Red Hat Enterprise Linux 9

Anaconda 사용자 정의

Red Hat Enterprise Linux에서 설치 프로그램의 모양 변경 및 사용자 지정 애드온 생성

Last Updated: 2024-07-06

Red Hat Enterprise Linux 9 Anaconda 사용자 정의

Red Hat Enterprise Linux에서 설치 프로그램의 모양 변경 및 사용자 지정 애드온 생성

법적 공지

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

초록

Anaconda는 Red Hat Enterprise Linux에서 사용하는 설치 프로그램입니다. 환경에 RHEL을 설치할 때 기능을 확장하기 위해 Anaconda를 사용자 지정할 수 있습니다.

차례

RED HAT 문서에 관한 피드백 제공 3

1장. ANACONDA 사용자 지정 소개 4
1.1. ANACONDA 사용자 지정 소개 4

2장. 사전 사용자 지정 작업 수행 5
2.1. ISO 이미지 작업 5
2.2. RH 부팅 이미지 다운로드 5
2.3. RED HAT ENTERPRISE LINUX 부팅 이미지 추출 5

3장. 부팅 메뉴 사용자 정의 7
3.1. 부팅 메뉴 사용자 정의 7
3.2. BIOS 펌웨어가 있는 시스템 7
3.3. UEFI 펌웨어가 있는 시스템 10

4장. 그래픽 사용자 인터페이스 브랜딩 및 검사 13
4.1. 그래픽 요소 사용자 정의 14
4.2. 제품 이름 사용자 정의 15
4.3. 기본 구성 사용자 지정 16

5장. 설치 프로그램 추가 기능 개발 24
5.1. ANACONDA 및 애드온 소개 24
5.2. ANACONDA 아키텍처 25
5.3. ANACONDA 사용자 인터페이스 27
5.4. ANACONDA 스레드 간 통신 29
5.5. ANACONDA 모듈 및 D-BUS 라이브러리 30
5.6. HELLO WORLD 애드온의 예 30
5.7. ANACONDA 애드온 구조 30
5.8. ANACONDA 서비스 및 구성 파일 32
5.9. GUI 애드온 기본 기능 33
5.10. GUI(ADD-ON 그래픽 사용자 인터페이스)에 대한 지원 추가 34
5.11. GUI 고급 기능 40
5.12. TUI 애드온 기본 기능 42
5.13. 간단한 TUI SPOKE 정의 43
5.14. NORMALTUISPOKE를 사용하여 텍스트 인터페이스 SPOKE 정의 47
5.15. ANACONDA 애드온 배포 및 테스트 49

6장. 사용자 지정 후 작업 완료 51
6.1. PRODUCT.IMG 파일 생성 51
6.2. 사용자 정의 부팅 이미지 생성 53

RED HAT 문서에 관한 피드백 제공

문서에 대한 피드백에 감사드립니다. 어떻게 개선할 수 있는지 알려주십시오.

Jira를 통해 피드백 제출 (등록 필요)

1. Jira 웹 사이트에 로그인합니다.
2. 상단 탐색 모음에서 **생성** 을 클릭합니다.
3. **Summary** (요약) 필드에 설명 제목을 입력합니다.
4. **Description** (설명) 필드에 개선을 위한 제안을 입력합니다. 문서의 관련 부분에 대한 링크를 포함합니다.
5. 대화 상자 하단에서 **생성** 을 클릭합니다.

1장. ANACONDA 사용자 지정 소개

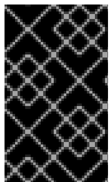
1.1. ANACONDA 사용자 지정 소개

Red Hat Enterprise Linux 및 Fedora 설치 프로그램인 **Anaconda** 는 최신 버전에서 많은 개선 사항을 제공합니다. 이러한 개선 사항 중 하나는 향상된 사용자 지정 기능입니다. 이제 애드온을 작성하여 기본 설치 프로그램 기능을 확장하고 그래픽 사용자 인터페이스의 모양을 변경할 수 있습니다.

이 문서에서는 다음을 사용자 지정하는 방법을 설명합니다.

- 부팅 메뉴 - 사전 구성된 옵션, 색상 체계 및 배경
- 그래픽 인터페이스 - 로고, 배경, 제품 이름
- 설치 프로그램 기능 - 그래픽 및 텍스트 사용자 인터페이스에서 새로운 Kickstart 명령 및 새 화면을 추가하여 설치 프로그램을 개선할 수 있는 애드온

또한 이 문서는 Red Hat Enterprise Linux 8 및 Fedora 17 이상에만 적용됩니다.



중요

이 문서에 설명된 절차는 Red Hat Enterprise Linux 9 또는 이와 유사한 시스템에 대해 작성되었습니다. 다른 시스템에서 사용되는 툴 및 애플리케이션(예: 사용자 지정 ISO 이미지 생성을 위한 **genisoimage** 등)은 다를 수 있으며 절차를 조정해야 할 수 있습니다.

지원 정책

Red Hat은 Red Hat Enterprise Linux Image Builder를 사용하여 Red Hat Enterprise Linux 설치 미디어 및 이미지 사용자 지정만 지원합니다. 또는 Kickstart를 사용하여 인프라에 일관된 시스템을 배포할 수 있습니다.

2장. 사전 사용자 지정 작업 수행

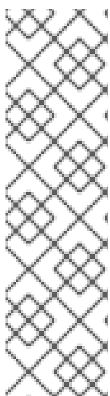
2.1. ISO 이미지 작업

이 섹션에서는 다음 방법에 대해 알아봅니다.

- Red Hat ISO 추출.
- 사용자 지정이 포함된 새 부팅 이미지를 생성합니다.

2.2. RH 부팅 이미지 다운로드

설치 프로그램을 사용자 지정하기 전에 Red Hat에서 제공하는 부팅 이미지를 다운로드하십시오. 계정 로그인 후 Red Hat 고객 포털에서 Red Hat Enterprise Linux 9 부팅 미디어를 받을 수 있습니다.



참고

- Red Hat Enterprise Linux 9 이미지를 다운로드할 수 있는 충분한 권한이 계정에 있어야 합니다.
- **바이너리 DVD** 또는 **부팅 ISO** 이미지를 다운로드해야 하며 모든 이미지 변형 (Server 또는 ComputeNode)을 사용할 수 있습니다.
- KVM 게스트 이미지 또는 추가 DVD와 같은 사용 가능한 다른 다운로드를 사용하여 설치 프로그램을 사용자 지정할 수 없습니다. **KVM 게스트 이미지** 또는 추가 **DVD** 와 같은 기타 사용 가능한 다운로드.

바이너리 DVD 및 부팅 ISO 다운로드에 대한 자세한 내용은 Red Hat Enterprise Linux 9 고급 RHEL 9 설치를 참조하십시오.

2.3. RED HAT ENTERPRISE LINUX 부팅 이미지 추출

부팅 이미지의 콘텐츠를 추출하려면 다음 절차를 수행합니다.

절차

1. ***/mnt/iso*** 디렉터리가 존재하고 현재 마운트되어 있지 않은지 확인합니다.
2. 다운로드한 이미지를 마운트합니다.

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

   여기서 *path/to/image.iso* 는 다운로드한 부팅 이미지의 경로입니다.
3. ISO 이미지의 콘텐츠를 배치하려는 작업 디렉터리를 생성합니다.

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

4. 마운트된 이미지의 모든 콘텐츠를 새 작업 디렉터리에 복사합니다. 파일 및 디렉터리 권한 및 소유권을 유지하기 위해 **-p** 옵션을 사용해야 합니다.

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

5. 이미지를 마운트 해제합니다.

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

추가 리소스

- 바이너리 DVD 및 부팅 ISO 다운로드에 대한 자세한 다운로드 지침 및 설명은 Red Hat Enterprise Linux 9 를 참조하십시오.

3장. 부팅 메뉴 사용자 정의

이 섹션에서는 부팅 메뉴 사용자 지정 및 사용자 지정 방법에 대한 정보를 제공합니다.

사전 요구 사항

부팅 이미지 다운로드 및 추출에 대한 자세한 내용은 Red Hat Enterprise Linux 부팅 이미지 추출을 참조하십시오.

부팅 메뉴 사용자 지정에는 다음과 같은 상위 수준 작업이 포함됩니다.

1. 사전 요구 사항을 완료합니다.
2. 부팅 메뉴를 사용자 지정합니다.
3. 사용자 정의 부팅 이미지를 생성합니다.

3.1. 부팅 메뉴 사용자 정의

부팅 메뉴는 설치 이미지를 사용하여 시스템을 부팅한 후 표시되는 메뉴입니다. 일반적으로 이 메뉴를 사용하면 **Red Hat Enterprise Linux 설치**,**로컬 드라이브 부팅** 또는 **설치된 시스템 복구와 같은 옵션 중에서 선택할 수** 있습니다. 부팅 메뉴를 사용자 지정하려면 다음을 수행할 수 있습니다.

- 기본 옵션을 사용자 지정합니다.
- 더 많은 옵션을 추가하십시오.
- 비주얼 스타일을 변경합니다(색 및 배경).

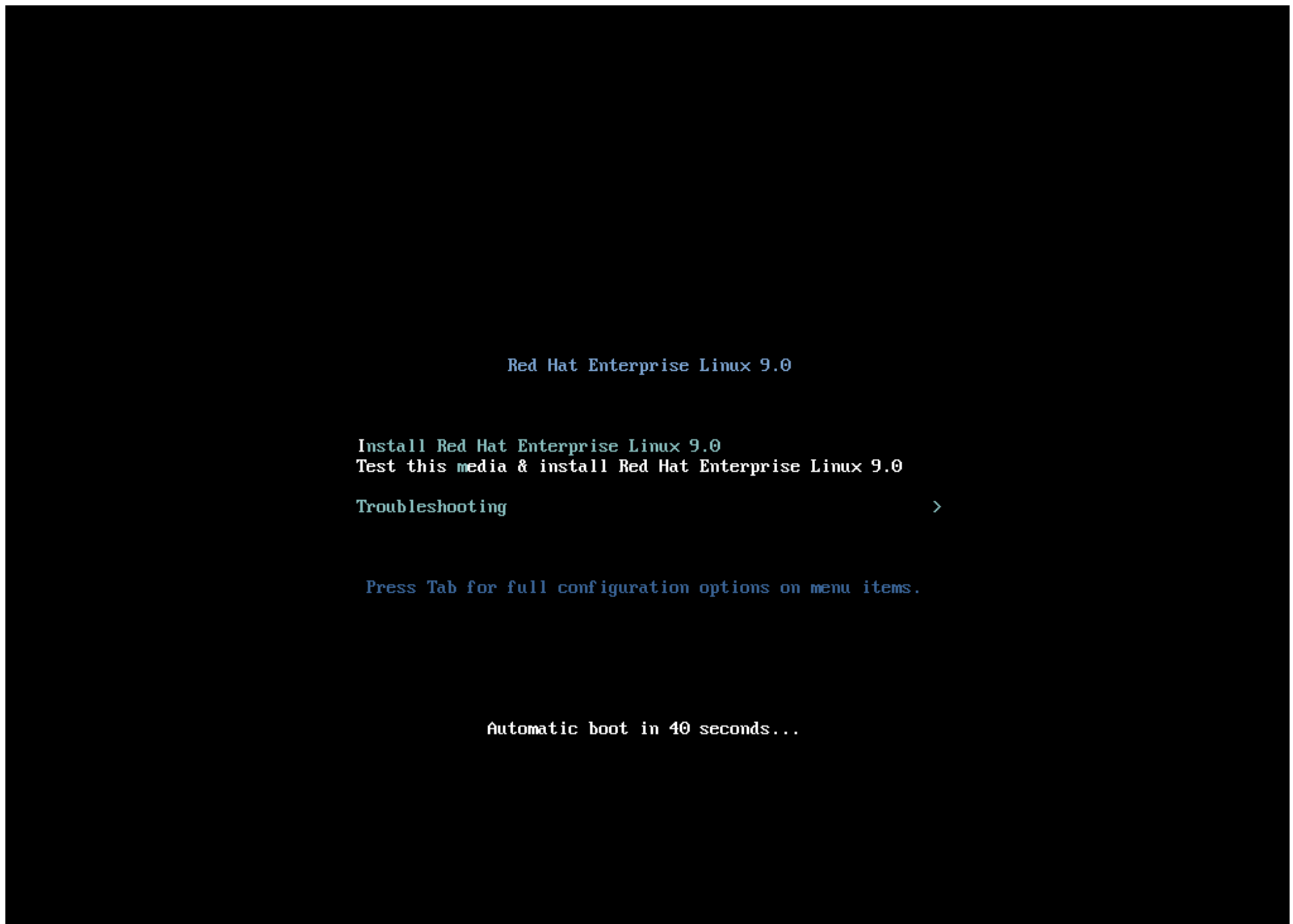
설치 미디어는 **ISOLINUX** 및 **GRUB2** 부트 로더로 구성됩니다. **ISOLINUX** 부트 로더는 BIOS 펌웨어가 있는 시스템에서 사용되며 **GRUB2** 부트 로더는 UEFI 펌웨어가 있는 시스템에서 사용됩니다. 부트 로더는 AMD64 및 Intel 64 시스템의 모든 Red Hat 이미지에 있습니다.

특히 Kickstart에 부팅 메뉴 옵션을 사용자 지정할 수 있습니다. 설치가 시작되기 전에 설치 프로그램에 Kickstart 파일을 제공해야 합니다. 일반적으로 기존 부팅 옵션 중 하나를 수동으로 편집하여 **inst.ks=** boot 옵션을 추가하여 수행합니다. 미디어에서 부트 로더 구성 파일을 편집하는 경우 사전 구성된 항목 중 하나에 이 옵션을 추가할 수 있습니다.

3.2. BIOS 펌웨어가 있는 시스템

ISOLINUX 부트 로더는 BIOS 펌웨어가 있는 시스템에서 사용됩니다.

그림 3.1. ISOLINUX 부팅 메뉴



부트 미디어의 ***isolinux/isolinux.cfg*** 구성 파일에는 색상 체계 및 메뉴 구조(entries 및 sub menu)를 설정하기 위한 지시문이 포함되어 있습니다.

Red Hat Enterprise Linux의 기본 메뉴 항목인 Red Hat Enterprise Linux의 기본 메뉴 항목인 **Test this media & Install Red Hat Enterprise Linux 9** 은 다음 블록에 정의되어 있습니다.

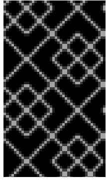
```
label check
  menu label Test this ^media & install Red Hat Enterprise Linux 9.
  menu default
  kernel vmlinuz
  append initrd=initrd.img inst.stage2=hd:LABEL=RHEL-9-BaseOS-x86_64 rd.live.check
quiet
```

다음과 같습니다.

- **메뉴 레이블** - 메뉴에서 항목의 이름을 지정하는 방법을 결정합니다. **^** 문자는 키보드 바로 가기(**m** 키)를 결정합니다.
- **메뉴 기본값** - 목록의 첫 번째 옵션이 아니지만 기본 선택을 제공합니다.
- **kernel** - 설치 프로그램 커널을 로드합니다. 대부분의 경우 변경해서는 안 됩니다.
- **append** - 추가 커널 옵션이 포함되어 있습니다. **initrd=** 및 **inst.stage2** 옵션은 필수입니다. 다른 옵션을 추가할 수 있습니다.
  Anaconda 에 적용되는 옵션에 대한 자세한 내용은 표준 RHEL 9 설치 가이드 수행 Red Hat Enterprise Linux 9에서 참조하십시오.

주목할 만한 옵션 중 하나는 **inst.ks=** 이며 Kickstart 파일의 위치를 지정할 수 있습니다. Kickstart 파일을 부팅 ISO 이미지에 배치하고 inst.ks= 옵션을 사용하여 위치를 지정할 수 있습니다. 예를 들어, Kickstart.ks 파일을 이미지의 루트 디렉터리에 배치하고 **inst.ks=hd:LABEL=RHEL-9-BaseOS-x86_64:/ kickstart.ks** 를 사용할 수 있습니다.

dracut.cmdline(7) 도움말 페이지에 나열된 dracut 옵션을 사용할 수도 있습니다.

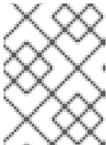


중요

디스크 레이블을 사용하여 특정 드라이브를 참조하는 경우 (**inst.stage2=hd:LABEL=RHEL-9-BaseOS-x86_64** 옵션 위의 참조) 모든 공백을 **\x20** 으로 교체합니다.

메뉴 항목 정의에 포함되지 않은 다른 중요한 옵션은 다음과 같습니다.

- **시간제한** - 기본 메뉴 항목이 자동으로 사용되기 전에 부팅 메뉴가 표시되는 시간을 결정합니다. 기본값은 **600** 이며, 이는 60 초 동안 메뉴가 표시됨을 의미합니다. 이 값을 **0** 으로 설정하면 timeout 옵션이 비활성화됩니다.



참고

시간 제한을 **1** 과 같은 낮은 값으로 설정하면 헤드리스 설치를 수행할 때 유용합니다. 이렇게 하면 기본 시간 초과를 완료하는 데 도움이 됩니다.

- **메뉴 시작** 및 **메뉴 종료** - *하위* 메뉴 블록의 시작 및 끝을 결정하여 문제 해결 및 하위 메뉴에서 그룹화와 같은 추가 옵션을 추가할 수 있습니다. 두 가지 옵션이 있는 간단한 하위 메뉴(보내기와 메인 메뉴로 돌아가는 방법)는 다음과 유사합니다.

```
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
```

하위 메뉴 항목 정의는 일반 메뉴 항목과 유사하지만 **메뉴 시작과 메뉴 끝점** 사이에 그룹화됩니다. 두 번째 옵션의 **메뉴 종료** 줄은 하위 메뉴를 종료하고 메인 메뉴로 돌아갑니다.

- **메뉴 배경** - 메뉴 배경은 솔리드 색상 (아래 **메뉴 색상** 참조) 또는 PNG, JPG 또는 LSS16 형식의 이미지일 수 있습니다. 이미지를 사용할 때는 해당 크기가 **set resolution** 문을 사용하여 설정한 해상도에 해당하는지 확인합니다. 기본 크기는 640x480입니다.

- **메뉴 색상** - 메뉴 요소의 색상을 결정합니다. 전체 형식은 다음과 같습니다.

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

이 명령의 가장 중요한 부분은 다음과 같습니다.

- *요소에* 따라 색상이 적용되는 요소를 결정합니다.
- *전경* 및 *배경* - 실제 색상을 확인합니다.
  색상은 16진수 형식의 ***#AARRGGBB*** 표기법을 사용하여 설명됩니다.
- **00** 완전히 투명하기 위한 것입니다.
- 완전히 불투명하기 위한 FF입니다.
- **메뉴 도움말 *텍스트* 파일 - 선택한 경우 도움말 텍스트 파일을 표시하는 메뉴 항목을 만듭니다.**

추가 리소스

- **ISOLINUX 구성 파일 옵션의 전체 목록은 Syslinux 6.0을 참조하십시오.**

3.3. UEFI 펌웨어가 있는 시스템

GRUB2 부트 로더는 UEFI 펌웨어가 있는 시스템에서 사용됩니다.

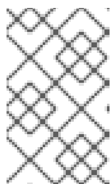
부트 미디어의 EFI/BOOT/grub.cfg 구성 파일에는 모양 및 부팅 메뉴 기능을 제어하는 사전 구성된 메뉴 항목 목록과 기타 지시문이 포함되어 있습니다.

구성 파일에서 Red Hat Enterprise Linux의 기본 메뉴 항목(이 미디어 테스트 및 Red Hat Enterprise Linux 9 설치)은 다음 블록에 정의됩니다.

```
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
}
```

다음과 같습니다.

- **메뉴 입력 - 항목의 제목을 정의합니다. 이는 단일 또는 이중 따옴표(' 또는 ")로 지정됩니다. --class 옵션을 사용하여 메뉴 항목을 다른 *클래스로* 그룹화할 수 있으며, GRUB2 주제를 사용하여 다르게 스타일을 지정할 수 있습니다.**



참고

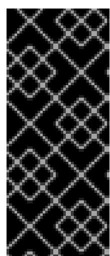
위 예제와 같이 각 메뉴 항목 정의를 **curly braces({})**로 묶어야 합니다.

linuxefi - 부팅되는 커널(위 예제에서/**images/pxeboot/vmlinuz**) 및 기타 추가 옵션(있는 경우)을 정의합니다.

이러한 옵션을 사용자 지정하여 부팅 항목의 동작을 변경할 수 있습니다. **Anaconda** 에 적용되는 옵션에 대한 자세한 내용은 *고급 RHEL 9 설치 수행을* 참조하십시오.

주목할 만한 옵션 중 하나는 **inst.ks=** 이며 **Kickstart** 파일의 위치를 지정할 수 있습니다. **Kickstart** 파일을 부팅 **ISO** 이미지에 배치하고 **inst.ks=** 옵션을 사용하여 위치를 지정할 수 있습니다. 예를 들어, **Kickstart.ks** 파일을 이미지의 루트 디렉터리에 배치하고 **inst.ks=hd:LABEL=RHEL-9-BaseOS-x86_64:/ kickstart.ks** 를 사용할 수 있습니다.

dracut.cmdline(7) 도움말 페이지에 나열된 **dracut** 옵션을 사용할 수도 있습니다.



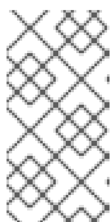
중요

디스크 레이블을 사용하여 특정 드라이브를 참조하는 경우 (**inst.stage2=hd:LABEL=RHEL-9-BaseOS-x86_64** 옵션 위의 참조) 모든 공백을 **\x20** 으로 교체합니다.

initrdefi - 로드할 초기 **RAM** 디스크(**initrd**) 이미지의 위치입니다.

grub.cfg 구성 파일에 사용되는 다른 옵션은 다음과 같습니다.

시간 제한 설정 - 기본 메뉴 항목이 자동으로 사용되기 전에 표시되는 부팅 메뉴의 기간을 결정합니다. 기본값은 **60** 이며, 이는 **60** 초 동안 메뉴가 표시됨을 의미합니다. 이 값을 **-1** 로 설정하면 시간 초과가 완전히 비활성화됩니다.



참고

이 설정은 기본 부팅 항목을 즉시 활성화하기 때문에 시간 초과를 **0** 으로 설정하면 헤드리스 설치를 수행할 때 유용합니다.

- 하위 메뉴 - *하위* 메뉴 블록을 사용하면 기본 메뉴에 표시하는 대신 하위 메뉴와 일부 항목을 그룹화할 수 있습니다. 기본 구성의 **Troubleshooting** 하위 메뉴에는 기존 시스템을 복구하기 위한 항목이 포함되어 있습니다.

  항목의 제목은 단일 또는 이중 따옴표(' 또는 ")입니다.

  하위 메뉴 블록에는 위에 설명된 대로 하나 이상의 메뉴 입력 정의가 포함되어 있으며 전체 블록은 curly braces({})로 묶습니다. 예를 들면 다음과 같습니다.

```
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
  }
}
```

- default 설정 - 기본 항목을 결정합니다. 입력 번호는 0 에서 시작합니다. *세 번째* 항목을 기본 항목으로 만들려면 set default=2 등을 사용합니다.

- Theme - GRUB2 Theme 파일이 포함된 디렉토리를 결정합니다. 주제를 사용하여 부트 로더의 시각적인 배경, 글꼴 및 특정 요소의 색을 사용자 지정할 수 있습니다.

추가 리소스

- 부팅 메뉴 사용자 지정에 대한 자세한 내용은 *GNU GRUB Manual 2.00* 을 참조하십시오.

- GRUB2 에 대한 자세한 내용은 *커널 관리, 모니터링 및 업데이트를* 참조하십시오.

4장. 그래픽 사용자 인터페이스 브랜딩 및 검사

Anaconda 사용자 인터페이스의 사용자 지정에는 그래픽 요소의 사용자 지정 및 제품 이름의 사용자 지정이 포함될 수 있습니다.

이 섹션에서는 그래픽 요소 및 제품 이름을 사용자 지정하는 방법에 대한 정보를 제공합니다.

사전 요구 사항

1. ISO 이미지를 다운로드하여 추출했습니다.

2. 당신은 당신의 자신의 브랜딩 자료를 만들 수 있습니다.

부팅 이미지 다운로드 및 추출에 대한 자세한 내용은 Red Hat Enterprise Linux 부팅 이미지 추출을참조하십시오.

사용자 인터페이스 사용자 지정에는 다음과 같은 상위 수준 작업이 포함됩니다.

1. 사전 요구 사항을 완료합니다.

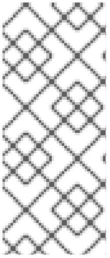
2. 사용자 정의 브랜딩 자료 만들기 (그래서 그래픽 요소를 사용자 정의하려는 경우)

3. 그래픽 요소 사용자 지정 (사용자 정의하려는 경우)

4. 제품 이름 사용자 정의 (사용자 정의하려는 경우)

5. product.img 파일 만들기

6. 사용자 정의 부팅 이미지 생성



참고

사용자 지정 브랜딩 자료를 만들려면 먼저 기본 그래픽 요소 파일 유형 및 크기를 참조하십시오. 따라서 사용자 정의 자료를 생성할 수 있습니다. 기본 그래픽 요소에 대한 자세한 내용은 그래픽 요소 사용자 지정 섹션에서 제공되는 샘플 파일에서 확인할 수 있습니다.

4.1. 그래픽 요소 사용자 정의

그래픽 요소를 사용자 정의하려면 사용자 정의 가능한 요소를 사용자 지정 브랜드 자료로 수정하거나 교체하고 컨테이너 파일을 업데이트할 수 있습니다.

설치 프로그램의 사용자 지정 가능 그래픽 요소는 설치 프로그램 런타임 파일 시스템의 **/usr/share/anaconda/pixmaps/** 디렉터리에 저장됩니다. 이 디렉터리에는 다음 사용자 지정 가능 파일이 포함되어 있습니다.

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

또한 **/usr/share/anaconda/** 디렉터리에는 기본 **UI** 요소의 파일 이름 및 매개 변수 - 로고 및 상단 표시줄의 배경을 결정하는 **anaconda-$k.css** 라는 **CSS** 스타일시트가 포함되어 있습니다. 파일에는 요구 사항에 따라 사용자 지정할 수 있는 다음 내용이 있습니다.

```
/* 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;
}
```

CSS 파일의 가장 중요한 부분은 해상도에 따라 스케일링을 처리하는 방법입니다. **PNG** 이미지 배경은 확장되지 않으며 항상 실제 크기에 표시됩니다. 대신 배경은 투명한 배경을 가지고 있으며 스타일시트는 **@define** 색상 라인에서 일치하는 배경 색상을 정의합니다. 따라서 배경 *이미지* "케이드"를 *배경색* 으로, 즉 이미지 확장 없이도 모든 해상도에서 배경 이미지가 작동합니다.

background-repeat 매개변수를 배경을 타일 타일로 변경하거나 설치하려는 모든 시스템에 동일한 표시 해상도가 있다고 확신하는 경우 전체 표시줄을 채우는 배경 이미지를 사용할 수도 있습니다.

위에 나열된 모든 파일을 사용자 지정할 수 있습니다. 이렇게 하면 **2.2**절의 지침에 따라 **"product.img File"**을 생성하여 사용자 지정 그래픽을 사용하여 고유한 **product.img**를 만든 다음, 섹션 **2.3, "**사용자 지정 부팅 이미지 생성"을 클릭하여 변경 사항이 포함된 새로운 부팅 가능한 **ISO** 이미지를 생성합니다.

4.2. 제품 이름 사용자 정의

제품 이름을 사용자 지정하려면 사용자 지정 **. buildstamp** 파일을 생성해야 합니다. 이렇게 하려면 다음 콘텐츠를 사용하여 새 파일 **.buildstamp.py** 를 생성합니다.

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

설치 *프로그램에 표시할 이름으로 내* 배포를 변경합니다.

사용자 지정 **.buildstamp** 파일을 생성한 후 **product.img 파일 생성** 섹션의 단계에 따라 사용자 지정이

포함된 새 product.img 파일을 생성하고 사용자 지정 부팅 이미지 생성 섹션을 사용하여 변경 사항이 포함된 새로운 부팅 가능 ISO 파일을 생성합니다.

4.3. 기본 구성 사용자 지정

고유한 구성 파일을 생성하고 이를 사용하여 설치 프로그램의 구성을 사용자 지정할 수 있습니다.

4.3.1. 기본 구성 파일 구성

Anaconda 구성 파일을 .ini 파일 형식으로 작성할 수 있습니다. Anaconda 구성 파일은 섹션, 옵션 및 주석으로 구성됩니다. 각 섹션은 [section] 헤더, # 문자로 시작하는 주석과 옵션을 정의하는 키로 정의합니다. 결과 구성 파일은 configparser 구성 파일 parser를 사용하여 처리됩니다.

/etc/anaconda/anaconda.conf 에 있는 기본 구성 파일에는 지원되는 문서화된 섹션 및 옵션이 포함되어 있습니다. 파일은 설치 프로그램의 전체 기본 구성을 제공합니다. /etc/anaconda/product.d/ 및 /etc/anaconda/conf.d/ 의 사용자 지정 구성 파일에서 제품 구성 파일의 구성을 수정할 수 있습니다.

다음 구성 파일은 RHEL 9의 기본 구성을 설명합니다.

```
[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 does not 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. 제품 구성 파일 구성

제품 구성 파일에는 제품을 식별하는 하나 또는 두 개의 추가 섹션이 있습니다. [Product] 섹션은 제품의 제품 이름을 지정합니다. [Base Product] 섹션은 해당하는 경우 기본 제품의 제품 이름을 지정합니다. 예를 들어 Red Hat Enterprise Linux는 Red Hat Virtualization의 기본 제품입니다.

설치 프로그램은 지정된 제품의 구성 파일을 로드하기 전에 기본 제품의 구성 파일을 로드합니다. 예를 들어, 먼저 Red Hat Enterprise Linux의 구성을 로드한 다음 Red Hat Virtualization을 위한 구성을 로드합니다.

Red Hat Enterprise Linux의 제품 구성 파일의 예를 참조하십시오.

```
# 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
```

Red Hat Virtualization의 제품 구성 파일의 예를 참조하십시오.

```
# 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
```

제품의 설치 프로그램 구성을 사용자 지정하려면 제품 구성 파일을 생성해야 합니다. 위 예제와 유사한 콘텐츠를 사용하여 **my-distribution.conf** 라는 새 파일을 만듭니다. **[Product]** 섹션의 ***product_name*** 을 제품 이름(예: **My Distribution**)으로 변경합니다. 제품 이름은 **.buildstamp** 파일에 사용된 이름과 동일해야 합니다.

사용자 지정 구성 파일을 생성한 후 product.img 파일 생성 섹션의 단계에 따라 사용자 지정이 포함된 새 **product.img** 파일을 생성하고 변경 사항이 포함된 새로운 부팅 가능 ISO 파일을 생성하도록 사용자 지정 부팅 이미지 생성을 수행합니다.

4.3.3. 사용자 지정 구성 파일 구성

제품 이름과 관계없이 설치 프로그램 구성을 사용자 지정하려면 사용자 지정 구성 파일을 생성해야 합니다. 이렇게 하려면 기본 구성 파일 구성의 예제와 유사한 내용을 사용하여 **100-my-configuration.conf** 라는 새 파일을 생성하고 **[Product]** 및 **[Base Product]** 섹션을 생략합니다.

사용자 지정 구성 파일을 생성한 후 product.img 파일 생성 섹션의 단계에 따라 사용자 지정이 포함된 새 **product.img** 파일을 생성하고 변경 사항이 포함된 새로운 부팅 가능 ISO 파일을 생성하도록 사용자 지정 부팅 이미지 생성을 수행합니다.

5장. 설치 프로그램 추가 기능 개발

이 섹션에서는 Anaconda와 아키텍처에 대한 세부 정보와 자체 애드온을 개발하는 방법을 설명합니다. Anaconda 및 해당 아키텍처에 대한 세부 정보는 Anaconda 백엔드 및 추가 기능 작동에 대한 다양한 플러그인 포인트를 이해하는 데 도움이 됩니다. 또한 애드온을 적절하게 개발하는 데 도움이 됩니다.

5.1. ANACONDA 및 애드온 소개

Anaconda 는 Fedora, Red Hat Enterprise Linux 및 해당 파생 제품에서 사용되는 운영 체제 설치 관리자입니다. Python 모듈과 스크립트 세트와 Gtk 위젯(C로 작성된), systemd 단위 및 dracut 라이브러리와 같은 몇 가지 추가 파일입니다. 이를 통해 사용자는 결과(대상) 시스템의 매개 변수를 설정한 다음 시스템에서 이 시스템을 설정할 수 있는 도구를 형성합니다. 설치 프로세스에는 다음과 같은 네 가지 주요 단계가 있습니다.

1. 설치 대상 준비 (일반적으로 디스크 파티션)
2. 패키지 및 데이터 설치
3. 부트 로더 설치 및 구성
4. 새로 설치된 시스템 구성

Anaconda를 사용하면 다음과 같은 세 가지 방법으로 Fedora, Red Hat Enterprise Linux 및 파생 제품을 설치할 수 있습니다.

GUI(그래픽 사용자 인터페이스) 사용:

가장 일반적인 설치 방법입니다. 인터페이스를 사용하면 설치를 시작하기 전에 필요한 구성이 거의 없거나 전혀 없이 대화형으로 시스템을 설치할 수 있습니다. 이 방법은 복잡한 파티션 레이아웃 설정을 포함하여 모든 일반적인 사용 사례를 다룹니다.

그래픽 카드나 연결된 모니터가 없는 시스템에서도 GUI를 사용할 수 있는 VNC 인터페이스를 통해 원격 액세스를 지원합니다.

텍스트 사용자 인터페이스(TUI) 사용:

TUI는 cursor 이동, 색상 및 기타 고급 기능을 지원하지 않는 직렬 콘솔에서 작동할 수 있는 monochrome 라인 프린터와 유사하게 작동합니다. 텍스트 모드는 제한되어 있으며 네트워크 설정, 언어 옵션 또는 설치(패키지) 소스와 같은 가장 일반적인 옵션만 사용자 지정할 수 있습니다. 수동 파티션과 같은 고급 기능은 이 인터페이스에서 사용할 수 없습니다.

Kickstart 파일 사용:

Kickstart 파일은 설치 프로세스를 구동하는 데이터를 포함할 수 있는 쉘과 유사한 구문이 있는 일반 텍스트 파일입니다. Kickstart 파일을 사용하면 설치를 부분적으로 또는 완전히 자동화할 수 있습니다. 설치를 완전히 자동화하려면 필요한 모든 영역을 구성하는 명령 세트가 필요합니다. 하나 이상의 명령이 누락된 경우 설치에 상호 작용이 필요합니다.

설치 프로그램 자체를 자동화하는 것 외에도 Kickstart 파일에는 설치 프로세스 중 특정 시점에 실행되는 사용자 지정 스크립트가 포함될 수 있습니다.

5.2. ANACONDA 아키텍처

Anaconda 는 Python 모듈 및 스크립트 세트입니다. 또한 여러 외부 패키지 및 라이브러리를 사용합니다. 이 툴셋의 주요 구성 요소에는 다음 패키지가 포함됩니다.

- **pykickstart** - Kickstart 파일을 구문 분석하고 검증합니다. 또한 설치를 구동하는 값을 저장하는 데이터 구조를 제공합니다.

- **DNF** - 패키지를 설치하고 종속성을 확인하는 패키지 관리자

- **Blivet** - 스토리지 관리와 관련된 모든 활동을 처리

- **pyanaconda** - 키보드 및 시간대 선택, 네트워크 구성 및 사용자 생성과 같은 Anaconda 용 사용자 인터페이스 및 모듈이 포함되어 있습니다. 또한 시스템 지향 기능을 수행하기 위한 다양한 유틸리티 제공

- **Python-meh** - 충돌시 추가 시스템 정보를 수집 및 저장하고 이 정보를 ABRT 프로젝트의 일부인 libreport 라이브러리에 전달합니다.

- **dasbus - anaconda**와 외부 구성 요소를 사용하여 **D-Bus** 라이브러리 간 통신 가능
- **Python-simpleline - Anaconda** 텍스트 모드에서 사용자 상호 작용을 관리하는 텍스트 **UI** 프레임워크 라이브러리
- **GTK - GUI**를 생성하고 관리하기 위한 **Gnome** 툴킷 라이브러리

이전에 언급된 패키지로의 분할 외에도 **Anaconda** 는 내부적으로 사용자 인터페이스와 별도의 프로세스로 실행되고 **D-Bus** 라이브러리를 사용하여 통신하는 모듈 세트로 나뉩니다. 이러한 모듈은 다음과 같습니다.

- 리더 **-** 내부 모듈 검색, 라이프사이클 및 조정을 관리합니다.
- 지역화 **-** 로켈 관리
- 네트워크 **-** 네트워크 처리
- 페이로드 **- rpm,ostree,tar** 및 기타 설치 형식과 같은 다양한 형식의 설치 데이터를 처리합니다. 페이로드는 설치를 위한 데이터 소스를 관리합니다. 소스는 **CD-ROM, HDD, NFS, URL** 및 기타 소스와 같은 형식으로 다양할 수 있습니다.
- 보안 **-** 보안 관련 측면 관리
- 서비스 **-** 서비스 처리
- 스토리지 **- blivet**를 사용하여 스토리지 관리
- 서브스크립션 **- subscription-manager** 도구 및 **Insights**를 처리합니다.

- 시간대 - 시간, 날짜, 영역 및 시간 동기화를 처리합니다.

- users - 사용자 및 그룹을 생성합니다.

각 모듈은 처리하는 Kickstart의 일부를 선언하고 Kickstart의 구성을 설치 환경 및 설치된 시스템에 적용하는 방법이 있습니다.

Anaconda의 Python 코드 부분(pyanaconda)은 사용자 인터페이스를 소유하는 "main" 프로세스로 시작됩니다. 제공하는 Kickstart 데이터는 pykickstart 모듈을 사용하여 구문 분석하고 Boss 모듈이 시작되고 다른 모든 모듈을 검색하고 시작합니다. 그런 다음 기본 프로세스는 선언된 기능에 따라 Kickstart 데이터를 모듈에 보냅니다. 모듈은 데이터를 처리하고 설치 환경에 구성을 적용하고 UI에서 필요한 모든 선택 항목이 수행되었는지 확인합니다. 그렇지 않은 경우 데이터를 대화형 설치 모드로 제공해야 합니다. 필요한 모든 선택 사항이 완료되면 설치를 시작할 수 있습니다. 모듈은 설치된 시스템에 데이터를 작성합니다.

5.3. ANACONDA 사용자 인터페이스

Anaconda 사용자 인터페이스(UI)에는 hub 및 spoke 모델라고도 하는 비선형 구조가 있습니다.

Anaconda 허브 및 스포크 모델의 장점은 다음과 같습니다.

- 설치 프로그램 화면을 따를 수 있는 유연성

- 기본 설정을 유지할 수 있는 유연성이 있습니다.

- 구성된 값에 대한 개요를 제공합니다.

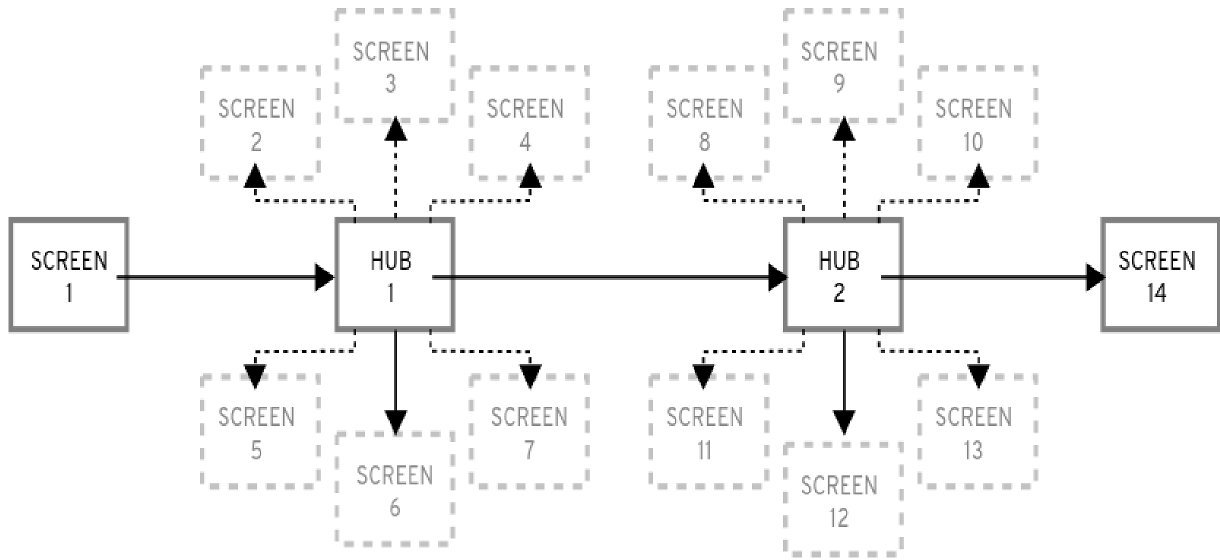
- 확장성을 지원합니다. 아무 것도 다시 정렬할 필요 없이 허브를 추가할 수 있으며 일부 복잡한 순서의 종속성을 해결할 수 있습니다.

- 그래픽 및 텍스트 모드로 설치를 지원합니다.

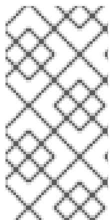
다음 다이어그램은 설치 프로그램 레이아웃과 *허브* 와 *스포크* (스크린) 간의 가능한 상호 작용을 보여

줍니다.

그림 5.1. hub 및 spoke model



다이어그램에서 화면 **2-13**은 *일반 대화 상자* 라고 하며 화면 **1**과 **14**는 *독립형 대화입니다*. 독립 실행형 대화는 독립 실행형 스포크 또는 허브 이전 또는 이후에 사용할 수 있는 화면입니다. 예를 들어 설치 시작 시 시작 부분에 있는 **welcome** 화면에서 나머지 설치에 대한 언어를 선택하라는 메시지가 표시됩니다.



참고

- 설치 요약 은 **Anaconda**의 유일한 허브입니다. 설치가 시작되기 전에 구성된 옵션에 대한 요약 정보가 표시됩니다.

각 **spoke**에는 허브를 반영하는 다음과 같은 사전 정의된 *속성이* 있습니다.

- **Ready** - 고객의 의견을 확인하실 수 있는지 여부입니다. 예를 들어 설치 프로그램이 패키지 소스를 구성할 때 **spoke**은 회색으로 색상이 지정되며 구성이 완료될 때까지 액세스할 수 없습니다.

- **completed** - 맞춤이 완료되었는지 여부를 표시합니다(모든 필수 값이 설정됨).

- **필수** - 설치를 계속하기 전에 스포크를 방문해야하는지 여부를 결정합니다. 예를 들어 자동 디스크 파티셔닝을 사용하려는 경우에도 설치 대상 대화 상자를 방문해야합니다.

- *status - 스포크 내에 구성된 값에 대한 간단한 요약을 제공합니다 (hub의 스포크 이름 아래에 표시됨)*

사용자 인터페이스를 더 명확하게하기 위해 대화는 카테고리 로 그룹화됩니다. 예를 들어, Localization 카테고리 그룹은 키보드 레이아웃 선택, 언어 지원 및 표준 시간대 설정을 함께 스포팅합니다.

각 대화 상자에는 하나 이상의 모듈에서 값을 표시하고 수정할 수 있는 UI 제어가 포함되어 있습니다. 이는 애드온이 제공하는 정책에도 동일하게 적용됩니다.

5.4. ANACONDA 스레드 간 통신

설치 프로세스 중에 수행해야 하는 일부 작업은 시간이 오래 걸릴 수 있습니다. 예를 들어 기존 파티션의 디스크를 스캔하거나 패키지 메타데이터를 다운로드합니다. 대기 및 나머지 응답 시간을 방지하기 위해 Anaconda 는 이러한 작업을 별도의 스레드에서 실행합니다.

Gtk 툴킷은 여러 스레드의 요소 변경을 지원하지 않습니다. Gtk 의 주요 이벤트 루프는 Anaconda 프로세스의 기본 스레드에서 실행됩니다. 따라서 GUI와 관련된 모든 작업은 기본 스레드에서 수행해야 합니다. 이렇게 하려면 GLib.idle_add 를 사용하십시오. 이는 항상 쉽고 원하는 것은 아닙니다. pyanaconda.ui.gui.utils 모듈에 정의된 여러 도우미 함수 및 데코레이터가 챌린지에 추가할 수 있습니다.

이 함수 또는 메서드를 호출할 때 해당 함수 또는 메서드를 호출할 때 기본 스레드에서 실행되는 Gtk의 기본 루프에 자동으로 큐에 큐에 저장되면 기본 스레드에서 실행되는 Gtk _action_ wait 및 @managek_action_nowait 데코레이터가 장식된 함수 또는 메서드를 변경합니다. 반환 값은 호출자로 반환되거나 각각 삭제됩니다.

대화 상자 및 허브 통신에서 대화 상대는 준비가 되어 있고 차단되지 않을 때 발표되었습니다. hubQ 메시지 큐는 이 기능을 처리하고 주기적으로 기본 이벤트 루프를 확인합니다. 대화 상자에 액세스 할 수 있게 되면 변경 사항을 발표하는 대기열에 메시지를 보내고 더 이상 차단해서는 안 됩니다.

spoke에서 상태를 새로 고치거나 플래그를 완료해야 하는 경우에도 동일하게 적용됩니다. 구성 및 진행률 허브에 는 설치 진행 상황 업데이트를 전송하기 위한 매체 역할을 하는 progressQ 라는 다른 큐가 있습니다.

이러한 메커니즘은 텍스트 기반 인터페이스에도 사용됩니다. 텍스트 모드에서는 기본 루프가 없지만 키보드 입력은 대부분의 시간을 사용합니다.

5.5. ANACONDA 모듈 및 D-BUS 라이브러리

Anaconda의 모듈은 독립적인 프로세스로 실행됩니다. D-Bus API를 통해 이러한 프로세스와 통신하려면 dasbus 라이브러리를 사용합니다.

D-Bus API를 통한 메서드 호출은 비동기식이지만 dasbus 라이브러리를 사용하면 Python에서 동기 메서드 호출로 변환할 수 있습니다. 다음 프로그램 중 하나를 작성할 수도 있습니다.

- 비동기 호출 및 반환 처리기를 사용한 프로그램
- 호출이 완료될 때까지 호출을 기다리는 동기 호출이 있는 프로그램입니다.

스레드 및 통신에 대한 자세한 내용은 Anaconda 스레드 간 통신을 참조하십시오.

또한 Anaconda는 모듈에서 실행되는 Task 오브젝트를 사용합니다. 작업에는 추가 스레드에서 자동으로 실행되는 D-Bus API 및 메서드가 있습니다. 작업을 성공적으로 실행하려면 sync_run_task 및 async_run_task 도우미 함수를 사용합니다.

5.6. HELLO WORLD 애드온의 예

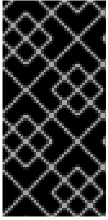
Anaconda 개발자는 GitHub에서 사용할 수 있는 "Hello World"라는 애드온을 게시합니다. https://github.com/rhinstaller/hello-world-anaconda-addon/ 추가 섹션의 설명은 이에서 재현됩니다.

5.7. ANACONDA 애드온 구조

Anaconda 애드온은 __init___.py 및 기타 소스 디렉터리(subpackages)가 있는 디렉토리를 포함하는 Python 패키지입니다. Python에서는 각 패키지 이름을 한 번만 가져올 수 있으므로 패키지 최상위 디렉터리에 대한 고유 이름을 지정합니다. add-ons는 이름에 관계없이 로드되므로 임의의 이름을 사용할 수 있습니다. 즉, 유일한 요구 사항은 특정 디렉터리에 배치되어야 한다는 것입니다.

애드온에 대한 제안된 이름 지정 규칙은 Java 패키지 또는 D-Bus 서비스 이름과 유사합니다.

디렉터리 이름을 Python 패키지의 고유 식별자로 만들려면 점 대신 밑줄(_)을 사용하여 조직의 역방향 도메인 이름으로 애드온 이름을 접두사로 지정합니다. 예: com_example_hello_world.



중요

*각 디렉터리에 **__init___.py** 파일을 생성해야 합니다. 이 파일이 누락된 디렉터리는 잘못된 **Python** 패키지로 간주됩니다.*

애드온을 작성할 때 다음을 확인합니다.

- *각 인터페이스(그래픽 인터페이스 및 텍스트 인터페이스)에 대한 지원은 별도의 하위 패키지에서 사용할 수 있으며 이러한 하위 패키지는 그래픽 인터페이스의 **gui** 와 텍스트 기반 인터페이스의 이름이 **gui**로 지정됩니다.*

- ***gui** 및 **tui** 패키지에는 **spokes** 하위 패키지가 포함되어 있습니다. **[1]***

- *패키지에 포함된 모듈에는 임의의 이름이 있습니다.*

- ***gui/** 및 **tui/** 디렉터리에는 모든 이름의 **Python** 모듈이 포함되어 있습니다.*

- *애드온의 실제 작업을 수행하는 서비스가 있습니다. 이 서비스는 **Python** 또는 다른 언어로 작성할 수 있습니다.*

- *이 서비스는 **D-Bus** 및 **Kickstart**에 대한 지원을 구현합니다.*

- ***addon**에는 서비스의 자동 시작을 활성화하는 파일이 포함되어 있습니다.*

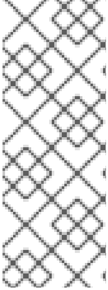
*다음은 모든 인터페이스 (**Kickstart, GUI** 및 **TUI**)를 지원하는 애드온의 샘플 디렉터리 구조입니다.*

*예 **5.1.** 샘플 애드온 구조*

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

```
├── init.py
└── spokes
└── init.py
```

각 패키지에는 API에 정의된 하나 이상의 클래스에서 상속된 클래스를 정의하는 임의의 이름이 포함된 하나 이상의 모듈이 포함되어야 합니다.



참고

모든 애드온의 경우 docstring 규칙에 대한 Python의 PEP 8 및 PEP 257 지침을 따르십시오. Anaconda 에서 docstrings의 실제 콘텐츠 형식에 대한 합의는 없으며, 유일한 요구 사항은 사람이 읽을 수 있다는 것입니다. 해당 애드온에 자동 생성된 문서를 사용하려는 경우 docstrings는 이를 수행하는 데 사용하는 툴킷의 지침을 따라야 합니다.

애드온에서 새 카테고리를 정의해야 하는 경우 카테고리 하위 패키지를 포함할 수 있지만 권장되지는 않습니다.

5.8. ANACONDA 서비스 및 구성 파일

Anaconda 서비스 및 구성 파일은 데이터/ 디렉터리에 포함되어 있습니다. 이러한 파일은 애드온 서비스를 시작하고 D-Bus를 구성하는 데 필요합니다.

다음은 Anaconda Hello World 애드온의 몇 가지 예입니다.

예 5.2. addon-name.conf의 예:

```
<!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>
```

이 파일은 설치 환경의 /usr/share/anaconda/dbus/confs/ 디렉터리에 있어야 합니다. org.fedoraproject.Anaconda.Addons.Hello world는 D-Bus의 addon 서비스의 위치에 대응해야 합니다.

예 5.3. addon-name.service의 예:

```
[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
```

이 파일은 설치 환경의 /usr/share/anaconda/dbus/services/ 디렉터리에 있어야 합니다. org.fedoraproject.Anaconda.Addons.Hello world는 D-Bus의 addon 서비스의 위치에 대응해야 합니다. Exec= 로 시작하는 행의 값은 설치 환경에서 서비스를 시작하는 유효한 명령이어야 합니다.

5.9. GUI 애드온 기본 기능

애드온의 Kickstart 지원과 마찬가지로, GUI 지원은 애드온의 모든 부분에 API에서 정의한 특정 클래스에서 상속된 클래스의 정의와 함께 하나 이상의 모듈을 포함해야 합니다. 그래픽 애드온 지원의 경우 추가할 유일한 클래스는 pyanaconda.ui.gui.spokes 에 정의된 NormalSpoke 클래스입니다. 자세한 내용은 Anaconda 사용자 인터페이스를 참조하십시오.

NormalSpoke 에서 상속된 새 클래스를 구현하려면 API에 필요한 다음 클래스 속성을 정의해야 합니다.

- builderObjects - 자식 객체(recursively)와 spoke에 노출되어야 하는 .glade 파일의 모든 최상위 오브젝트를 나열합니다. 모든 것이 spoke에 노출되어야 하는 경우에는 권장되지 않는 목록을 비어 있어야 합니다.
- mainWidgetName - .glade 파일에 정의된 기본 창 위젯(추가 링크)의 ID를 포함합니다.
- uiFile - . glade 파일 의 이름을 포함합니다.
- category - 스포크가 속한 카테고리의 클래스를 포함합니다.

- *icon - 허브에서 스포크에 사용할 아이콘의 식별자가 포함되어 있습니다.*

- *title - 허브에서 스포크에 사용할 제목을 정의합니다.*

5.10. GUI(ADD-ON 그래픽 사용자 인터페이스)에 대한 지원 추가

이 섹션에서는 다음과 같은 고급 단계를 수행하여 추가 기능의 GUI(그래픽 사용자 인터페이스)에 지원을 추가하는 방법에 대해 설명합니다.

1. **Normalspoke** 클래스에 필요한 속성 정의

2. **__init__** 및 초기화 방법 정의

3. 새로 고침 정의,적용, 실행 방법

4. 상태 및 준비 완료 및 필수 속성을 정의합니다.

사전 요구 사항

- 애드온에 **Kickstart** 지원이 포함되어 있습니다. Anaconda 애드온 구조를 참조하십시오.

- **Anaconda** 에 고유한 **Gtk** 위젯(예: **SpokeWindow**)이 포함된 **anaconda-widgets** 및 **anaconda-widgets-devel** 패키지를 설치합니다.

절차

- 다음 예와 같이 **Add-on** 그래픽 사용자 인터페이스(**GPU**)에 대한 지원을 추가하는 데 필요한 모든 정의로 다음 모듈을 생성합니다.

예 5.4. 일반 스포크 클래스에 필요한 속성 정의:

```
# 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")
```

__all__ 속성은 spoke 클래스를 내보낸 다음, GUI 애드온 기본 기능에서 이전에 언급한 속성 정의를 포함하여 첫 번째 줄로 구성됩니다. 이러한 특성 값은 com_example_hello_world/gui/spokes/hello.glade 파일에 정의된 위젯을 참조합니다. 다른 주목할 만한 두 가지 속성이 있습니다.

- category, com_example_hello_world.gui.categories 모듈의 Helloworld Category 클래스에서 가져온 값을 갖습니다. add-ons 경로가 sys.path에 있어 com_example_hello_world 패키지에서 값을 가져올 수 있도록 Helloworld Category that add-ons is in sys.path so that

values can be imported from the com_example_hello_world package. category 속성은 변환에 대한 문자열을 표시하는 N_ 함수 이름의 일부이지만 이후 단계에서 번역이 이루어지므로 문자열의 번역되지 않은 버전을 반환합니다.

- name, 해당 정의에 하나의 밑줄이 포함되어 있습니다. title 속성은 title 자체의 시작을 표시하고 Alt+H 키보드 바로 가기를 사용하여 대화 상자를 연결할 수 있도록 합니다.

일반적으로 클래스 정의의 헤더와 클래스 특성 정의는 클래스의 인스턴스를 초기화하는 생성자입니다. Anaconda 그래픽 인터페이스 오브젝트의 경우 새 인스턴스를 초기화하는 방법에는 __init__ 메서드와 initialize 메서드라는 두 가지 방법이 있습니다.

이러한 기능 뒤에 있는 이유는 GUI 객체가 한 번에 메모리에 생성될 수 있고 스포크 초기화가 시간 소모될 수 있기 때문에 다른 시간에 완전히 초기화될 수 있기 때문입니다. 따라서 __init__ 메서드는 상위의 __init__ 메서드만 호출해야 하며, 예를 들어, GPU가 아닌 특성을 초기화해야 합니다. 반면 설치 프로그램의 그래픽 사용자 인터페이스가 초기화될 때 호출되는 initialize 메서드는 spoke의 전체 초기화를 완료해야 합니다.

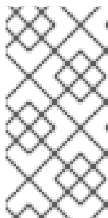
Hello World 애드온 예에서 다음과 같이 이 두 가지 방법을 정의합니다. __init__ 메서드에 전달된 인수의 수와 설명을 기록해 둡니다.

예 5.5. __init__ 및 초기화 방법 정의:

```
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")
```

__init__ 메서드에 전달된 data 매개 변수는 모든 데이터가 저장된 Kickstart 파일의 메모리 내 트리와 같습니다. RuntimeClass의 __init__ 메서드 중 하나에서 이 메서드는 해당 구조를 읽고 수정할 수 있도록 하는 self.data 특성에 저장됩니다.



참고

스토리지 오브젝트는 RHEL9에서 더 이상 사용할 수 없습니다. 추가 기능이 스토리지 구성과 상호 작용해야 하는 경우 Storage DBus 모듈을 사용합니다.

HelloworldData 클래스는 Hello World 애드온 예제에 이미 정의되어 있기 때문에 이 애드온에 대한 subtree가 이미 self.data에 있습니다. 클래스의 루트는 self.data.addons.com_example_hello_world 로 사용할 수 있습니다.

RuntimeClass의 __init__ 가 수행하는 또 다른 작업은 대화 상자의 .glade 파일을 사용하여 GtkBuilder의 인스턴스를 초기화하고 self.builder 로 저장하는 것입니다. 초기화 방법은 이 방법을 사용하여 Kickstart 파일의 %addon 섹션에서 텍스트를 표시하고 수정하는 데 사용되는 GtkTextEntry 를 가져옵니다.

spoke이 생성될 때 __init__ 및 initialize 메서드는 모두 중요합니다. 그러나 스포크의 주요 역할은 대화된 값의 표시 및 집합을 변경하거나 검토하려는 사용자가 방문하는 것입니다. 이를 활성화하려면 다음 세 가지 다른 방법을 사용할 수 있습니다.

- 새로 고침 - 스포크를 방문할 때 호출됩니다. 이 방법은 spoke의 상태를 새로 고침하여 표시된 데이터가 내부 데이터 구조와 일치하는지 확인하기 위해 자체.data 구조에 저장된 현재 값이 표시되도록 합니다.

- apply - 맞춤이 남아 있고 UI 요소의 값을 다시 self.data 구조에 저장하는 데 사용될 때 호출됩니다.

- execute - 사용자가 스포크를 나가고 스포크의 새 상태에 따라 런타임 변경을 수행하는 데 사용되는 경우 호출됩니다.

*이 함수는 다음과 같은 방식으로 샘플 **Hello World** 애드온으로 구현됩니다.*

예 5.6. 새로 고침 정의, 적용 및 실행 방법

```
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
```

*여러 가지 추가 방법을 사용하여 **spoke**의 상태를 제어할 수 있습니다.*

- ***Ready** - 스포크를 방문할 준비가 되었는지 여부를 결정합니다. 값이 **"False"**이면 패키지 소스가 구성되기 전에 패키지 선택 대화 상자에 액세스할 수 없습니다.*

- 완료 - 대화가 완료되었는지 확인합니다.

- 필수 - 대화 상자(예: 설치 대상 대화)가 필수인지 여부를 결정합니다. 이 대화는 자동 파티셔닝을 사용하려는 경우에도 항상 방문해야 합니다.

이러한 모든 속성은 설치 프로세스의 현재 상태에 따라 동적으로 결정해야 합니다.

다음은 **Hello World** 애드온에서 이러한 메서드의 샘플 구현으로, **Helloworld Data** 클래스의 텍스트 특성에 특정 값을 설정해야 합니다.

예 **5.7.** 준비, 완료 및 필수 방법 정의

```
@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
```

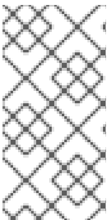
이러한 속성이 정의되면 스포크는 접근성과 완전성을 제어할 수 있지만, 내에 구성된 값의 요약을 제공할 수 없습니다. **spoke**를 방문하여 이 구성이 어떻게 구성되어 있는지 확인해야 합니다. 이는 바람직하지 않을 수 있습니다. 따라서 **status** 라는 추가 속성이 있습니다. 이 속성에는 구성된 값에 대한 간단한 요약이 포함된 한 줄의 텍스트가 포함되어 있으며, 이는 스포크 제목에 따라 허브에 표시할 수 있습니다.

status 속성은 다음과 같이 Hello World 예제 애드온에 정의되어 있습니다.

예 5.8. 상태 속성 정의

```
@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))
```

예제에 설명된 모든 속성을 정의하면 애드온에서 그래픽 사용자 인터페이스(GPU)와 Kickstart를 표시하는 완전 지원이 제공됩니다.



참고

여기에 설명 된 예제는 매우 간단 하 고 컨트롤을 포함 하지 않습니다. Python Gtk 프로그래밍에 대한 지식이 GUI에서 기능적인 대화식 스포크를 개발해야합니다.

한 가지 주목할 만한 제한 사항은 각 대화의 자체 기본 창 - SpokeWindow 위젯의 인스턴스가 있어야 한다는 것입니다. 이 위젯은 Anaconda에 관련된 다른 위젯과 함께 anaconda-widgets 패키지에 있습니다. anaconda-widgets-devel 패키지에서 GUI 지원과 같은 추가 기능 개발에 필요한 다른 파일을 찾을 수 있습니다.

그래픽 인터페이스 지원 모듈에 필요한 모든 방법이 포함되어 있으면 다음 섹션을 계속 사용하여 텍스트 기반 사용자 인터페이스에 대한 지원을 추가하거나 Anaconda 애드온 배포 및 테스트를 계속 수행하고 애드온을 테스트할 수 있습니다.

5.11. GUI 고급 기능

pyanaconda 패키지에는 허브 및 스포크에서 사용할 수 있는 여러 도우미 및 유틸리티 기능이 포함되어 있습니다. 대부분은 pyanaconda.ui.gui.utils 패키지에 있습니다.

샘플 Hello World add-on은 Anaconda 에서 사용하는 englightbox 콘텐츠 관리자의 사용법을 보여줍니다. 이 콘텐츠 관리자는 창을 Lightbox에 추가하여 가시성을 높이고 사용자가 기본 창과 상호 작용하는 것을 방지할 수 있습니다. 이 기능을 설명하기 위해 샘플 추가 기능에는 새 대화 상자 창을 여는 버튼이 포함되어 있습니다. 대화 상자 자체는 pyanaconda.ui.gui.init 에 정의된 GUIObject 클래스에서 상속되는 특수 HelloworldDialog입니다.

dialog 클래스는 mainWidgetName class 특성을 사용하여 채운 self.window 특성을 통해 액세스할 수 있는 내부 Gtk 대화 상자를 실행하고 삭제하는 run 메서드를 정의합니다. 따라서 대화 상자를 정의하는 코드는 다음 예와 같이 매우 간단합니다.

예 5.9. 엔트로피 대화 상자 정의

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

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

englightbox 대화 상자 예제 코드를 통해 대화 상자의 인스턴스를 만든 다음 enlightbox 컨텍스트 관리자를 사용하여 lightbox 내에서 대화 상자를 실행합니다. 컨텍스트 관리자는 대화 상자의 창에 대한 참조가 있으며 대화 상자의 Lightbox를 인스턴스화하기 위해 대화 상자만 있으면 됩니다.

Anaconda 에서 제공하는 또 다른 유용한 기능은 설치 중에 및 첫 번째 재부팅 후 표시되는 스포크를 정의하는 기능입니다. Initial Setup 유틸리티는 애드온 그래픽 사용자 인터페이스 (GUI)에 대한 지원 추가에서 설명됩니다. Anaconda 및 Initial Setup 모두에서 대화 상자를 사용하려면 pyanaconda.ui.common 모듈에 정의된 첫 번째 상속 클래스로 라고도 하는 특수한 FirstbootSpokeMixIn 클래스를 상속해야 합니다.

Anaconda 의 대화 상자와 Initial Setup의 재구성 모드에서 사용할 수 있도록 하려면 pyanaconda.ui.common 모듈에 정의된 첫 번째 상속된 클래스로 믹싱 이라고도 하는 특수한 FirstbootSpokeMixIn 클래스를 상속해야 합니다.

Initial Setup에서만 특정 대화 상자를 사용할 수 있도록 하려면 이 대화 상자에서는 FirstbootOnlySpokeMixIn 클래스를 상속해야 합니다.

대화 상자에서 항상 Anaconda 및 Initial Setup에서 사용할 수 있도록 하려면 다음 예제와 같이 spoke 에서 should_run 메서드를 재정의해야 합니다.

-

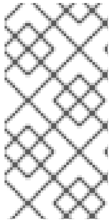
예 5.10. should_run 메서드 정의

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

pyanaconda 패키지는 @managek _action_wait 및 @Virtk_ action_nowait 데코레이터와 같은 고급 기능을 많이 제공하지만 이 가이드에서는 다루지 않습니다. 자세한 내용은 설치 프로그램의 소스를 참조하십시오.

5.12. TUI 애드온 기본 기능

Anaconda는 TUI(텍스트 기반 인터페이스)도 지원합니다. 이 인터페이스는 기능에서 더 제한되지만 일부 시스템에서는 대화식 설치를 위한 유일한 선택일 수 있습니다. 텍스트 기반 인터페이스와 그래픽 인터페이스 간의 차이점 및 TUI의 제한 사항에 대한 자세한 내용은 Anaconda 및 애드온 소개를 참조하십시오.



참고

텍스트 인터페이스에 대한 지원을 애드온에 추가하려면 Anaconda 애드온 구조에 설명된 tui 디렉터리에 새 하위 패키지 세트를 만듭니다.

설치 프로그램에서 텍스트 모드를 지원하는 것은 간단한 줄 라이브러리를 기반으로 하며, 이는 매우 간단한 사용자 상호 작용만 허용합니다. 텍스트 모드 인터페이스:

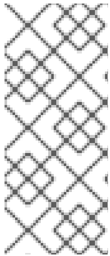
- 커서 이동을 지원하지 않습니다. 대신 라인 프린터처럼 작동합니다.
- 예를 들어 다양한 색상 또는 글꼴 사용과 같은 시각적 개선 사항은 지원하지 않습니다.

내부적으로 simpleline 툴킷에는 세 가지 주요 클래스가 있습니다. 앱,UIoctets 및 위젯. 위젯은 화면에 출력할 정보가 포함된 단위입니다. App 클래스의 단일 인스턴스에 의해 전환되는 UIsandboxs에 배치됩니다. 기본 요소인 hubs , spoke 's 및 'dialogs 에는 그래픽 인터페이스와 유사한 방식으로 다양한 위젯이 포함되어 있습니다.

애드온에서 가장 중요한 클래스는 NormalTUISpoke 이며 pyanaconda.ui.tui.spokes 패키지에 정의된 다른 다양한 클래스입니다. 이러한 모든 클래스는 TUIObject 클래스를 기반으로 하며, 이 클래스는 Add-on GUI 고급 기능에서 설명하는 GUI Object 클래스와 동일합니다. 각 TUI spoke는

NormalTUISpoke 클래스에서 상속하고 API에서 정의한 특수 인수와 메서드를 재정의하는 Python 클래스입니다. 텍스트 인터페이스는 GUI보다 단순하기 때문에 다음과 같은 두 가지 인수만 있습니다.

- 제목 - GUI의 제목 인수와 유사하게 대화 상자의 제목을 결정합니다.
- category - 문자열의 범주를 결정합니다. 카테고리 이름은 어디에서나 표시되지 않으며 그룹화에만 사용됩니다.



참고

TUI는 GUI와 다른 카테고리를 처리합니다. 기존 범주를 새 대화 상자에 할당하는 것이 좋습니다. 새 범주를 만들려면 Anaconda에 패치를 적용해야 하며 약간의 이점이 있습니다.

각 대화 상자에서 여러 가지 방법, 즉 init, initialize,refresh ,apply,execute,input, prompt ,prompt, and properties (ready,completed,mandatory, status)를 덮어씁니다.

추가 리소스

- Add-on GUI에 대한 지원 추가 를 참조하십시오.

5.13. 간단한 TUI SPOKE 정의

다음 예제에서는 Hello World 샘플 애드온에서 간단한 텍스트 사용자 인터페이스(TUI) 스포크의 구현을 보여줍니다.

사전 요구 사항

- Anaconda 애드온 구조에 설명된 대로 tui 디렉터리에 새 하위 패키지 세트를 생성했습니다.

절차

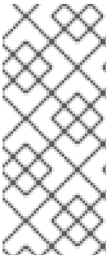
- 다음 예에 따라 추가 기능 텍스트 사용자 인터페이스(TUI)에 대한 지원을 추가하는 데 필요한 모든 정의로 모듈을 생성합니다.

예 5.11. 간단한 TUI Spoke 정의

```
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
```



참고

RuntimeClass의 init 만 호출할 경우에만 init 메서드를 재정의할 필요는 없지만 예제에 있는 주석은 스포크 클래스의 생성자에 전달된 인수를 이해할 수 있는 방식으로 설명합니다.

이전 예에서 다음을 수행합니다.

- 설정 방법은 모든 항목에 대해 스포크의 내부 특성에 대한 기본값을 설정합니다. 그러면 **refresh** 메서드에서 표시하고 입력 방법으로 업데이트하고 **apply** 방법으로 내부 데이터 구조를 업데이트하는 데 사용합니다.

- **execute** 방법은 GUI의 동등한 방법과 동일한 목적을 가지고 있습니다; 이 경우 방법은 적용되지 않습니다.

- 입력 방법은 텍스트 인터페이스에 고유합니다. Kickstart 또는 GUI에는 동등하지 않습니다. 입력 방법은 사용자 상호 작용을 담당합니다.

- 입력 메서드는 입력된 문자열을 처리하고 유형 및 값에 따라 조치를 취합니다. 위의 예에서는 값을 요청한 다음 내부 속성(키)으로 저장합니다. 더 복잡한 추가 기능에서는 일반적으로 구문 분석 문자와 같은 일부 비추적 작업을 수행하고, 숫자를 정수로 변환하고, 추가 화면을 표시하거나, 부울 값을 토글링해야 합니다.

- 입력 클래스의 반환 값은 **InputState Track** 또는 입력 문자열 자체여야 하며, 이 입력을 다른 화면에서 처리해야 합니다. 그래픽 모드와 달리 **apply** 및 **execute** 메서드는 대화된 상태로 둘 때 자동으로 호출되지 않습니다. 입력 방법에서 명시적으로 호출해야 합니다. 스포크의 화면을 닫기(hiding)하는 데 동일한 적용: 닫기 방법에서 명시적으로 호출해야 합니다.

다른 화면을 표시하려면 예를 들어 다른 대화 상자에 입력된 추가 정보가 필요한 경우 다른 TUIObject 를 인스턴스화하고 이를 표시하려면 ScreenHandler.push_screen_modal() 을 사용할 수 있습니다.

텍스트 기반 인터페이스의 제한으로 인해 TUI spokes는 확인란 목록 또는 선택 해제로 구성되는 매우 유사한 구조를 갖는 경향이 있습니다.

5.14. NORMALTUISPOKE를 사용하여 텍스트 인터페이스 SPOKE 정의

간단한 TUI Spoke 예제를 정의 하면 메서드가 사용 가능하고 제공된 데이터를 인쇄하고 처리하는 TUI spoke를 구현할 수 있었습니다. 그러나 pyanaconda.ui.tui.spokes 패키지의 Normal EditTUISpoke 클래스를 사용하여 이를 수행할 수 있는 다른 방법이 있습니다. 이 클래스를 상속하면 해당 클래스에서 설정해야 하는 필드 및 속성만 지정하여 일반적인 TUI 대화 상자를 구현할 수 있습니다.By inheriting this class, you can implement a typical TUI spoke by only specifying fields and attributes that should be set in it. 다음 예제에서는이를 보여 줍니다.The following example demonstrates this:

사전 요구 사항

- Anaconda 애드온 구조에 설명된 대로 TUI 디렉터리에 새 하위 패키지 세트를 추가했습니다.

절차

- 다음 예에 따라 추가 기능 텍스트 사용자 인터페이스(TUI)에 대한 지원을 추가하는 데 필요한 모든 정의로 모듈을 생성합니다.

예 5.12. NormalTUISpoke를 사용하여 텍스트 인터페이스 Spoke 정의

```
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. ANACONDA 애드온 배포 및 테스트

*설치 환경에 자신의 **Anaconda** 애드온을 배포하고 테스트할 수 있습니다. 이렇게 하려면 다음 단계를 수행합니다.*

사전 요구 사항

- *애드온을 생성했습니다.*
- ***D-Bus** 파일에 액세스할 수 있습니다.*

절차

1. *선호하는 위치에 **DIR** 디렉토리를 생성합니다.*
2. *애드온 **python** 파일을 **DIR/usr/share/anaconda/addons/**.에 추가합니다.*
3. ***D-Bus** 서비스 파일을 **DIR/usr/share/anaconda/dbus/services/**.에 복사합니다.*
4. ***D-Bus** 서비스 구성 파일을 **/usr/share/anaconda/dbus/confs/**.에 복사합니다.*
5. *업데이트 이미지를 생성합니다.*

   ***DIR** 디렉토리에 액세스합니다.*

   ```
   cd DIR
   ```

   업데이트 이미지를 찾습니다.

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

6. ***ISO** 부팅 이미지의 콘텐츠를 추출합니다.*

7. 결과 업데이트 이미지를 사용합니다.

   a. 업데이트.img 파일을 압축 해제된 ISO 콘텐츠의 images 디렉터리에 추가합니다.

   b. 이미지를 다시 끕니다.

   c. HTTP를 통해 Anaconda 설치 프로그램에 업데이트.img 파일을 제공하도록 웹 서버를 설정합니다.

   d. 부팅 옵션에 다음 사양을 추가하여 부팅 시 .img 파일을 업데이트 합니다.

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

기존 부팅 이미지의 압축을 풀고 product.img 파일을 만들고 이미지를 다시 패키징 하는 방법에 대한 자세한 내용은 Red Hat Enterprise Linux 부팅 이미지 추출을 참조하십시오.

---

[1] 애드온에서 새 범주를 정의해야 하는 경우 gui 패키지에 카테고리 하위 패키지가 포함될 수 있지만 권장되지는 않습니다.

6장. 사용자 지정 후 작업 완료

사용자 지정을 완료하려면 다음 작업을 수행합니다.

- product.img 이미지 파일을 만듭니다(그래픽 사용자 지정에만 적용).
- 사용자 지정 부팅 이미지를 생성합니다.

이 섹션에서는 product.img 이미지 파일을 생성하고 사용자 지정 부팅 이미지를 생성하는 방법에 대한 정보를 제공합니다.

6.1. PRODUCT.IMG 파일 생성

product.img 이미지 파일은 런타임 시 기존 설치 프로그램 파일을 대체하는 새 설치 프로그램이 포함된 아카이브입니다.

시스템 부팅 중에 Anaconda 는 부팅 미디어의 images/ 디렉터리에서 product.img 파일을 로드합니다. 그런 다음 이 디렉터리에 있는 파일을 사용하여 설치 프로그램의 파일 시스템에서 동일하게 이름이 지정된 파일을 대체합니다. 교체 시 파일은 설치 프로그램을 사용자 정의합니다(예: 기본 이미지를 사용자 지정으로 교체).

참고: product.img 이미지에는 설치 프로그램과 동일한 디렉터리 구조가 포함되어야 합니다. 설치 프로그램 디렉터리 구조에 대한 자세한 내용은 아래 표를 참조하십시오.

표 6.1. 설치 프로그램 디렉터리 구조 및 사용자 지정 콘텐츠

사용자 정의 콘텐츠의 유형	파일 시스템 위치
pixmaps (logo, 사이드바, 상단 표시줄 등)	**/usr/share/anaconda/pixmaps/**
GUI 스타일시트	**/usr/share/anaconda/anaconda-gtk.css**
Anaconda 애드온	**/usr/share/anaconda/addons/**
제품 설정 파일	**/etc/anaconda/product.d/**
사용자 정의 설정 파일	**/etc/anaconda/conf.d/**

사용자 정의 콘텐츠의 유형	파일 시스템 위치
Anaconda DBus 서비스 conf 파일	**/usr/share/anaconda/dbus/confs/**
Anaconda DBus 서비스 파일	**/usr/share/anaconda/dbus/services/**

*아래 절차에서는 **product.img** 파일을 만드는 방법을 설명합니다.*

절차

1. */tmp 와 같은 작업 디렉터리로 이동하여 product/:라는 하위 디렉터리를 만듭니다.*

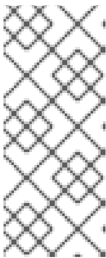
```
$ cd /tmp
```

2. *하위 디렉터리 제품/*

```
$ mkdir product/
```

3. *교체할 파일의 위치와 동일한 디렉터리 구조를 생성합니다. 예를 들어 설치 시스템의 **/usr/share/anaconda/addons** 디렉터리에 있는 애드온을 테스트하려면 작업 디렉터리에 동일한 구조를 생성합니다.*

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



참고

*설치 프로그램의 런타임 파일을 보려면 설치를 부팅하고 가상 콘솔 **1**(커널**Alt+F1**)으로 전환한 다음 두 번째 **tmux** 창(**Ctrl+b2**)으로 전환합니다. 파일 시스템을 검색하는 데 사용할 수 있는 쉘 프롬프트가 열립니다.*

4. *사용자 지정 파일(이 예에서는 **Anaconda**에 대한 사용자 정의 애드온)을 새로 생성된 디렉터리에 배치합니다.*

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

5. *설치 프로그램에 추가하려는 모든 파일에 대해 **3**단계와 **4**단계를 반복하고 디렉터리 구조를 만들고 사용자 지정 파일을 여기에 배치합니다.*

6. 디렉터리의 루트에 .buildstamp 파일을 생성합니다. .buildstamp 파일은 시스템 버전, 제품 및 기타 여러 매개변수를 설명합니다. 다음은 Red Hat Enterprise Linux 8.4의 .buildstamp 파일의 예입니다.

```
[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
```

IsFinal 매개변수는 이미지가 제품의 릴리스 (GA) 버전인지 (True), 또는 Alpha, Beta 또는 내부 이정표 (False)와 같은 사전 릴리스를 지정합니다.

7. product/ 디렉터리로 이동하여 product.img 아카이브를 생성합니다.

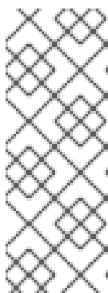
```
$ cd product
```

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

이렇게 하면 product/ 디렉터리 위에 product.img 파일이 생성됩니다.

8. product.img 파일을 추출된 ISO 이미지의 images/ 디렉터리로 이동합니다.

이제 product.img 파일이 생성되고 만들 사용자 지정이 해당 디렉터리에 배치됩니다.



참고

부팅 미디어에 product.img 파일을 추가하는 대신 이 파일을 다른 위치에 배치하고 부팅 메뉴에서 inst.updates= 부팅 옵션을 사용하여 로드할 수 있습니다. 이 경우 이미지 파일은 어떤 이름을 가질 수 있으며, 설치 시스템에서 이 위치에 도달할 수 있는 한 모든 위치(USB플러그 드라이브, 하드 디스크, HTTP, FTP 또는 NFS 서버)에 배치할 수 있습니다.

6.2. 사용자 정의 부팅 이미지 생성

부팅 이미지와 GUI 레이아웃을 사용자 지정한 후 변경한 내용이 포함된 새 이미지를 만듭니다.

사용자 지정 부팅 이미지를 생성하려면 아래 절차를 따르십시오.

절차

1. 변경 사항이 모두 작업 디렉터리에 포함되어 있는지 확인합니다. 예를 들어 애드온을 테스트하는 경우 **image /** 디렉터리에 **product.img** 를 배치해야 합니다.

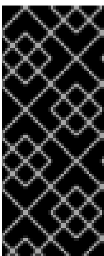
2. 현재 작업 디렉터리가 추출된 ISO 이미지의 최상위 디렉터리인지 확인합니다(예: **/tmp/ISO/iso/**).

3. **genisoimage** 를 사용하여 새 ISO 이미지를 생성합니다.

```
# genisoimage -U -r -v -T -J -joliet-long -V "RHEL-9 Server.x86_64" -volset "RHEL-9
Server.x86_64" -A "RHEL-9 Server.x86_64" -b isolinux/isolinux.bin -c isolinux/boot.cat
-no-emul-boot -boot-load-size 4 -boot-info-table -eltorito-alt-boot -e images/efiboot.img
-no-emul-boot -o ../NEWISO.iso .
```

위 예에서는 다음을 수행합니다.

- 동일한 디스크에 파일을 로드해야 하는 옵션에 대해 **LABEL=** 지시문을 사용하는 경우 **-V**,**-volset**, 및 **-A** 옵션의 값이 이미지의 부트 로더 구성과 일치하는지 확인합니다. BIOS 용**isolinux/isolinux.cfg** 및 UEFI의 경우 **EFI/BOOT/grub.cfg** 에서 **inst.stage2=LABEL=disk_label** 스탠자를 사용하여 동일한 디스크에서 설치 프로그램의 두 번째 단계를 로드하면 디스크 레이블이 일치해야 합니다.



중요

부트 로더 구성 파일에서 디스크 레이블의 모든 공간을 **\x20** 으로 바꿉니다. 예를 들어 RHEL 9.0 레이블이 있는 ISO 이미지를 생성하는 경우 부트 로더 구성에서 **RHEL\x209.0** 을 사용해야 합니다.

- **-o** 옵션 값(**-o ../NEWISO.iso**)을 새 이미지의 파일 이름으로 바꿉니다. 예제의 값은 현재 디렉터리의 디렉터리에 **NEWISO.iso** 파일을 생성합니다.

이 명령에 대한 자세한 내용은 **genisoimage(1)** 매뉴얼 페이지를 참조하십시오.

4. 이미지에 MD5 체크섬을 주입합니다. MD5 검사를 수행하지 않으면 이미지 확인 검사 (부트

로더 구성의 rd.live.check 옵션)가 실패하고 설치가 중단될 수 있습니다.

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

위의 예에서 ../NEWISO.iso 를 파일 이름 및 이전 단계에서 만든 ISO 이미지의 위치로 바꿉니다.

이제 새 ISO 이미지를 물리적 미디어 또는 네트워크 서버에 작성하여 실제 하드웨어에서 부팅하거나 가상 머신 설치를 시작할 수 있습니다.

추가 리소스

- 부팅 미디어 또는 네트워크 서버 준비 방법에 대한 자세한 내용은 고급 RHEL 9 설치 수행 을참조하십시오.
- ISO 이미지를 사용하여 가상 머신 생성에 대한 지침은 가상화 구성 및 관리를 참조하십시오.