le paramètre **IsFinal** précise si l'image correspond à une version (GA) du produit (**True**) ou à une préversion telle que Alpha, Beta ou un jalon interne (**False**).

7. Naviguez jusqu'au répertoire **product/** et créez l'archive **product.img**:

```
$ cd product
```

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

Cela crée un fichier **product.img** un niveau au-dessus du répertoire **product/**.

8. Déplacez le fichier **product.img** dans le répertoire **images/** de l'image ISO extraite.

Le fichier product.img est maintenant créé et les personnalisations que vous souhaitez effectuer sont placées dans les répertoires respectifs.

NOTE

Au lieu d'ajouter le fichier **product.img** sur le support de démarrage, vous pouvez placer ce fichier dans un autre emplacement et utiliser l'option de démarrage **inst.updates=** dans le menu de démarrage pour le charger. Dans ce cas, le fichier image peut porter n'importe quel nom et être placé à n'importe quel endroit (clé USB, disque dur, serveur HTTP, FTP ou NFS), pour autant que cet endroit soit accessible depuis le système d'installation.

6.2. CRÉATION D'IMAGES DE DÉMARRAGE PERSONNALISÉES

Après avoir personnalisé les images de démarrage et la présentation de l'interface graphique, créez une nouvelle image qui inclut les modifications que vous avez apportées.

Pour créer des images de démarrage personnalisées, suivez la procédure ci-dessous.

Procédure

1. Assurez-vous que toutes vos modifications sont incluses dans le répertoire de travail. Par exemple, si vous testez un module complémentaire, veillez à placer le fichier **product.img** dans le répertoire **images/**.

2. Assurez-vous que votre répertoire de travail actuel est le répertoire de premier niveau de l'image ISO extraite - par exemple **/tmp/ISO/iso**.

3. Créez une nouvelle image ISO à l'aide de la fonction **genisoimage**:

```
# genisoimage -U -r -v -T -J -joliet-long -V "RHEL-9 Server.x86_64" -volset \N- "Serveur RHEL-9.x86_64" (en anglais) -A "Serveur RHEL-9.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 .
```

Dans l'exemple ci-dessus :

- Assurez-vous que les valeurs des options **-V**, **-volset**, et **-A** correspondent à la configuration du chargeur de démarrage de l'image, si vous utilisez la directive **LABEL=** pour les options qui nécessitent un emplacement pour charger un fichier sur le même disque. Si la configuration de votre chargeur de démarrage (**isolinux/isolinux.cfg** pour le BIOS et **EFI/BOOT/grub.cfg** pour l'UEFI) utilise la directive **inst.stage2=LABEL=*disk_label*** pour charger la deuxième étape du programme d'installation à partir du même disque, les étiquettes des disques doivent correspondre.

IMPORTANT

Dans les fichiers de configuration du chargeur de démarrage, remplacez tous les espaces dans les étiquettes de disque par **\x20**. Par exemple, si vous créez une image ISO avec un label **RHEL 9.0**, la configuration du chargeur de démarrage doit utiliser **RHEL\x209.0**.

- Remplacez la valeur de l'option **-o** (**-o ../NEWISO.iso**) par le nom de fichier de votre nouvelle image. La valeur indiquée dans l'exemple crée le fichier **NEWISO.iso** dans le répertoire *above*, le répertoire actuel.
  Pour plus d'informations sur cette commande, voir la page de manuel **genisoimage(1)**.

4. Implanter une somme de contrôle MD5 dans l'image. Notez que sans somme de contrôle MD5, la vérification de l'image peut échouer (option **rd.live.check** dans la configuration du chargeur de démarrage) et l'installation peut se bloquer.

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

Dans l'exemple ci-dessus, remplacez *../NEWISO.iso* par le nom de fichier et l'emplacement de l'image ISO que vous avez créée à l'étape précédente.

Vous pouvez maintenant écrire la nouvelle image ISO sur un support physique ou un serveur réseau pour la démarrer sur du matériel physique, ou vous pouvez l'utiliser pour commencer l'installation d'une machine virtuelle.

Ressources supplémentaires

- Pour obtenir des instructions sur la préparation du support de démarrage ou du serveur réseau, voir Exécution d'une installation avancée de RHEL 9 .
- Pour savoir comment créer des machines virtuelles avec des images ISO, voir Configuration et gestion de la virtualisation.