WNDRVR
SUPPORT NETWORK

WIND RIVER LINUX DISTRO DEVELOPER'S GUIDE, LTS 22 (日本語版)

著作権について

本社

Wind River
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.
Toll free (U.S.A.): +1-800-545-WIND
Telephone: +1-510-748-4100

その他の連絡先については、下記のウェブサイトをご覧ください。
http://www.windriver.com

カスタマーサポートへのお問い合わせ
www.windriver.com/support

※ 本ドキュメントは、参照目的のために英語版「Wind River Linux Distro Developer's Guide, LTS 22」を翻訳したものです。

Wind River Linux Distro Developer's Guide, LTS 22
2023年5月2日

1. Wind River Linux Distro Developer's Guideの概要

このクイックスタートの説明を参考にして、Wind River® Linux Distro (バイナリ・リリース)をご利用ください。

Wind River Linux Distroバイナリ・リリースは、開発期間の短縮を支援するため、すぐに導入できる完全なターゲットシステムまたはコンテナイメージを提供します。これにより、開発期間を短縮することができます。また、独自のリポジトリを管理し、ウインドリバーのエコシステムの外部でイメージの更新や変更を行うことができるため、ニーズに合わせてイメージを調整することもできます。開発環境でのバイナリ・リリースをより良く利用するために、ウインドリバーはWind River Linux Assembly Toolを提供しています。

このツールを使用して、RPM パッケージのビルドと公開、特定のハードウェア用のパッケージフィードからのイメージ作成、更新された SDK の作成、および SDK ビルドの正常性の検証など、イメージの管理に役立つ多くのタスクを実行することができます。さらに、パッケージの追加や削除、ビルドの前後（do_build）の指示も指定できるため、ニーズに合わせてイメージをカスタマイズすることができます。

このツールは、***.yaml** 入力ファイルを使用して、ビルド、パッケージ、およびパッケージ関連の設定の多くの側面を定義し、さまざまな用途に使用できます。意図したプロジェクトの出力を定義する ***.yaml** 入力ファイルの検証を行い、不正確なオプションやスペルミスを特定できるようにします。サンプルの ***.yaml** ファイルのリストについては、「Linux Assembly Tool Core input.yaml サンプルファイル (48ページ)」を参照してください。

exampleyamls位置引数を使用すると、ビルドをカスタマイズするための基礎として使用できる、さまざまなサンプル***.yaml**ファイルのリストを作成することができます。これらの再利用可能なファイルは、すべてのイメージのカスタマイズのためのデータリファレンスを提供します。詳細については、「input.yamlサンプルファイルを作成する (31ページ)」を参照してください。

バイナリ・リリースSDKに含まれるLinux Assembly Toolは、Wind River Linuxプラットフォームプロジェクトのビルド環境のLinuxホストシステムで使用することができます。詳細については、「Linux Assembly Toolのインストールと実行(17ページ)」を参照してください。

必要条件

- Wind River Linux Distro を使用するための要件を満たすLinuxホスト ではGitバージョン1.9以上とPython 3を使用します。さらに、Python 3にまだ移行していない特定の依存関係を満たすために、Python 2も必要です。 詳細については、Wind River Linux Release Notes: Host System Recommendations and Requirementsをご参照ください。
- イメージのダウンロードや、システムのアップデートを行うためのインターネットアクセスが必要です。
- Linuxおよびコマンドラインに関する中級レベル知識。これには、コマンドの実行、パッケージのインストール、プロジェクトの設定ファイルの編集などが含まれます。

ワークフロー

Wind River Linux DistroのDistroイメージを立ち上げるためのワークフローは、以下の通りです。

a. Wind River Linux ビルドからパッケージフィードを作成します。詳細については、「ソースからビルドするバイナリベースコンテナ、SDK、パッケージフィード（20ページ） 」を参照してください。

b. Linux Assembly Tool（appsdk）をインストールし、実行します。詳しくは、「Linux Assembly Toolのインストールと実行 （17ページ） 」を参照してください。

c. 設定リファレンスとして使用するためのサンプル **input.yaml** ファイルを作成します。コンテナまたはシステムファイルシステムをカスタマイズする予定がある場合、***.yaml** ファイルは、カスタマイズを維持するための簡単なリファレンスを提供します。詳細については、「input.yamlサンプルファイルを作成する (31ページ)」を参照してください。

d. appsdk ツールを使用して、パッケージフィードの更新を実行し、展開されたターゲットデバイスで使用できるようにします。詳細については、「 Linux Assembly Toolを使用してソースパッケージを追加する（54 ページ）」を参照してください。

e. ユーザーとネットワークのカスタマイズ、コンテナの構成と更新、その他のカスタマイズを含むシステムを計画し、定義します。詳細については、「Linux Assembly Tool システム定義の概要（3ページ）」を参照してください。

注

このドキュメントでは、コマンドラインの例として、ARMベースのシステムにはbcm-2xxx-rpi4 BSPを、IAベースのシステムにはintel-x86-64 BSPを使用しています。お使いのBSPに合わせてコマンドを変更する必要がある場合があります。

2. システムイメージの定義

2.1. Linux Assembly Toolシステム定義の概要

まず、Linux Assembly Toolを使った開発プロセスの一部として、コンフィギュレーション変更がシステム全体の中でどのように位置づけられるかを理解しましょう。

システム定義の概要

システムの定義が重要な理由は、作成する全体的なシステムイメージは、単純なタスクを実行するために構成されたカーネルとユーザースペースパッケージのコンパイルによるものだからです。各システムはそれぞれ異なり、通常、各々の用途のために作成されます。一度配備されたシステムを、継続的な更新や構成の変更に適応しながら、その要件を満たし続けることができるのかは疑問です。

一般的な Linux システムでは、そのユースケースとシステム所有者に特化した、特定の資産のためにカスタマイズが必要です。以下はその例です。

- ホスト名やIPなどのシステム設定
- コンテナ、イメージ、コンフィギュレーション
- ユーザーアカウントと設定
- ベンダーデータ
- クラウドデータ

このタイプのアセットは、システムを定義するために必要な情報を提供します。この要件を満たすため、Linux Assembly Tool には **system *.yaml** オプションがあり、OS を構成する既存のツールと連携して動作するシステムの側面を定義することができます。これにより、イメージ作成時をはじめ、初回起動時、またはその後の起動時、そしてアップグレード後に異なるレベルのカスタマイズが可能になります。

システム *.yaml ファイル オプション

システムオプションは、appsdk genimageコマンドで作成された***.ustart**、***.vdi**、***.vmdk**、および***.wic**イメージにのみ適用されます。initramfsやコンテナイメージには対応していません。

下は、***.yaml**ファイルにおける**system**オプションの例です。

```
system:
  - run_once:
    - $PWD/run_once.d/10_add_system_user.sh
    - $PWD/run_once.d/20_add_user_home.sh
    - $PWD/run_once.d/30_set_hostname.sh
  - run_on_upgrade:
    - $PWD/run_on_upgrade.d/10_update_containers.sh
  - run_always:
    - $PWD/run_always.d/10_start_containers.sh
  - files:
    - file:
        src: $PWD/files/hello-world.tar.bz2
        dst: /var/containers/hello-world.tar.bz2
        mode: 600
    - file:
        src: $PWD/files/dnsmasq.conf
        dst: /etc/dnsmasq.d/shared.conf
        mode: 600
```

```
- contains:
  - $PWD/guests/guest1.yaml
```

注　　***.yaml** ファイル内の**システム**オプションの他の例については、「Linux Assembly Tool システム定義 input.yaml サンプルファイル （10ページ）」を参照してください。

この例では、以下の**システム**オプションによるカスタマイズ可能なタグが含まれています。

run_once

このタグは、初回起動時に一度だけ実行されるスクリプトファイルソースをリストアップします。Linux Assembly Tool は、リストされた各スクリプトファイルソースをシステムイメージ **/etc** ディレクトリ内の場所にコピーし、システム定義ツールがインストール後に一度だけ実行するようにします。この例では、**/etc/sysdef/run_once.d/10_add_system_user.sh** スクリプトが実行されてシステムにアクセスするためのデフォルトのユーザー名とパスワードを設定し、**/etc/sysdef/run_once.d/20_add_user_home.sh** スクリプトが実行されてユーザーのホームディレクトリを作り、**/etc/sysdef/run_once.d/30_set_hostname.sh** スクリプトが実行されてシステムイメージのホストネームを設定するように なっています。スクリプト名には、実行する順番を定義するために意図的に番号が付けられていることに注意してください。これは、追加するスクリプトの実行順序を指定し、依存関係の問題がないことを確認するために役立ちます。

run_on_upgrade

このタグは、最初のインストール後とアップグレード後にそれぞれ一度ずつ実行する必要がある各スクリプトのスクリプトファイルのソースをリストアップします。スクリプトの更新を追跡するために、Linux Assembly Tool は **/etc/sysdef/run_on_upgrade.d** の日付と時間に基づいて新しいサブディレクトリを作成し、そのスクリプトにはそのアップグレードバンドルに含まれているスクリプトのコピーが含まれます。例の中では、**/etc/sysdef/run_on_upgrade.d/10_update_containers.sh** スクリプトが、システムがアップグレードされるたびに、コンテナの更新をチェックするために実行されます。

run_always

このタグは、システムが起動するたびに実行される必要がある各スクリプトのスクリプトファイルのソースを一覧表示します。Linux Assembly Tool は、各スクリプトを **/etc/sysdef** ディレクトリにコピーし、システム定義ツールはシステムが起動または再起動するたびに各スクリプトを実行します。この例では、**/etc/sysdef/run_always.d/10_start_containers** スクリプトが実行されて、システム起動時に任意のコンテナが自動的に開始されます。

files

このタグは、システムイメージの**ファイル**エレメントをリストアップします。**各ファイル**エレメントには、イメージに含まれる各ファイルのソース（**src**）、デスティネーション（**dst**）、**モードが**定義されています。

contains

このタグは、システムの ***.yaml** ファイルのリストを指定し、他のシステム指定ファイルのインクルードとして機能し、インクルードされたシステムの限定的なネストを定義します。例えば、このタグを使用することにより、Xen のトップレベルシステムを含まれる VM と共に定義できます。Linux Assembly Tool は 1 レベルのネストしかサポートしないため、**contains** タグによって参照される ***.yaml** ファイルは、それ自身に **contains** タグを含めることができません。

contains タグは、***.ustart**、***.vdi**、および ***.vmdk** イメージタイプのみをサポートします。Linux Assembly Tool は、コンテナイメージ ***.yaml** ファイルから**contains** タグの参照が識別されると、イメージを作成しようとするときに失敗してエラーが出力されます。

Linux Assembly Toolは、システム定義のプロセスを簡素化するために、***.yaml**ファイルの例と定義済みスクリプトのリストを提供します。ウインドリバーは、これら

のスクリプトを提供して、お客様の迅速な立ち上げと実行を支援します。

例えば、前述した **run_once.d/10_add_system_user.sh** スクリプトは、**system-user** という名前のユーザーを作成します。このスクリプトを修正して、プラットフォーム・プロジェクト・イメージに必要な任意のユーザー名を設定し、さらにパスワードも指定することができます。詳細については、「Linux Assembly Tool システム定義 input.yaml サンプルファイル（10 ページ）」を参照してください。

システム定義のサンプル ***.yaml** ファイルを表示するには、Linux Assembly Tool を使用してファイルを作成する必要があります。詳細については、「input.yamlサンプルファイルを作成する (31ページ)」を参照してください。このファイルを取得すると、たとえば **exampleyamls/ sysdef** ディレクトリにあるファイルを表示できます。

```
$ tree exampleyamls/sysdef/
exampleyamls/sysdef/
├── add-system-user.yaml
├── add-user-home.yaml
├── contains-container-
base.yaml
├── files
│ ├── dns.conf
│ └── sudoers_sudo
├── run_always.d
│ └── 10_start_containers.sh
├── run_once.d
│ ├── 10_add_system_user.sh
│ ├── 20_add_user_home.sh
│ ├── 30_set_hostname.sh
│ └── 40_set_ntp.sh
├── run_on_upgrade.d
│ ├── 10_update_containers.sh
│ └── containers.dat
├── set-dns.yaml
├── set-hostname.yaml
├── set-ntp.yaml
└── update-containers.yaml
```

sysdef.sh Runtime Tool

Linux Assembly Tool は、**システム**オプションで定義されたスクリプトがターゲットシステム上で確実に実行されるように、システムイメージに sysdef.sh ランタイムツールを含んでいます。ブートプロセス中の適切なタイミングで sysdef.sh ツールが自動的に実行されるように、Linux Assembly Tool で作成されたシステムイメージは systemd **sysdef.service** サービスを含んでいます。このサービスが失敗した場合、エラー出力前に、最大3回まで再実行されます。

本ツールが実行した各スクリプトは、**/var/log/syslogに**取り込まれ、ターゲットシステム上にて手動で閲覧することにより、スクリプトの状態や実行時刻を確認することができます。スクリプトが失敗すると、残りのスクリプトは、/var/log/syslogの各スクリプトが実行されるまで実行し続けます。
シーケンス **/etc/sysdef/run_once.d**, **/etc/sysdef/run_on_upgrade.d**, **/etc/sysdef/run_always.d が**実行されます。スクリプトは英数字順に実行されるため、実行順序を指定しても、それに応じてスクリプト名を設定することができます。

このツールは、日付スタンプファイルを利用して、一度だけ実行するように設計されたスクリプトが複数回実行されることを防止します。この機能により、スクリプトの誤実行を防ぎ、システムの整合性を確保することができます。**-f**オプションを使用することで、日付スタンプに関係なくスクリプトを実行することができます。

ターゲットシステムでのsysdef.shツールの使用方法については、「sysdef.shランタイムツールの使用方法（9ページ）」を参照してください。

2.2. ロングライフDockerコンテナへの対応の概要

Linux Assembly Toolには、システムイメージにパーマント（長寿命）な追加物としてコンテナを追加するためのオプションが多数用意されています。

概要

Linux Assembly Toolを使用すると、システムイメージ内のコンテナを操作・管理するために、以下のすべてのアクションを実行することができます。

- docker loadコマンドでDockerイメージを読み込む
- docker importコマンドでファイルシステム*.tarファイルをインポートする
- コンテナレジストリからDockerイメージを取り込む
- 1つのイメージから複数のコンテナを実行
- ユーザー定義のdocker runオプションでコンテナを実行する
- ユーザー定義のDockerコマンドによるコンテナの実行
- システム起動のたびにコンテナを自動起動
- システムシャットダウン時にコンテナを円滑に停止
- システムアップグレード時のコンテナ更新

インプリメンテーション

システムイメージにコンテナを実装するには、どのコンテナを追加するか、コンテナイメージをどこから取得するか、システムイメージのどこに配置するか、コンテナをいつ起動し更新するかなどを定義した *.yaml ファイルを使用します。サンプル　*.yamlファイルについては、「Linux Assembly Tool システム定義 input.yaml サンプルファイル（10 ページ）」を参照してください。

追加するコンテナを指定するために、*.yamlファイルのrootfs-post-scriptsタグを更新して、コンテナの取得と起動のためのコマンドを指定します。コンテナをどこから取得するかは、docker loadコマンドとdocker pullコマンドのどちらを使用してコンテナを取得するかによって決まります。docker load を使用してシステムイメージから実行する各コンテナには、コンテナの取得先、システム内のコピー先、およびファイルモードを定義する system: file タグ内のエントリも必要です。これは、docker pullコマンドを使用してレジストリから直接取得したコンテナには必要ありません。

システムイメージを作成するとき、rootfs-post-scripts タグ内のコンテナエントリが container.dat ファイルを更新するのに使用されます。

```
ubuntu
  ->        docker pull ubuntu
  ->        docker run --name ubuntu ubuntu
  ->        systemctl start start-container@ubuntu.service

ubuntu-bash image=ubuntu run-opt="-p 2224:22 -it" run-cmd=/bin/bash
  ->        docker pull ubuntu
  ->        docker run -p 2224:22 -it -d --name ubuntu-bash ubuntu /bin/bash
  ->        systemctl start start-container@ubuntu-bash.service

wrlinux-image-full import=/var/docker-images/wrlinux-image-full-intel-x86-64.tar.bz2 run-cmd=/bin/
sh
  ->        docker import /var/docker-images/wrlinux-image-full-intel-x86-64.tar.bz2 wrlinux-image
-full
  ->        docker run -itd --name wrlinux-image-full wrlinux-image-full /bin/sh
  ->        systemctl start start-container@wrlinux-image-full.service

container-base  load=/var/docker-images/container-base-intel-x86-64.docker-image.tar.bz2
  ->        docker load -i /var/docker-images/container-base-intel-x86-64.docker-image.tar.bz2
  ->        docker run -itd --name container-base
  ->        systemctl start start-container@container-base.service
```

container.dat ファイルでは、container-name は必須です。これは、例えば、コンテナに対するアクションを実行するためのDockerコマンドで使用されるコンテナの名前を表します。

$ **docker run --name** container-name options

各セクションはコンテナ名から始まり、コンテナの取得、ロード、起動に必要なコマンドを含んでいます。

rootfs-post-scripts タグでコンテナを定義したら、次のステップでは **system: run_always** と **system: run_on_upgrade** タグを更新することになります。これらのタグは、システムの起動や再起動のたびにコンテナが自動的に実行されるようにし、また、システムの更新が行われるたびにコンテナの更新がチェックされるようにするものです。Linux Assembly Tool は、この目的のために、次の場所にある 2 つのサンプルスクリプトを提供します。

- **exampleyamls/sysdef/run_on_upgrade.d/10_update_containers.sh**
- **exampleyamls/sysdef/run_always.d/10_start_containers.sh**

ワークフロー

a. システムコンテナの設定に合わせて更新するサンプル ***.yaml** ファイルと、コンテナの機能を制御するのに役立つスクリプトを作成します。

詳細については、「input.yamlサンプルファイルを作成する（31ページ）」を参照してください。

b. Linuxホストシステムまたはアクセス可能なWebサーバーで、必要なコンテナを利用できるようにします。

これは、docker loadコマンドを使用してシステムイメージから直接ロードするコンテナに対してのみ必要です。docker pull コマンドを使用してレジストリからコンテナを取得する予定の場合は、このステップを無視できます。

c. コンテナの ***.yaml** ファイルを更新して、システムのコンテナ構成と一致させます。

次の例では、システムイメージに追加する3つのコンテナを指定し、コンテナファイルの場所とイメージへの追加方法を異なる例にしています

```
image_type:
- ostree-repo
- ustart
packages:
- startup-container
- docker
rootfs-post-scripts:
- echo "contains-base load=/var/docker-images/container-base-intel-x86-64.docker-image.tar.bz2
image=container-base-intel-x86-64" >> $IMAGE_ROOTFS/etc/sysdef/run_on_upgrade.d/containers.dat
- echo "ubuntu-tar load=/var/docker-images/ubuntu.docker-image.tar.bz2" >> $IMAGE_ROOTFS/etc/sy
sdef/run_on_upgrade.d/containers.dat
- skopeo copy --src-tls-verify=false --insecure-policy docker://path_to_repo /ubuntu docker-
archiv
e:$IMAGE_ROOTFS/var/docker-images/ubuntu.docker-image.tar.bz2:ubuntu-tar
system:
- run_on_upgrade:
  - exampleyamls/sysdef/run_on_upgrade.d/10_update_containers.sh
- run_always:
  - exampleyamls/sysdef/run_always.d/10_start_containers.sh
- contains:
  - exampleyamls/container-base-intel-x86-64.yaml
- files:
 - file:
   src: deploy/container-base-intel-x86-64.docker-image.tar.bz2
   dst: /var/docker-images/
```

この例では、イメージに含まれるコンテナを定義する3つの主要な***.yaml**タグがあります。

image-type

このタグは、コンテナが常駐するイメージタイプを識別します。このオプションは、イメージ作成時に Linux Assembly Tool によってコンテナファイルが正しく処理されるように、**ustart** または **wic** に設定する必要があります。

packages

このタグは、コンテナの追加をサポートするためにインストールするパッケージを指定します。これは、必要な **startup-container** と**docker**パッケージの依存関係を確認します。

rootfs-post-scripts

このタグは、システムイメージにコンテナを追加するために実行するコマンドのリストを提供します。

- 最初のエントリでは、**gencontainer**位置引数で作成したDockerコンテナである**deploy/container-base-intel-x86-64.docker-image.tar.bz2**コンテナを使用しています。**system: file** タグでコンテナイメージをホスト上のシステムイメージの **/var/docker-images** ディレクトリにコピーし、 **rootfs-post- scripts** タグで **containers.dat** ファイルを操作してターゲット上でコンテナイメージをロードして起動します。
- 2番目のエントリでは、Dockerレジストリにあるサードパーティ製の**ubuntu.docker-image.tar.bz2**イメージを使用しています。**rootfs-post-scripts**タグを使用して**containers.datファイルを**操作し、ターゲット上でコンテナイメージをロードして起動します。
- 3番目のエントリでは、Wind River Linux Binary release ウェブサイトにある **wrlinux-image-full-intel-x86-64.tar.bz2** ファイルシステムイメージを使用しています。これは、**system: file** タグを使用して、サーバからイメージを取得し、ホスト上のシステムイメージの **/var/docker-images** ディレクトリに保存し、次に **rootfs-post-scripts** タグを使用して containers.dat ファ

イルを操作し、ターゲット上でコンテナイメージをインポートして実行するものです。

- 各エントリと ***.yaml** ファイル全般で、**$IMAGE_ROOTFS** 変数はシステムイメージのルートファイルシステムディレクトリの場所を参照します。

rootfs-post-scripts タグは、コンテナとその入れ方を指定するだけでなく、コンテナ設定を **containers.dat** ファイルに、コンテナごとに 1 行ずつ追加します。

system: run_on_upgrade

このタグは、システムのアップグレードが行われるたびに **10_update_containers.sh** スクリプトを実行するように指定します。

system: run_always

このタグは、システム起動時に **rootfs-post-scripts** タグで指定されたコンテナが起動するように、システム起動時に **10_start_containers.sh** スクリプトを実行するように指定します。

system: file

これらのタグは、コンテナイメージの場所（**src**）、システム内のインストール先（**dst**）、パーミッションを設定するファイル**モード**などを指定するものです。

d. **contains-container-base.yaml**ファイルが完成したら、このファイルを使用して新しいシステムイメージを作成します。

e. コンテナを含むシステムイメージを展開し、コンテナの状態を確認します。

このプロセスは、追加の例とともに、「システムイメージにコンテナを追加する（57ページ）」で詳しく説明されています。

2.3. ランタイムツールsysdef.shの使用方法

sysdef.shツールは、ブートプロセス中にターゲット上で自動的に実行されるように設計されていますが、このツールを使用して情報を取得し、手動でsysdefスクリプトを実行することができます。

このタスクについて

sysdef.shツールに関する追加情報は、「Linux Assembly Tool システム定義の概要（3 ページ）」参照してください。

始める前に

Linux Assembly Toolで作成したデプロイ済みシステムイメージに、少なくとも1つのsysdefスクリプトを持つsystemオプションが含まれている必要があります。

手順

a. **sysdef.sh** のヘルプ出力を表示します。

```
# sysdef.sh -h
usage: sysdef.sh [-f] [-v] |run-once|run-on-upgrade|run-always [script1] [script2] [...]
    sysdef.sh [-f] [-v] run-all
    sysdef.sh [-v] list
       -f: ignore stamp, force to run
       -v: verbose
```

b. **sysdef.service**のステータスを確認します。

```
# systemctl status sysdef.service
* sysdef.service - A tool to implement the runtime functionality of the System Definition.
     Loaded: loaded (/usr/lib/systemd/system/sysdef.service; enabled; vendor preset: disabled)
     Active: active (exited) since Mon 2021-05-23 07:53:30 UTC; 1h 8min ago
    Process: 329 ExecStart=/usr/bin/sysdef.sh run-all (code=exited, status=0/SUCCESS)
   Main PID: 329 (code=exited, status=0/SUCCESS)
```

c. 特定のスクリプトを手動で再実行する。

```
$ sysdef.sh -f run-on-upgrade 10_update_containers.sh
Start run-on-upgrade(20210523074928) 10_update_containers.sh
...
Run run-on-upgrade(20210523074928) 10_update_containers.sh success
```

d. すべてのスクリプトを手動で再実行する。

```
$ sysdef.sh -f run-all
Start run-on-upgrade(20210523074928) 10_update_containers.sh
...
Run run-on-upgrade(20210523074928) 10_update_containers.sh success
```

e. sysdef.shのログを確認する。

```
$ grep sysdef /var/log/syslog
2021-05-23T07:52:54.216341+00:00 intel-x86-64 sysdef.sh[329]: Start run-once 10_add_system_user
.sh
2021-05-23T07:52:54.220970+00:00 intel-x86-64 sysdef.sh[329]: Run run-once 10_add_system_user.s
h success
2021-05-23T07:52:54.222146+00:00 intel-x86-64 sysdef.sh[329]: Start run-once 20_add_user_home.s
h
```

f. システムで実行するすべてのスクリプトをリストアップします。

```
$ sysdef.sh list
run-once
    10_add_system_user.sh
    10_add_system_user.sh.stamp
    20_add_user_home.sh
    20_add_user_home.sh.stamp
    30_set_hostname.sh
    30_set_hostname.sh.stamp
    40_set_ntp.sh
    40_set_ntp.sh.stamp
run-on-upgrade(20210523074928)
    10_update_containers.sh
    10_update_containers.sh.stamp
    containers.dat
run-always
    10_start_containers.sh
```

2.4. Linux Assembly Tool システム定義 input.yaml サンプルファイル

システム定義 **input.yaml** ファイルは、あなたがイメージを作成する多くの場面で、システム関連の情報を定義するための具体的な情報を提供します。

注 **input.yaml**のファイル名は参考例です。子が**.yaml**であれば、用途に応じたファイル名を付けることができます。

中核となる**input.yaml** ファイルの例については、「Linux Assembly Tool Core input.yaml サンプルファイル (48ページ)」を参照してください。

最初にSDKをインストールしてセットアップするとき、参照用の***.yaml**ファイルは含まれていません。これを作成するには、ソースとなるSDKの作業ディレクトリで次のコマンドを実行します。

```
$ appsdk exampleyamls
appsdk - INFO: Deploy Directory: /opt/windriver/wrlinux-graphics/20.38/exampleyamls
+-------------------+-----------------------------------------+
|     Yaml Type     |                  Name                   |
+===================+=========================================+
| Image             | container-base-intel-x86-64.yaml        |
|                   |  core-image-minimal-intel-x86-64.yaml   |
|                   | core-image-sato-intel-x86-64.yaml       |
|                   | initramfs-ostree-image-intel-x86-64.yaml |
|                   |  wrlinux-image-small-intel-x86-64.yaml  |
|                   |                                         |
+-------------------+-----------------------------------------+
| Feature           | feature/debug-tweaks.yaml               |
|                   | feature/package_management.yaml         |
|                   | feature/set_root_password.yaml          |
|                   | feature/startup-container.yaml          |
|                   | feature/vboxguestdrivers.yaml           |
|                   | feature/xfce_desktop.yaml               |
|                   |                                         |
+-------------------+-----------------------------------------+
| System Definition | sysdef/add-system-user.yaml             |
|  Yamls            | sysdef/add-user-home.yaml               |
|                   | sysdef/contains-container-base.yaml     |
|                   | sysdef/set-dns.yaml                     |
|                   | sysdef/set-hostname.yaml                |
|                   | sysdef/set-ntp.yaml                     |
|                   | sysdef/update-containers.yaml           |
|                   |                                         |
+-------------------+-----------------------------------------+
```

これらのサンプルファイルを元に、独自のファイルを作成してください。

注　このサンプルファイルは、あくまで参考用です。バイナリ開発は継続的に行われているため、このセクションには最新の更新が含まれていない可能性があります。常に最新の内容で作業できるように、上記のコマンドを実行して、最新のサンプル ***.yaml** ファイルを生成してください。

add-system-user.yamlサンプルファイル

次のサンプルファイルでは、パスワード付きで、ホームディレクトリを持たない新規ユーザーをシステムに追加するように指定しています。

```
# - Add a new user to the system, do not create home directory
# Username: system-user
# Password: 123456
image_type:
- ostree-repo
- ustart
packages:
- shadow
- base-passwd
- base-files
system:
- run_once:
  - exampleyamls/sysdef/run_once.d/10_add_system_user.sh
  _once.d/10_add_system_user.sh
```

add-user-home.yaml サンプルファイル

次のサンプルファイルでは、システムに追加する新しいユーザーとパスワード、ホームディレクトリを指定しています。

```
# - Add a new user and create user's home directory and
#   add the user to sudo group
# - Allow members of group sudo to execute any command
# Username: admin
# Password: 123456

image_type:
- ostree-repo
- ustart
packages:
- shadow
- base-passwd
- base-files
- sudo
system:
- run_once:
  - exampleyamls/sysdef/run_once.d/20_add_user_home.sh
- files:
  - file:
      src: exampleyamls/sysdef/files/sudoers_sudo
      dst: /etc/sudoers.d/sudo
      mode: 644
```

contains-ontainer-base サンプルファイル

以下のサンプルファイルでは、シェルスクリプトによるコンテナの起動とアップグレードのためのシステム固有のオプションとともに、デプロイするコンテナを指定しています。

```
image_type:
- ostree-repo
- ustart
packages:
- startup-container
- docker
rootfs-post-scripts:
- echo "contains-container-base load=/var/docker-images/container-base-intel-x86-64.docker-image.t
ar.bz2 image=container-base-intel-x86-64" >> $IMAGE_ROOTFS/etc/sysdef/run_on_upgrade.d/containers
.dat
system:
- run_on_upgrade:
  - exampleyamls/sysdef/run_on_upgrade.d/10_update_containers.sh
- run_always:
  - exampleyamls/sysdef/run_always.d/10_start_containers.sh
- contains:
  - exampleyamls/container-base-intel-x86-64.yaml
- files:
  - file:
       src: sub_deploy/deploy/container-base-intel-x86-64.docker-image.tar.bz2
       dst: /var/docker-images/
```

set-dns.yamlサンプルファイル

次のサンプルファイルは、イメージの DNS 設定を定義するために使用する ***.conf** ファイルを指定します。

```
# Set Wind River DNS
  age_type
- ostree-repo
- ustart
system:
- files:
  - file:
       src: exampleyamls/sysdef/files/dns.conf dst: /
       etc/systemd/resolved.conf.d/dns.conf
       mode: 644
```

set-hostname.yamlサンプルファイル

次のサンプルファイルは、MACアドレスまたは現在時刻をもとにシステムのホスト名の設定を指定します。

```
#!/bin/sh
# Set hostname based on MAC address or current time
  age_type
ostree-repo
ustart
packages:
sed
system:
- run_once:
- exampleyamls/sysdef/run_once.d/30_set_hostname.sh
```

set-ntp.yaml サンプルファイル

次のサンプルファイルは、システムネットワークタイムプロトコル（NTP）サーバーの設定を指定します。

```
# Set Wind River NTP
image_type:
- ostree-repo
- ustart
system:
- run_once:
  - exampleyamls/sysdef/run_once.d/40_set_ntp.sh
```

スタートアップコンテナファイルのサンプル

以下は、intel-x86-64 BSP 用の startup-container **input.yaml** ファイルの例です。この例では、システムイメージにコンテナを含め、その実行方法を指定するための 3 種類の方法を提供しています。

```
packages:
- startup-container
- docker
rootfs-post-scripts:
- echo "container-base load=/var/docker-images/container-base-intel-x86-64.docker-image.tar.bz2 im
age=container-base-intel-x86-64" >> $IMAGE_ROOTFS/etc/sysdef/run_on_upgrade.d/containers.dat
- echo "ubuntu-tar load=/var/docker-images/ubuntu.docker-image.tar.bz2" >> $IMAGE_ROOTFS/etc/sysde
f/run_on_upgrade.d/containers.dat
- echo "wrlinux-image-full import=/var/docker-images/target-full-intel-x86-64.tar.bz2 run-cmd=/bin
/sh" >> $IMAGE_ROOTFS/etc/sysdef/run_on_upgrade.d/containers.dat
- skopeo copy --src-tls-verify=false --insecure-policy docker://pek-lpdfs01:5000/ubuntu docker-arc
hive:$IMAGE_ROOTFS/var/docker-images/ubuntu.docker-image.tar.bz2:ubuntu-tar
system:
- run_on_upgrade:
  - exampleyamls/sysdef/run_on_upgrade.d/10_update_containers.sh
- run_always:
  - exampleyamls/sysdef/run_always.d/10_start_containers.sh
- files:
  - file:
      src: deploy/container-base-intel-x86-64.docker-image.tar.bz2
      dst: /var/docker-images/
      mode: 644
  - file:
      src: http://path_to_server /dist/wrlinux/lts-22/intel-x86-64/latest/dist/intel-x86-64/container-
full-intel-x86-64/wrlinux-image-full-intel-x86-64.tar.bz2
      dst: /var/docker-images/
      mode: 644
```

update-containers.yamlサンプルファイル

以下のサンプルファイルでは、起動およびアップグレードするコンテナを指定しています。この例では、**containers.dat** ファイルが更新するコンテナのリストを保持しています。

```
# - At a boot after upgrade, pulls listed containers (from containers.dat)
# from a public registry and runs them.
# - At each boot, start listed containers (from containers.dat)
# - Two containers in containers.dat: hello-world and ubuntu
# - Add a docker_daemon.jason to set private insecure registries of Wind River
image_type:
- ostree-repo
- ustart
packages:
- startup-container
- docker
rootfs-post-scripts:
- echo "ubuntu" >> $IMAGE_ROOTFS/etc/sysdef/run_on_upgrade.d/containers.dat
- echo "ubuntu-bash image=ubuntu run-opt="-p 2224:22 -it" run-cmd=/bin/bash" >> $IMAGE_ROOTFS/
etc/
sysdef/run_on_upgrade.d/containers.dat
system:
- run_on_upgrade:
- exampleyamls/sysdef/run_on_upgrade.d/10_update_containers.sh
- run_always:
- exampleyamls/sysdef/run_always.d/10_start_containers.sh
- files:
- file:
src: exampleyamls/sysdef/run_on_upgrade.d/containers.dat
dst: /etc/sysdef/run_on_upgrade.d/
mode: 644
```

2.5. パッケージのアップデートをシードするための開発用HTTPサーバーのセットアップ

SDKはビルドパッケージリポジトリとは別に使用するように設計されているため、SDKパッケージに追加や変更を加える場合、これらの更新を他のターゲットデバイスで利用できるようにする機能が必要です。

このタスクについて

これを実現するために、SDK がインストールされている Linux ホストシステム上に HTTP サービスを作成します。これにより、開発中に使用するパッケージリポジトリに簡単にアクセスでき、「Distroイメージによるベースコンテナ、SDK、パッケージフィードの構築（23 ページ）」で説明するWind River Linuxビルドシステムを必要としないため、開発テストとパッケージの更新が簡素化されます。

この例は開発目的だけのものであり、多数のデバイスに対応するように設計されていません。顧客向けのパッケージリポジトリには、この手順で使用した Python ベースのサーバーよりも堅牢な Linux ホスト（Web サーバー）を使用する必要があるかもしれませんが、これはこのドキュメントの範囲外です。

始める前に

Linux Assembly Toolを使用するためには、インターネットに接続可能なLinuxホストシステムが必要です。

手順

a. パッケージリポジトリ用のディレクトリを作成します。

```
$ mkdir -p /path-to /http_service_data/third_party_repo
```

b. サーバーを起動します。

```
$ python3 -m http.server 8888 --directory /path-to /http_service_data
```

サーバーが起動すると、ブラウザで http://host-IP-address **:8888/third_party_repo** にアクセスできるようになります。

3. WIND RIVER LINUX ASSEMBLY TOOLの使い方

3.1. Linux Assembly Toolのインストールと実行

Linux Assembly Toolは、バイナリ・リリースリポジトリを管理するための一般的なタスクを実行するのに役立ちます。

このタスクについて

これらのタスクには、RPM パッケージの構築と公開、特定のハードウェア用のパッケージ フィードからのイメージ作成、更新された SDK の作成、および SDK ビルドの正常性の確認が含まれます。Linux Assembly Tool は、x86-64 Linux ホストシステムでサポートされています。

始める前に

- 「ソースからビルドするバイナリベースコンテナ、SDK、パッケージフィード (20ページ)」および「Distroイメージによるベースコンテナ、SDK、パッケージフィードの構築（23 ページ）」で説明するように、パッケージリポジトリをセットアップしてビルドを完了させる必要があります。
- インターネットに接続する必要があります。

手順

a. SDK を Linux ホストシステムにインストールします。

SDKが存在するディレクトリ（通常はプラットフォームプロジェクトのビルドディレクトリである**tmp-glibc/deploy/sdk**に移動してください。
ディレクトリに移動し、シェルスクリプトを実行します。

```
$ ./wrlinux-*-container-base-sdk.sh -y -d sdkDir
```

この例のsdkDirはSDKをインストールする場所を表しています。

b. SDKを使用できるように環境を整えます。

SDKの作業ディレクトリsdkDir **/windriver/wrlinux-graphics/2X.XX**に移動し、環境設定スクリプトを実行します。

```
$ . ./environment-setup-*-wrs-linux
```

このスクリプトは、環境を設定し、**sysroots** サブディレクトリを作成します。これでアプリケーションSDK管理ツールを使用する準備が整いました。

c. Linux Assembly Toolを起動し、**-h**オプションで使用オプションを表示します。

> 注　Linux Assembly Toolは、ツールの位置引数およびサブコマンドのBashシェルタブ補完をサポートしています。appsdkと位置引数の先頭をタイプして**TAB**キーを押すだけで補完され、利用可能なオプションが表示されます。

```
$ appsdk -h
usage: appsdk [-h] [-d] [-q] [--log-dir LOGDIR]
              {gensdk,checksdk,genrpm,publishrpm,genimage,geninitramfs,gencontainer,genyaml,exa
mpleyamls}
              ...
```

```
Wind River Linux Assembly Tool

positional arguments:
  {gensdk,checksdk,genrpm,publishrpm,genimage,geninitramfs,gencontainer,genyaml,exampleyamls}
                        Subcommands. "appsdk <subcommand> --help" to get more info
    gensdk              Generate a new SDK
    checksdk            Sanity check for SDK
    genrpm              Build RPM package
    publishrpm          Publish RPM package
    genimage            Generate images from package feeds for specified machines
    geninitramfs        Generate Initramfs from package feeds for specified machines
    gencontainer        Generate Container Image from package feeds for specified
                        machines
    genyaml             Generate Yaml file from Input Yamls
    exampleyamls        Deploy Example Yaml files

optional arguments:
  -h, --help            show this help message and exit
  -d, --debug           Enable debug output
  -q, --quiet           Hide all output except error messages
  --log-dir LOGDIR      Specify dir to save debug messages as log.appsdk regardless of the logg
ing
                        level

Use appsdk <subcommand> --help to get help
```

このツールは、オプションの引数とともに、利用可能なアクションを定義するために位置引数を使用することに注意してください。位置引数に関する追加のヘルプを得るには、例えば**-h**オプションをつけてその引数を実行します。

```
$ appsdk gensdk -h
usage: appsdk gensdk [-h] -f FILE [-o OUTPUT]

optional arguments:
  -h, --help                show this help message and exit
  -f FILE, --file FILE      An input yaml file specifying image information.
  -o OUTPUT, --output OUTPUT
                            The path of the generated SDK. Default to deploy/AppSDK.sh in curr
ent
                            directory
```

利用可能なすべてのオプションの一覧は、「Linuxアセンブリツールオプションリファレンス（42 ページ）」を参照してください。

次のステップ

Linux Assembly Tool を起動すると、これを使用してバイナリ・リリースを管理することができます。ソースから新しいパッケージを追加して、外部のターゲットデバイスで利用できるようにする例については、「Linux Assembly Toolを使用してソースパッケージを追加する（54 ページ）」を参照してください。

3.2. input.yamlサンプルファイルのサポート情報

サンプルの***.yaml**ファイルを作成する際、いくつかの注意事項があります。

appsdk exampleyamls コマンドは、開発プロジェクトを立ち上げる際の参考として使用できるよう、SDK 内の ***.yaml** ファイルのリストを作成します。これらのサンプルファイルの作成の詳細については、「input.yamlサンプルファイルを作成する (31ページ)」を参照してください。このトピックに含まれるサンプル ファイルの内容の一覧は、「「Linux Assembly Tool Core input.yaml サンプルファイル (48ページ)」参照してください。

appsdkコマンドでgenimageやgencontainerなどの位置引数を使用して**input.yaml**ファイルを参照する場合、作成するイメージに固有の***.yaml**ファイルのみを参照することが重要です。たとえば、genimage位置引数でイメージの***.yaml**ファイルのみを参照し、コンテナやinitramfsの***.yaml**ファイルを参照しないようにします。異なるタイプの *.yaml を使用すると、イメージの作成に失敗する、不完全なイメージになって起動に失敗するといったことが発生します。

Image *.yaml ファイル

- core-image minimal, core-image-sato, wrlinux-image-small では、appsdk genimage コマンドでイメージ ***.yaml ファイルを**参照するのみです。
- **image_type** セクションには、**- ostree_repo** と **- ustart** オプションを別々の項目として含める必要があります。
- **packages**セクションは、利用可能なすべてのパッケージをリストアップするのではなく、以下の環境コマンドを使用して、**PACKAGE_INSTALL** BitBake変数を参照し、作成中のパッケージのリストを取得します。

  $ **bitbake -e** imageName

コンテナとinitramfsの*.yamlファイル

- コンテナベースのイメージの場合、**image_type**セクションに**-container**オプションが含まれている必要があります。appsdk gencontainerコマンドでコンテナ***.yaml**ファイルのみを参照します。
- initramfsイメージの場合、**image_type**セクションに**- initramfs**オプションが含まれていなければなりません。appsdk geninitramfsコマンドを使用して、initramfsの***.yaml**ファイルのみを参照します。

Feature *.yaml ファイル

Feature ***.yaml** ファイルは、パッケージセクションで機能そのものを参照することによって、追加機能を追加できるようにすることで、カスタマイズを簡素化するのに役立ちます。これらの機能は、Wind River Linuxで使用される機能テンプレートと同じです。利用可能なテンプレートオプションの一覧は、Wind River Linux Platform Developer's Guideを参照してください。Feature Templates by Layer Referenceを参照してください。

> 注　Linux Assembly Toolは、すべての機能の組み合わせに対応しているわけではありません。

Feature ***.yaml** ファイルを参照する場合、その機能が作成するイメージの種類と一致していることを確認します。例えば、**vboxguestdrivers.yaml** や **xfce_desktop.yaml などの** feature yaml ファイルはシステムイメージを必要とし、コンテナや initramfs ベースのイメージでは動作しません。これらを使用しようとすると、Linux Assembly Tool はイメージ作成中に検証エラーを報告します。

Wind River Linux Distro システムイメージ ***.yaml** ファイル (**wrlinux-image-small-intel- x86-64.yaml** など) でFeature ***.yaml** ファイルを参照する場合、次のFeature ***.yaml** ファイルは Wind River によって検証されています。

- **package_management.yaml**
- **vboxguestdrivers.yaml**

- **xfce_desktop.yaml**

サポートされているパッケージの一覧は、***.yaml**ファイルのpackagesセクションを参照してください。

3.3. ソースからビルドするバイナリベースコンテナ、SDK、パッケージフィード

Wind River Linux Distroのバイナリ・リリースイメージ、SDK、バイナリアップデートをソースから保守するために必要なパッケージフィードを構築する方法を紹介します。

このタスクについて

以下の手順は、バイナリ・リリースイメージとそれを維持するために必要なパッケージリポジトリを作成するための1つの方法です。もう1つの方法は、既存のバイナリ・リリースを再利用することです。詳細は、「Distroイメージによるベースコンテナ、SDK、パッケージフィードの構築（23 ページ）」を参照してください。

Note: この手順は、Wind River Linux LTSをお使いのお客様で、Wind River Linux Distroもお使いのお客様が対象です。

イメージを構築し、パッケージリポジトリを設定すると、Linux Assembly Tool を使用して、RPM パッケージの構築と公開、特定のハードウェア用のパッケージフィードからのイメージ作成、更新された SDK の作成、および SDK 構築の正常性の確認など、イメージ管理に役立つ多くのタスクを実行できるようになります。詳細については、「Linux Assembly Toolのインストールと実行 (17ページ）」を参照してください。

バイナリ・リリースでサポートされているBSPの一覧は、Wind River Linux Distro Quick Startを参照してください。

始める前に

- Wind River Linux Distro を使用するための要件を満たす Linux ホスト。詳細は、Wind River Linux Release Notes：Host System Recommendations and Requirementsを参照してください。
- クローンされたプラットフォームプロジェクトのGitリポジトリ。詳細は、Wind River Linux Platform Development Quick Start：Setting Up the Platform Project Development Environmentを参照してください。
- プラットフォームプロジェクトのビルドを行う際に、コンポーネントをダウンロードするためのインターネットアクセス。
- パッケージの更新を行うために専用のウェブサーバーを設置することを強く推奨します。この手順では、Python 3 http.serverを例としてのみ使用します。

手順

a. プラットフォームプロジェクトを設定します。

```
$ setup.sh --machines=intel-x86-64 --distro=wrlinux-graphics \
--dl-layers --all-layers
```

この例では、intel-x86-64 BSPを使用しています。バイナリビルドには、bcm-2xxx-rpi4 BSPを使用することもできます。

b. 開発環境を整える。

このステップは、プロジェクトのたびに実行する必要があります。

a. プロジェクトディレクトリから**環境設定**スクリプトを実行します。
このステップでは、Linux開発ホストシステムのツールを、ビルドシステムが必要とする特定のバージョンで補強します。

```
$ . ./environment-setup-x86_64-wrlinuxsdk-linux
```

b.oe-init-build-envスクリプトを実行します。

このステップでは、特定のビルドディレクトリに対して開発環境を設定します。以下のコマンドをオプションなしで実行すると、デフォルトのprojectDir **/build**がビルドディレクトリとして設定されます。

```
$ . ./oe-init-build-env
```

別のディレクトリを使用したい場合は、コマンドにディレクトリ名またはパスを付加してください。

```
$ . ./oe-init-build-env newBuildDir
```

コマンドが完了すると、ターミナルは指定されたビルドディレクトリに自動的に移動します。これで、プラットフォームプロジェクトを開発する準備が整いました。

c. ビルドディレクトリの**conf/local.conf**ファイルを更新して、以下を追加します。

以下のテキストをコピーしてターミナルに貼り付け、**local.conf**ファイルに内容を追加してください。

```
cat << ENDOF >> conf/local.conf
PACKAGE_FEED_BASE_PATHS = "rpm"
PACKAGE_FEED_URIS = "http://web-server-url :8080/lat"
ENDOF
```

この例の、web-server-urlを、スタンドアロンのウェブサーバーまたは開発用ホストのURLまたはIPアドレスに置き換えてください。

d. バイナリ・リリースをサポートするために必要なレシピをビルドします。

a. RPMレポジトリを構築します。

```
$ bitbake world && bitbake package-index
```

このコマンドは、すべてのパッケージをビルドし、DNF パッケージマネージャでパッケージを管理するために必要な repodata を作成します。Linuxホストシステムによっては、このコマンドの完了に時間がかかることがあります。

コマンドが完了すると、ビルドディレクトリ **tmp-glibc/deploy/rpm/arch / repodata/repmod.xml** ファイルなどのパッケージリポジトリデータが作成されます。

```
$ ls tmp-glibc/deploy/rpm/*/repodata/repomd.xml -1
tmp-glibc/deploy/rpm/corei7_64/repodata/repomd.xml
tmp-glibc/deploy/rpm/intel_x86_64/repodata/repomd.xml
tmp-glibc/deploy/rpm/noarch/repodata/repomd.xml
```

ビルドシステムは、BSPで定義されたMACHINEおよびDEFAULTTUNE設定に応じて、一致するアーキテクチャの追加のパッケージフィードを作成し、ビルドログで確認できるようにすることに注意してください。詳細については、「Wind River Platform Developer's Guide：Enabling Multilib Support in Platform Projects」を参照してください

b. ベースとなるリファレンスイメージを構築します。

```
$ bitbake imageName
```

この例では、imageNameはLinux Assembly Toolのサポートでビルドしたいイメージレシピの名前、例えば**wrlinux-image-std**を表しています。

c. SDKをビルドする。

```
$ bitbake imageName -c populate_sdk
```

このコマンドは、Linux Assembly Toolを使用したSDKを作成します。SDKは、**tmp-glibc/deploy/sdk/**ディレクトリに、拡張子***.shで配置されます。**

e. パッケージリポジトリを設定します。

注　専用Webサーバーをご利用の場合は、次のステップに進んでください。

Webサーバを簡単に作成する方法として、以下のようにPython3の**http.server**モジュールを活用する方法があります。

a. **tmp-glibc/deploy**ディレクトリに移動します。

```
$ cd tmp-glibc/deploy
```

b. 以下のコマンドを実行して、8080番ポートでサーバーを起動します。

```
$ python3 -m http.server 8080
```

f. 前のステップで作成したリポジトリへのシンボリックリンクを作成します。

```
$ ln -snf path-to-build /tmp-glibc/deploy /var/www/html/lat
```

この時点で、http**://web-server-url :8080/lat**ディレクトリは、インターネット上でアクセスできるようになっているはずです。

g. この手順で構築したイメージへのアクセスをテストするには、別のターミナルウィンドウからwgetコマンドを使用します。

イメージタイプ	実行するコマンド
ベースリファレンスコンテナ	`$ wget http://web-server-url :8080/lat/images/bspName /container-ba se-bspName .tar.bz2`
アプリケーションSDK	`$ wget http://web-server-url :8080/lat/sdk/wrlinux-graphics-10.2X .X X .X -glibc-x86_64-bspName -container-base-sdk.sh`

コンテナまたはSDKは、コマンドを実行した場所に正常にダウンロードされます。

次のステップ

初期ビルドが完了し、パッケージ リポジトリが稼働したら、バイナリ更新の管理に使用する Linux Assembly Tool を使用して SDK をインストールすることができます。詳細については、「Linux Assembly Toolのインストールと実行 (17 ページ) 」を参照してください。

3.4. Distroイメージによるベースコンテナ、SDK、パッケージフィードの構築

Wind River Linux Distroのバイナリ・リリースイメージ、SDK、バイナリアップデートをソースから保守するために必要なパッケージフィードを構築する方法を紹介します。

このタスクについて

以下の手順は、バイナリ・リリースイメージとそれを維持するために必要なパッケージリポジトリを作成するための1つの方法です。もう1つの方法は、ソースからイメージをビルドすることです。詳細は、「ソースからビルドするバイナリベースコンテナ、SDK、パッケージフィード (20ページ)」を参照してください。

イメージを構築し、パッケージリポジトリを設定すると、Linux Assembly Tool を使用して、RPM パッケージの構築と公開、特定のハードウェア用のパッケージフィードからのイメージ作成、更新された SDK の作成、および SDK 構築の正常性の確認など、イメージ管理に役立つ多くのタスクを実行できるようになりま す。詳細については、「Linux Assembly Toolのインストールと実行 (17ページ)」を参照してください。

バイナリ・リリースでサポートされているBSPの一覧は、Wind River Linux Distro Quick Startを参照してください。

始める前に

- Wind River Linux Distro を使用するための要件を満たす Linux ホスト。詳細については、Wind River Linux Release Notes：Host System Recommendations and Requirements参照してください。
- 以前にクローンされたプラットフォームプロジェクトのGitリポジトリ。詳細は、Wind River Linux Platform Development Quick Start：Setting Up the Platform Project Development Environmentを参照してください。
- プラットフォームプロジェクトのビルドを行う際に、コンポーネントをダウンロードするためのインターネットアクセス。
- パッケージの更新を行うために専用のウェブサーバーを使用することを強くお勧めします。この手順では、例としてPython 3 http.serverを使用しています。

手順

a. プラットフォームプロジェクトを設定します。

```
$ setup.sh --machines=intel-x86-64 --distro=wrlinux-graphics \
--dl-layers --all-layers
```

この例では、intel-x86-64 BSPを使用しています。バイナリビルドには、bcm-2xxx-rpi4 BSPを使用することもできます。

b. 開発環境を整える。

このステップは、プロジェクトのたびに実行する必要があります。

a. プロジェクトディレクトリから**環境設定**スクリプトを実行します。

このステップでは、Linux開発ホストシステムのツールを、ビルドシステムが必要とする特定のバージョンで補強します。

```
$ . ./environment-setup-x86_64-wrlinuxsdk-linux
```

b. **oe-init-build-env**スクリプトを実行します。

このステップでは、特定のビルドディレクトリに対して開発環境を設定します。以下のコマンドをオプションなしで実行すると、デフォルトのprojectDir

/buildがビルドディレクトリとして設定されます。

```
$ . ./oe-init-build-env
```

別のディレクトリを使用したい場合は、コマンドにディレクトリ名またはパスを付加してください。

```
$ . ./oe-init-build-env newBuildDir
```

コマンドが完了すると、ターミナルは指定されたビルドディレクトリに自動的に移動します。これで、プラットフォー・プロジェクトを開発する準備が整いました。

c. ビルドディレクトリの**conf/local.conf**ファイルを更新して、以下を追加します。

以下のテキストをコピーしてターミナルに貼り付け、**local.conf**ファイルに内容を追加してください。

```
cat << ENDOF >> conf/local.conf PACKAGE_FEED_URIS
= "http://web-server-url :8080/lat" ENDOF
```

この例では、web-server-urlを、スタンドアロンのウェブサーバーまたは開発用ホストのURLまたはIPアドレスに置き換えてください。

d. **gen-image**スクリプトで必要なコンポーネントをビルドします。

Wind River Linux CD	`$ ../layers/wrlinux/scripts/gen-image/gen-image -p lincd -m bspName`
Wind River Linux LTS 22	`$ ../layers/wrlinux/scripts/gen-image/gen-image -p lts-22 -m bspName`

この例では、**gen-image**スクリプトは以下のオプションを使用してイメージを作成します。

- **-m**は**-machine**の略で、マシン名、またはBSPを定義します。
- **-p** は **--product** の略で、製品を lincd として定義します。

オプションとして、**-o**、**--outdir**オプションで出力ディレクトリを指定することができます。デフォルトでは、ビルドディレクトリの**outdir**サブディレクトリに出力イメージとリポジトリが作成されます。

e. パッケージリポジトリを設定します。

注　専用Webサーバーをご利用の場合は、次のステップに進んでください。

Webサーバを簡単に作成する方法として、以下のようにPython3の**http.server**モジュールを活用する方法があります。

a. ビルドディレクトリ **outdir** サブディレクトリに移動します。

```
$ cd outdir
```

b. 以下のコマンドを実行して、8080番ポートでサーバーを起動します。

```
$ python3 -m http.server 8080
```

f. 前のステップで作成したリポジトリへのシンボリックリンクを作成します。

```
$ ln -snf path-to-build /outdir /var/www/html/lat
```

この時点で、http**://web-server-url :8080/assembly-tool**ディレクトリは、インターネット経由でアクセスできるようになっているはずです。

g. この手順で構築したイメージへのアクセスをテストするには、別のターミナルウィンドウからwgetコマンドを使用します。

イメージタイプ	実行するコマンド
ベースリファレンスコンテナ	`$ wget http://web-server-url :8080/lat/lts-21/bspName /lat-bspName / container-base-bspName .tar.bz2`
アプリケーションSDK	`$ wget http://web-server-url :8080/lat/lts-21/bspName /lat-bspName / wrlinux-graphics-10.2X .XX .X -glibc-x86_64-bspName -container-ba se-sdk.sh`

コンテナまたはSDKは、コマンドを実行した場所に正常にダウンロードされるはずです。

次のステップ

初期ビルドが完了し、パッケージ リポジトリが稼働したら、バイナリ更新の管理に使用する Linux Assembly Tool を使用して SDK をインストールすることができます。詳細については、「Linux Assembly Toolのインストールと実行 (17ページ) 」を参照してください。

3.5. プラットフォームビルド環境でLinux Assembly Toolを使う

Linux Assembly Toolは、Distroリリースリポジトリを管理するための一般的なタスクを、ビルド環境から直接実行できるようにするものです。

このタスクについて

このタスクには、RPMパッケージの構築と公開、特定のハードウェア用のパッケージフィードからのイメージ生成などが含まれます。この Linux Assembly Tool は、x86-64 の Linux ホストシステムでサポートされています。

> 注　Linux Assembly Tool は、ビルド環境からの SDK 生成をサポートしていません。そのため、**gensdk**および**checksdk**位置引数は使用できません。

始める前に

- 「Distroイメージによるベースコンテナ、SDK、パッケージフィードの構築（23ページ）」で説明したプラットフォームプロジェクトのビルドが完了し、以下のsetup.shのオプションが設定されている必要があります。

```
$ setup.sh --machines=intel-x86-64 --distro=wrlinux-graphics ¥
--dl-layers --all-layers
```

手順

a. 開発環境をセットアップします。
このステップは、プロジェクトのたびに実行する必要があります。

1. プロジェクトディレクトリから**environment-setup**スクリプトを実行します。
このステップでは、Linux開発ホストシステムのツールを、ビルドシステムが必要とする特定のバージョンで補強します。

```
$ . ./environment-setup-x86_64-wrlinuxsdk-linux
```

2. **oe-init-build-env**スクリプトを実行します。
このステップでは、特定のビルドディレクトリのために開発環境を設定します。次のコマンドをオプションなしで実行します。
デフォルトのprojectDir **./build**をビルドディレクトリとして設定するために、オプションなしで次のコマンドを実行します。

```
$ . ./oe-init-build-env
```

別のディレクトリを使用したい場合は、コマンドにディレクトリ名またはパスを付加してください。

```
$ . ./oe-init-build-env newBuildDir
```

コマンドが完了すると、ターミナルは指定されたビルドディレクトリに自動的に移動します。
これで、プラットフォームプロジェクトを開発する準備が整いました。

b. **appsdk-native**レシピをビルドして、appsdkの実行ファイルを作成します。

プラットフォームプロジェクトのビルドディレクトリから、以下のコマンドを実行します。

パッケージオプション	実行するコマンド
Minimal RPMs	このオプションは、Linux Assembly Toolを、その実行に必要な最小限のパッケージを構築します。 `$ bitbake appsdk-native && bitbake package-index build-sysroots`
Full RPMs	このオプションは、Linux Assembly Tool を、platform project configuration で利用可能なすべてのパッケージをビルドします。プラットフォームプロジェクトの構成で利用可能なすべてのパッケージで構築します。 `$ bitbake world appsdk-native && bitbake package-index build-sysroots` 注 Linux Assembly Tool は、ビルド環境からの SDK 生成をサポー gensdkおよびchecksdk位置引数は使用できません。 Linuxホストシステムによっては、このコマンドの実行に時間がかかる場合があります。

コマンドが完了すると、appsdkの実行ファイルがビルドディレクトリの **tmp-glibc/sysroots/x86_64/usr/bin/** サブディレクトリに作成されます。

c. オプションで、ツールのBashシェル補完を有効にします。

有効にすると、Linux Assembly Toolは、ツールの位置引数およびサブコマンドのBashシェルタブ補完をサポートします。

appsdkと位置引数の先頭をタイプして**TAB**キーを押すだけで補完されるか、利用可能なオプションが表示されます。

```
$ bash --rcfile tmp-glibc/sysroots/x86_64/environment-appsdk-native
```

d. Linux Assembly Toolを起動し、**-h**オプションで使用オプションを表示します。

```
$ tmp-glibc/sysroots/x86_64/usr/bin/appsdk -h
usage: appsdk [-h] [-d] [-q] [--log-dir LOGDIR]
25 | Documentation
Wind River Linux Distro Developer's Guide, LTS 22
{genrpm,publishrpm,genimage,geninitramfs,gencontainer,genyaml,exampleyamls}
...
Wind River Linux Assembly Tool
positional arguments:
{gensdk,checksdk,genrpm,publishrpm,genimage,geninitramfs,gencontainer,genyaml,exampleyamls}
Subcommands. "appsdk <subcommand> --help" to get more info
genrpm Build RPM package
publishrpm Publish RPM package
genimage Generate images from package feeds for specified machines
geninitramfs Generate Initramfs from package feeds for specified machines
gencontainer Generate Container Image from package feeds for specified
machines
genyaml Generate Yaml file from Input Yamls
exampleyamls Deploy Example Yaml files
optional arguments:
   -h, --help show this help message and exit
   -d, --debug Enable debug output
    -q, --quiet Hide all output except error messages
    --log-dir LOGDIR Specify dir to save debug messages as log.appsdk regardless of the logg
       ing level
Use appsdk <subcommand> --help to get help
```

このツールは、オプションの引数とともに、利用可能なアクションを定義するための位置引数を使用していることに注意してください。位置引数に関する追加のヘルプを取得するには 位置引数に関する追加のヘルプを得るには、**-h**オプションでその引数を実行します。

```
$ appsdk genimage -h
usage: appsdk genimage [-h] [-t {wic,vmdk,vdi,ostree-repo,ustart,all}]
[-o OUTDIR] [-w WORKDIR] [-n NAME] [-u URL] [-p PKG] [--pkg-external
PKG_EXTERNAL][--rootfs-post-script ROOTFS_POST_SCRIPT]
[--rootfs-pre-script ROOTFS_PRE_SCRIPT] [--env ENV] [--no-clean]
[--no-validate] [-g GPGPATH][input [input ...]]

positional arguments:
   input            Input yaml files that the tool can be run
                    against a package feed to generate an image

optional arguments:
   -h, --help       show this help message and exit
   -t, --type {wic,vmdk,vdi,ostree-repo,ustart,all}
                    Specify image type, it overrides 'image_type'
                    in Yaml
   -o OUTDIR, --outdir OUTDIR
                    Specify output dir, default is current working
                    directory
   -w WORKDIR, --workdir WORKDIR
                    Specify work dir, default is current working
                    directory
   -n NAME, --name NAME Specify image name, it overrides 'name' in Yaml
   -u URL, --url URL Specify extra urls of rpm package feeds
   -p PKG, --pkg PKG Specify extra package to be installed

   --pkg-external PKG_EXTERNAL
                    Specify extra external package to be installed
   --rootfs-post-script ROOTFS_POST_SCRIPT
                    Specify extra script to run after do_rootfs
   --rootfs-pre-script ROOTFS_PRE_SCRIPT
                    Specify extra script to run before do_rootfs
   --env ENV Specify extra environment to export before
                    do_rootfs: --env NAME=VALUE
   --no-clean       Do not cleanup generated rootfs in workdir
   --no-validate    Do not validate parameters in Input yaml files
   -g GPGPATH, --gpgpath GPGPATH
                    Specify gpg homedir, it overrides 'gpg_path' in
                    Yaml, default is /tmp/.lat_gnupg
```

利用可能なすべてのオプションの一覧は、「Linuxアセンブリツールオプションリファレンス（38 ページ）」を参照してください。

次のステップ

Linux Assembly Toolが立ち上がると、これを利用してDistroリリースを管理することができます。新しいパッケージをソースから追加して パッケージをソースから追加し、外部のターゲットデバイスで利用できるようにする例は、「Linux Assembly Toolを使用してソースパッケージを追加する（54 ページ）」を参照してください。

3.6. Wi-Fiを利用したネットワークベースのインストールを実行する

イメージにWi-Fiサポートが含まれると、Wi-Fiを使ったネットワークベースのインストールを行うことができます。

このタスクについて

Wi-Fi導入の主なメリットの1つは携帯性で、特にデバイスが固定のネットワークコントローラーに縛られることがありません。

始める前に

- 「ネットワークベースのインストールにWi-Fiサポートを追加する（64ページ）」で説明するように、インストールまたはインストールしたイメージに Wi-Fi サポートがあるシステムイメージが必要です。「Linux Assembly Toolのインストールと実行 (17ページ)」で説明するように、以前にインストールした SDK のいずれかにサポートしている必要があります。
- インストールおよびインストールされたイメージがサポートするために作成されたBSPのハードウェアボードを持っている必要があります。Wi-Fiは実際のハードウェアが必要であり、QEMUエミュレーションでは動作しません。
- この手順で使用する Python ベースのサーバーよりも、OSTree パッケージリポジトリ用に、より堅牢な Linux ホスト（Web サーバー）を使用する必要があるかもしれません。また、例として提供されている「Distroイメージによるベースコンテナ、SDK、パッケージフィードの構築（23 ページ）」でも説明されています。

手順

a. 「Distroイメージによるベースコンテナ、SDK、パッケージフィードの構築（23 ページ）」の説明に従って、パッケージフィードと Web サーバーをセットアップして、リモート OSTree リポジトリを提供します。

b. 「ネットワークベースのインストールにWi-Fiサポートを追加する（64 ページ）」で説明するように、インストールイメージの作成にリポジトリURLが使用されていることを確認します。

これは、インストールプロセスの一環として、必要なパッケージをWi-Fiネットワーク経由でイメージから取得できるようにするためと、配備されたシステムのアップデートを提供するために必要です。

c. ハードウェアボードでWi-Fi対応イメージを起動します。ボード固有の詳細については

詳しくは、Wind River Linux Distro Quick Start：「Booting Up the Target System Image on Hardware」参照してください。

ハードウェア固有の情報については、下記で説明するダウンロードした *.tar.gz ファイル内の imageType **.README.md** ファイルを参照してください。
Wind River Linux Distro Quick Start：「Downloading the images and SDK」をダウンロードする。

d. デプロイされたイメージでwlan0がアクティブになっていることを確認します。

```
# ifconfig wlan0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 128.224.176.100  netmask 255.255.254.0  broadcast 128.224.177.255
    inet6 fe80::bc29:4395:60ce:54c9  prefixlen 64  scopeid 0x20<link>
    ether dc:a6:32:fa:0d:5b  txqueuelen 1000  (Ethernet)
    RX packets 36  bytes 10456 (10.2 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 37  bytes 4239 (4.1 KiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

e. オプションで、OSTreeリポジトリへのアクセスを確認するためのシステムアップデートを実行します。

詳しくは、Wind River Linux Distro Quick Start：「Updating Binary Images」ご覧ください。

3.7. 一般的な使用例

3.7.1. input.yamlサンプルファイルを作成する

Linux Assembly Toolは、開発ニーズに合わせてカスタマイズ可能な**input.yaml**ファイルのサンプルリストを作成する方法を提供します。

このタスクについて

注	**input.yaml**のファイル名は参考程度にお考えください。拡張子が**.yaml**であれば、用途に応じたファイル名を付けることができます。

これらの *.yaml ファイルは、RPM パッケージの構築と公開など、開発タスクの構成基盤を提供します。特定のハードウェアのパッケージ フィードからのイメージの生成、更新された SDK の生成、および SDK ビルドの正常性の検証。Linux Assembly Tool は、x86-64 Linux ホストシステムでサポートされています。

始める前に

Linux Assembly Toolのインストールと実行（17ページ）で説明したように、SDKが事前にインストールされている必要があります。

手順

a. SDKを使用できるように環境を整える。

sdkDirに移動して、環境設定スクリプトを実行します。

```
$ . ./environment-setup-*-wrs-linux
```

b. Linux Assembly Toolを**exampleyamlsの**位置引数で実行します。

```
$ appsdk exampleyamls
appsdk - INFO: Deploy Directory: /opt/windriver/wrlinux-graphics/20.38/exampleyamls
+------------------+-----------------------------------------+
|      Yaml Type      |                   Name                    |
+==================+=========================================+
| Image              | container-base-intel-x86-64.yaml          |
|                    |  core-image-minimal-intel-x86-64.yaml     |
```

```
|                    | core-image-sato-intel-x86-64.yaml         |
|                    | initramfs-ostree-image-intel-x86-64.yaml |
|                    |  wrlinux-image-small-intel-x86-64.yaml     |
|                    |                                           |
+--------------------+-------------------------------------------+
| Feature            | feature/debug-tweaks.yaml                 |
|                    | feature/package_management.yaml           |
|                    | feature/set_root_password.yaml            |
|                    | feature/startup-container.yaml            |
|                    | feature/vboxguestdrivers.yaml             |
|                    | feature/xfce_desktop.yaml                 |
|                    |                                           |
+--------------------+-------------------------------------------+
| System Definition | sysdef/add-system-user.yaml                |
|  Yamls             | sysdef/add-user-home.yaml                 |
|                    | sysdef/contains-container-base.yaml       |
|                    | sysdef/set-dns.yaml                       |
|                    | sysdef/set-hostname.yaml                  |
|                    | sysdef/set-ntp.yaml                       |
|                    | sysdef/update-containers.yaml             |
|                    |                                           |
+--------------------+-------------------------------------------+
appsdk - INFO: Then, run genimage or genyaml with Yaml Files:
appsdk genimage <Image>.yaml <Feature>.yaml
Or
appsdk genyaml <Image>.yaml <Feature>.yaml
```

コマンドが完了すると、作業ディレクトリ **exampleyamls** サブディレクトリに一連の ***.yaml** ファイルが追加されます。

```
$ tree -L 2 exampleyamls
exampleyamls
├── container-base-intel-x86-64.yaml
├── core-image-minimal-intel-x86-64.yaml
├── core-image-sato-intel-x86-64.yaml
├── feature
│   ├── debug-tweaks.yaml
│   ├── package_management.yaml
│   ├── set_root_password.yaml
│   ├── startup-container.yaml
│   ├── vboxguestdrivers.yaml
│   └── xfce_desktop.yaml
├── initramfs-ostree-image-intel-x86-64.yaml
├── sysdef
│   ├── add-system-user.yaml
│   ├── add-user-home.yaml
│   ├── contains-container-base.yaml
│   ├── files
│   │   ├── sudoers_sudo
│   │   └── dns.conf
│   ├── run_always.d
│   │   └── 10_start_containers.sh
│   ├── run_once.d
│   │   ├── 10_add_system_user.sh
│   │   ├── 20_add_user_home.sh
│   │   ├── 30_set_hostname.sh
│   │   └── 40_set_ntp.sh
│   ├── run_on_upgrade.d
│   │   ├── 10_update_containers.sh
│   │   └── containers.dat
│   ├── set-dns.yaml
│   ├── set-hostname.yaml
│   ├── set-ntp.yaml
│   └── update-containers.yaml
└── wrlinux-image-small-intel-x86-64.yaml
```

これらのサンプルファイルは、イメージを修正する際のベースとして使用することができます。これらのファイルの使用に関する追加情報については、「input.yamlサンプルファイルのサポート情報（19ページ）」を参照してください。

関連情報

新しいSDKの作成とSDKの正常性の確認（33 ページ）
アプリケーションソースからのRPMパッケージの作成と公開（34 ページ）
システムイメージの作成（36 ページ）
コンテナイメージの作成（37 ページ）
OSTreeによるinitramfsイメージの作成（39ページ）
既存のファイルから結合されたinput.yamlファイルを作成する（40 ページ）

3.7.2. 新しいSDKの作成とSDKの正常性の確認

Assembly Tool は、新しい SDK を作成し、既存の SDK の整合性を検証するために使用できます。

始める前に

SDKがインストールされ、ソースが提供されており、Linux Assembly Toolを実行できる必要があります。詳細については、「Linux Assembly Toolのインストールと実行 (17ページ)」を参照してください。

手順

a. SDKを使用できるように環境を整える。

sdkDirに移動して、環境設定スクリプトを実行します。

```
$ . ./environment-setup-*-wrs-linux
```

b. SDKを作成する。

```
$ appsdk gensdk -f image-file .yaml -o output-dir
```

この例では、イメージを定義するための ***.yaml** ファイルを **-f** (file) オプションで指定し、出力ディレクトリを **-o** (output) オプションで指定しています。オプションを付けずにコマンドを実行した場合、出力ディレクトリはデフォルトで **deploy** サブディレクトリになります。

c. SDKの整合性を確認する。

```
appsdk checksdk
```

コマンドが完了すると、ターミナルにステータスの情報が表示されます。

関連情報

input.yamlサンプルファイルを作成する (31ページ)
アプリケーションソースからのRPMパッケージの作成と公開（34ページ）
システムイメージの作成（36 ページ）
コンテナイメージの作成（37 ページ）
OSTreeによるinitramfsイメージの作成（39 ページ）
既存のファイルから結合されたinput.yamlファイルを作成する（40ページ）

3.7.3. アプリケーションソースからのRPMパッケージの作成と公開

Linux Assembly Toolを使用して、既存のアプリケーションバイナリのパッケージを作成します。

このタスクについて

Wind River Linux Distroイメージにカスタムアプリケーションを含めるには、まず、アプリケーションをビルドして、パッケージとしてOSTreeパッケージリポジトリに公開する必要があります。追加されたら、イメージの **input.yaml** ファイルの **packages** セクションにパッケージを追加することで、イメージにパッケージを含めることができます。

この手順では、アプリケーションからパッケージを作成する方法と、次にそれを公開するための場所について説明します。

始める前に

- Linux Assembly Toolのインストールと実行 (17ページ)で説明したように、SDKが事前にインストールされている必要があります。
- SDKで作成されたビルド済みアプリケーションが必要です。詳細については、「Linux Assembly Toolを使用してソースパッケージを追加する（54 ページ）を参照してください。

手順

a. SDKを使用できるように環境を整える。

sdkDirに移動して、環境設定スクリプトを実行します。

```
$ . ./environment-setup-*-wrs-linux
```

b. パッケージ情報を指定するために、appName **.yaml** または RPM spec ファイルを作成します。以下は、hello アプリケーションの appName **.yaml** ファイルの例です。

```
name: hello
version: '2.10'
release: r0
概要：Hello World Program From Gnu
license:GPLv3
description: |
  A simple hello world program that only does one thing.
  It's from GNU.

# dirs:
# - /usr
# - /usr/local
# - /usr/local/bin

files:
  - /usr/local/bin/hello

post_install: |
  #!/bin/sh
  echo "This is the post install script of hello program"
  echo "It only prints some message."
```

この例では、**dirs:** と **files:** セクションは、パッケージ化する特定のディレクトリを定義するのに役立ちます。例えば、次のステップのappsdk genrpmコマンドは、デフォルトですべてのファイルをパッケージ化しますが、***.yaml**ファイルで特定のディレクトリを設定し、そのディレクトリに含まれる特定のファイルをパッケージ化するために使用することができます。

c. appName **.yaml** ファイルを使用して、RPM パッケージを作成します。

```
$ appsdk genrpm -f appName .yaml -i /path-to /install-appName
```

この例では、パッケージを定義するための ***.yaml** ファイルに **-f** (file) オプションを指定し、インストール先ディレクトリに **-i** (installdir) オプションを指定しています。

パッケージが作成されたので、それをシステムイメージに追加するためのいくつかのオプションがあります。

パッケージをリポジトリに公開する

これにより、デバイス上でdnf updateを実行すると、デプロイされたデバイスでパッケージが利用できるようになります。

システムイメージにパッケージを追加する

これにより、システムイメージにパッケージが追加され、各新規イメージにシステム展開の一部としてパッケージが含まれるようになります。詳しくは、以下を参照してください。

これらのオプションは両方とも、以前にデプロイされたシステムイメージにパッケージを追加する機能を提供します。詳細については、「Linux Assembly Tool システム定義の概要（3 ページ）」をご覧ください。

関連情報

input.yamlサンプルファイルを作成する (31ページ)
新しいSDKの作成とSDKの正常性の確認（33 ページ）
システムイメージの作成（36 ページ）
コンテナイメージの作成（37ページ）
OSTreeによるinitramfsイメージの作成（39 ページ）
既存のファイルから結合されたinput.yamlファイルを作成する（40 ページ）

3.7.4. システムイメージの作成

Linux Assembly Tool を使用して、サンプルの **input.yaml** ファイルから Wind River Linux Distro システムイメージを作成します。

このタスクについて

システムイメージを作成するために必要なのは、特定の構成要件を含めて更新されたinput.yamlファイルだけです。ウインドリバーでは、イメージをカスタマイズするためのベースラインとして、参考となる***.yaml**ファイルの例を提供しています。カスタマイズが完了したら、appsdk genimageコマンドを使用してイメージを作成できます。

この手順では、appsdk genimageコマンドを使用してシステムイメージを作成する簡略化した例を示します。リモートでイメージを更新するためのパッケージリポジトリと一緒にイメージを作成することを含む、より詳細な手順については、「Distroイメージによるベースコンテナ、SDK、パッケージフィードの構築（23ページ）」を参照してください。

始める前に

- Linux Assembly Toolのインストールと実行 (17ページ)で説明したように、SDKが事前にインストールされている必要があります。
- 参考となるサンプルイメージ ***.yaml** ファイルを用意する必要があります。詳細は、「input.yamlサンプルファイルを作成する (31ページ)」を参照してください。

手順

a. SDKを使用できるように環境を整える。

   sdkDirに移動して、環境設定スクリプトを実行します。

   ```
   $ . ./environment-setup-*-wrs-linux
   ```

b. sdkDir **/exampleyamls**ディレクトリにあるシステムイメージサンプル **wrlinux-image-small**-arch **.yaml** ファイルをコピーします。

   例えば、新しいファイルに **my-custom-image.yaml** という名前を付けます。

c. イメージに必要な追加機能やシステム構成があれば、ファイルを編集してください。

   利用可能なカスタマイズの詳細については、本書の他のトピックを参照してください。

d. 更新した**my-custom-image.yaml**ファイルからイメージを作成する。

```
$ appsdk genimage exampleyamls/my-custom-image.yaml
```

次のイメージは、典型的なコマンド出力を示しています。

```
appsdk - Input YAML File: my-custom-image
appsdk - INFO: Machine: intel-x86-64
appsdk - INFO: Image Name: wrlinux-image-small
appsdk - INFO: Image Type: ostree-repo ustart
appsdk - INFO: Packages Number: 16
appsdk - INFO: External Packages Number: 0
appsdk - INFO: Package Feeds:
http://url/lat/dist/intel-x86-64/repos/rpm/corei7_64
http://url/lat/dist/intel-x86-64/repos/rpm/intel_x86_64
http://url/lat/dist/intel-x86-64/repos/rpm/noarch

appsdk - INFO: enviroments: ['NO_RECOMMENDATIONS="0"', 'KERNEL_PARAMS="key=value"']
appsdk - INFO: Create Rootfs: Started
appsdk - INFO: Create Rootfs: Succeeded(took 81 seconds)
appsdk - INFO: Create Initramfs: Started
appsdk - INFO: Reuse existed Initramfs
appsdk - INFO: Create Initramfs: Succeeded(took 0 seconds)
appsdk - INFO: Create Ostree Repo: Started
appsdk - INFO: Create Ostree Repo: Succeeded(took 57 seconds)
appsdk - INFO: Create Ustart Image: Started
appsdk - INFO: Create Ustart Image: Succeeded(took 21 seconds)
appsdk - INFO: Deploy Directory: /opt/windriver/wrlinux-graphics/20.38/deploy
+------------------+-------------------------------------------------+
|       Type       |                      Name                       |
+==================+=================================================+
| Image Yaml File  | my-custom-image.yaml                            |
+------------------+-------------------------------------------------+
| Ostree Repo      | ostree_repo                                     |
+------------------+-------------------------------------------------+
| Ustart Image     | wrlinux-image-small-intel-x86-64.ustart.img.gz -> |
|                  | wrlinux-image-small-                            |
|                  | intel-x86-64-20200916193051.ustart.img.gz       |
+------------------+-------------------------------------------------+
| Ustart Image Doc | wrlinux-image-small-intel-x86-64.ustart.img.gz. |
|                  | README.md                                       |
+------------------+-------------------------------------------------+
```

ビルドが完了すると、新しいイメージが **deploy** サブディレクトリに、コマンド出力の最後に記載されているファイルと共に置かれます。

この例では、イメージの名前に **my-custom-image.yaml** ファイルの情報が使われています。
wrlinux-image-small-intel-x86-64.yaml ファイルを作成します。その結果、コンテナイメージの名前がオリジナルと一致するようになりました。

genimageの位置引数に以下の**--type**オプションを指定して実行すると、イメージの起動と実行方法を説明したtype固有のREADMEファイルが作成されます。

- **--type ustart**: **wrlinux-image-small-intel-x86-64.ustart.README.md。**
- **--type wic**: **wrlinux-image-small-intel-x86-64.wic.README.md**

関連情報

input.yamlサンプルファイルを作成する (31ページ)
新しいSDKの作成とSDKの正常性の確認（33 ページ）
アプリケーションソースからのRPMパッケージの作成と公開（34 ページ）
コンテナイメージの作成（37 ページ）
OSTreeによるinitramfsイメージの作成（39 ページ）
既存のファイルから結合されたinput.yamlファイルを作成する（40ページ）

3.7.5. コンテナイメージの作成

Assembly Tool を使用して、サンプルの **input.yaml** ファイルから Wind River Linux Distro コンテナイメージを作成します。

このタスクについて

コンテナイメージを作成するために必要なのは、特定の構成要件を含めて更新された**input.yaml**ファイルだけです。ウインドリバーでは、イメージをカスタマイズするためのベースラインとして、参考となるサンプルの***.yamlファイルを**提供しています。カスタマイズが完了したら、appsdk gencontainerコマンドを使用してイメージを作成することができます。

始める前に

- Linux Assembly Toolのインストールと実行 (17ページ)で説明したように、SDKが事前にインストールされている必要があります。
- 参考となるサンプルイメージ ***.yaml** ファイルを用意する必要があります。詳細は、「input.yamlサンプルファイルを作成する (31ページ)」を参照してください。

手順

a. SDKを使用できるように環境を整える。

sdkDirに移動して、環境設定スクリプトを実行します。

```
$ . ./environment-setup-*-wrs-linux
```

b. sdkDir **/exampleyamlsに**あるコンテナイメージsample **container-base-arch -startup-container.yaml** ファイルをコピーしてください。ディレクトリに保存されます。

例えば、新しいファイルに **my-custom-container.yaml という**名前を付けます。

c. コンテナに必要な追加機能やシステム構成があれば、ファイルを編集してください。

利用可能なカスタマイズの詳細については、本書の他のトピックを参照してください。

d. 更新された **my-custom-container.yaml** ファイルからコンテナイメージを作成します。

```
$ appsdk gencontainer exampleyamls/my-custom-container.yaml
```

次のイメージは、典型的なコマンド出力を示しています。

```
appsdk - INFO: Input YAML File: my custom-container.yaml
appsdk - INFO: Machine: intel-x86-64
appsdk - INFO: Image Name: container-base
appsdk - INFO: Image Type: container
appsdk - INFO: Packages Number: 10
appsdk - INFO: External Packages Number: 0
appsdk - INFO: Package Feeds:
http://url/lat/dist/intel-x86-64/repos/rpm/corei7_64
http://url/lat/dist/intel-x86-64/repos/rpm/intel_x86_64
http://url/lat/dist/intel-x86-64/repos/rpm/noarch

appsdk - INFO: enviroments: ['NO_RECOMMENDATIONS="1"']
appsdk - INFO: Create Rootfs: Started
appsdk - INFO: No Kernel Modules found, not running depmod
appsdk - INFO: Create Rootfs: Succeeded(took 35 seconds)
appsdk - INFO: Create Docker Container: Started
appsdk - INFO: Create Docker Container: Succeeded(took 4 seconds)
appsdk - INFO: Deploy Directory: /opt/windriver/wrlinux-graphics/20.38/deploy
+------------------------+-----------------------------------------------------+
| Docker Image           | container-base-intel-x86-64.docker-image.tar.bz2 -> |
|                        | container-base-intel-x86-64-20201016031435.docker-  |
|                        | image.tar.bz2                                       |
+------------------------+-----------------------------------------------------+
| OCI Image Rootfs       | container-base-intel-x86-64.rootfs-oci              |
+------------------------+-----------------------------------------------------+
| Container Image Doc    | container-base-intel-x86-64.container.README.md     |
+------------------------+-----------------------------------------------------+
| Yaml file for genimage | container-base-intel-x86-64.startup-container.yaml  |
| to load and run        |                                                     |
+------------------------+-----------------------------------------------------+
```

ビルドが完了すると、新しいイメージは **deploy** サブディレクトリで利用可能になります。さらに、このコマンドは **container-base-**arch **.container.README.md** にイメージの起動と実行の手順が書かれています。

この例では、イメージ名には **my-custom-container.yaml** ファイルの情報を使用しています。**container-base-intel-x86-64.yaml** ファイルを修正しました。その結果、コンテナイメージの名前がオリジナルと一致するようになりました。

gencontainer位置指定引数では、コンテナイメージとREADMEファイルに加え、**container-base-arch - startup-container.yaml**ファイルも作成される。このファイルを使用して、コンテナに対して実行するDocker起動アクションを指定します。追加情報については、「システムイメージにコンテナを追加する（57ページ）」を参照してください。

関連情報

input.yamlサンプルファイルを作成する (31ページ)
新しいSDKの作成とSDKの正常性の確認（33 ページ）
アプリケーションソースからのRPMパッケージの作成と公開（34 ページ）
コンテナイメージの作成（37 ページ）
OSTreeによるinitramfsイメージの作成（39 ページ）
既存のファイルから結合されたinput.yamlファイルを作成する（40 ページ）

3.7.6. OSTreeによるinitramfsイメージの作成

Linux Assembly Tool を使用して、サンプルの **input.yaml** ファイルから Wind River Linux Distro initramfs ベースのシステムイメージを作成します。

このタスクについて

initramfsシステムイメージを作成するために必要なのは、特定の構成要件を含むように更新されたinput.yamlファイルだけです。ウインドリバーでは、イメージをカスタマイズするためのベースラインとして参照できる、サンプルの***.yaml**ファイルを提供しています。カスタマイズが完了したら、appsdk genimageコマンドを使用してイメージを作成することができます。

始める前に

- Linux Assembly Toolのインストールと実行 (17ページ)で説明したように、SDKが事前にインストールされている必要があります。
- 参考となるサンプルイメージ ***.yaml** ファイルを用意する必要があります。詳細は、「input.yamlサンプルファイルを作成する (31ページ）」を参照してください。

手順

a. SDKを使用できるように環境を整える。

sdkDirに移動して、環境設定スクリプトを実行します。

```
$ . ./environment-setup-*-wrs-linux
```

b. sdkDir **/exampleyamls**ディレクトリにあるinitramfs image sample **initramfs-ostree-image-**arch **.yaml**ファイルをコピーしてください。

例えば、新しいファイルに **my-custom-initramfs-image.yaml という**名前を付けます。

c. イメージに必要な追加機能やシステム構成があれば、ファイルを編集してください。

利用可能なカスタマイズの詳細については、本書の他のトピックを参照してください。

d. 更新された **my-custom-initramfs-image.yaml** ファイルからイメージを作成します。

```
$ appsdk geninitramfs exampleyamls/my-custom-initramfs-image.yaml
```

ビルドが完了すると、新しいイメージは **deploy** サブディレクトリで利用可能になります。

この例では、イメージ名には、オリジナルの **initramfs-ostree-image-intel-x86-64.yaml** ファイルのコピーである **my-custom-initramfs-image.yaml** ファイルの情報が使用されています。その結果、initramfsのイメージ名はオリジナルと一致します。

> 注
>
> 元の**initramfs-ostree-image-*.yamlファイルを**変更し、ファイル名を変更しない場合、appsdk genimageコマンドを使用してイメージを生成する際に、appsdkツールはこの変更したファイルをシステムの既定値として使用します。

関連情報

input.yamlサンプルファイルを作成する (31ページ)
新しいSDKの作成とSDKの正常性の確認（33 ページ）
アプリケーションソースからのRPMパッケージの作成と公開（34 ページ）
システムイメージの作成（36 ページ）
コンテナイメージの作成（37 ページ）
既存のファイルから結合されたinput.yamlファイルを作成する（40 ページ）

3.7.7. 既存のファイルから結合されたinput.yamlファイルを作成する

開発を簡単にするために、複数の既存の ***.yaml** ファイルから単一の **input.yaml** ファイルを作成することができます。

このタスクについて

システムまたはコンテナイメージの開発を続けると、多くの異なる **input.yaml** ファイルを全体的な設定に含めることが可能になります。たとえば、イメージを定義するために ***.yaml** ファイルを使用し、Wi-Fi 設定を指定するために別のファイルを使用し、さらにイメージにユーザーを追加するために別のファイルを使用することができます。イメージを作成するときには、それぞれの ***.yaml** ファイルを個別に指定する必要があります。この手順では、appsdk genyamlコマンドを使用して、すべての***.yaml**ファイルを1つのファイルに結合する方法について説明します。完了したら、appsdk genimageなどのコマンドを使用して、結合された***.yaml**ファイルを参照することができます。

```
appsdk genyaml path_to /your-image-yaml-file path_to /your-yaml-file1 path_to /your-yaml-fileN
```

*.yamlファイルを結合するための構文は次のとおりです。

結合する ***.yaml** ファイルは、完全でエラーがないものでなければなりません。たとえば、YOUR_OSTREE_REMOTE_URL などの変数は、正しい値が含まれるように更新する必要があります。

始める前に

- Linux Assembly Toolのインストールと実行 (17ページ)で説明したように、SDKが事前にインストールされている必要があります。
- 少なくとも 1 つのシステム、コンテナ、または initramfs イメージ ***.yaml** ファイルと、結合したい追加の ***.yaml** ファイルが必要です。オプションとして、「 input.yamlサンプルファイルを作成する (31ページ)」で説明するサンプル ***.yaml** ファイルを使用することができます。

手順

a. SDKを使用できるように環境を整える。

sdkDirに移動して、環境設定スクリプトを実行します。

```
$ . ./environment-setup-*-wrs-linux
```

b. 結合された ***.yaml** ファイルを作成します。

目的	方法
	$ appsdk genyaml exampleyamls/wrlinux-image-small-intel-x86-64.yaml ** **exampleyamls/feature/package_management.yaml exampleyamls/feature/debug-tweaks.yaml この例では、sdkDir **/ exampleyamls/feature**ディレクトリで提供される2つの機能 ***.yaml**ファイルを指定します。 次のイメージは、典型的なコマンド出力を示しています。 `appsdk - INFO: Input YAML File: exampleyamls/wrlinux-image-small-intel-x86-64.yaml` `appsdk - INFO: Input YAML File: exampleyamls/feature/package_management.yaml` `appsdk - INFO: Input YAML File: exampleyamls/feature/debug-tweaks.yaml` `appsdk - INFO: Include Default Packages: 0` `appsdk - INFO: Machine: intel-x86-64` `appsdk - INFO: Image Name: wrlinux-image-small` `appsdk - INFO: Image Type: ostree-repo ustart` `appsdk - INFO: Packages Number: 20` `appsdk - INFO: External Packages Number: 0` `appsdk - INFO: Package Feeds:` `https://distro.windriver.com/dist/wrlinux/lts-21/intel-x86-64/repos/rpm/corei7_64` `https://distro.windriver.com/dist/wrlinux/lts-21/intel-x86-64/repos/rpm/intel_x86_64` `https://distro.windriver.com/dist/wrlinux/lts-21/intel-x86-64/repos/rpm/noarch` `appsdk - INFO: enviroments: ['NO_RECOMMENDATIONS="0"', 'KERNEL_PARAMS="key=value"']` `appsdk - INFO: Save Yaml FIle to : /home/wruser/app-sdk/wrlinux-image-small-intel-x86-64.yaml`

特定のディレクトリにあるすべてのファイルから結合された *.yaml ファイルを作成します。	`$ appsdk genyaml exampleyamls/wrlinux-image-small-ite l-x86-64.yaml \. exampleyamls/feature/*.yaml` この例では、sdkDir **/exampleyamls/機能**ディレクトリで提供されるすべての ***.yaml**ファイルを使用します。

コマンドが完了すると、新しいファイルは、**-o**オプションで出力ディレクトリを指定しない限り、指定された元のイメージの***.yaml**ファイルと同じファイル名で作業ディレクトリに用意されます。

c. オプションで、結合された ***.yaml** ファイルを使用して、新しいイメージを作成します。

詳しくは、「システムイメージの作成（36ページ）」をご覧ください。

関連情報

input.yamlサンプルファイルを作成する (31ページ)
新しいSDKの作成とSDKの正常性の確認（33 ページ）
アプリケーションソースからのRPMパッケージの作成と公開（34 ページ）
システムイメージの作成（36 ページ）
コンテナイメージの作成（37 ページ）
OSTreeによるinitramfsイメージの作成（39 ページ）

3.8. Linuxアセンブリツールオプションリファレンス

利用可能なオプションを使用して、開発要件に合うようにイメージをカスタマイズしてください。

Linux Assembly Toolのオプション（引数）には、位置指定とオプションがあります。位置引数はツールで実現したいことの範囲を定義し、オプション引数は位置引数の具体的な使い方を指定するのに役立ちます。

ヘルプオプションの表示については、「Linux Assembly Toolのインストールと実行 (17ページ)」を参照してください。これらの

コマンドの使用例については、本ガイドの「一般的な使用例」を参照してください。

指定しない限り、すべての位置引数は **-h** (ヘルプ) オプションをサポートします。

一般的な使用法

```
appsdk option arg(s) positional_arg subcommand_arg arg_content
```

Optional Arguments

使用方法

```
appsdk optional arg(s)
```

Argument	概要
-d, **--debug**	ツール使用時の追加デバッグ出力を有効にします。
-h, **--help**	appsdkコマンドの使用法に関するヘルプを表示します。
-q, **--quiet**	エラーメッセージを除くすべての出力を非表示にする場合に使用します。
--log-dir logDirectory	ログレベルに関係なく、ログメッセージをlog.appsdkとして保存するディレクトリを指定するために使用します。

Positional Arguments

使用方法

```
appsdk positional_arg(s)
```

Argument	概要
checksdk	SDKのサニティチェックを実行するために使用します。この引数にはオプションの引数はありません。
exampleyamls	サンプルの ***.yaml** ファイルをデプロイするために使用します。
gencontainer	指定されたマシンのパッケージフィードからコンテナイメージを作成するために使用します。パッケージフィードは **input.yaml** ファイルで定義されます。このオプションは、systemd および関連するパッケージを除外し、**genimage** コマンドでシステムイメージを作成するために必要な推奨パッケージはインストールしません。 **gencontainer の**位置引数を実行すると、**container- base-intel-x86-64.container.README.md のような** image-name **-bspname .container.README.md** ファイルを作成し、イメージを起動し実行するための手順が記述されています。

Argument	概要
genimage	指定したマシンのパッケージフィードからイメージを作成するために使用します。パッケージフィードは **input.yaml** ファイルで定義します。 **genimage**の位置引数に以下の**--typeオプションを**指定して実行すると、イメージの起動と実行方法を説明したtype固有のREADMEファイルが作成されます。 • **--type ustart** は**wrlinux-image-small-intel- x86-64.ustart.README.mdを**作成します。 • **--type wic**は**wrlinux-image-small-intel- x86-64.wic.README.md**を作成します。
geninitramfs	指定されたマシンのパッケージフィードから initramfs イメージを作成するために使用します。パッケージフィードは **input.yaml** ファイルで定義されます。
genrpm	SDK の RPM パッケージをビルドするために使用します。
gensdk	新しいSDKを作成するために使用します。
genyaml	**input.yaml**ファイルを使用して***.yaml**ファイルを作成するために使用します。
publishrpm	SDK で作成した RPM をパッケージフィードリポジトリに公開するために使用します。

Subcommand Arguments

使用方法

```
appsdk positional_arg subcommand_arg arg_content
```

Positional Argument	Subcommand Argument(s)	Subcommand Argument 概要
exampleyamls	**-o**, **--outdir**	オプションで指定したディレクトリにサンプルの***.yaml**ファイルをデプロイするために使用します。指定しない場合、デフォルトのディレクトリは、作業ディレクトリsdkDir **/ exampleyamls**です。
gencontainer	**--env**	イメージを構築する前に、エクスポートする環境を追加で指定する場合に使用します。

Positional Argument	Subcommand Argument(s)	Subcommand Argument概要
	input.yaml	イメージ設定の基礎となる **input.yaml** ファイルを指定するために使用します。
	-n, **--name**	イメージ名を指定するために使用します。の**name** オプションにて**input.yaml**ファイルより優先されます。 。
	--no-clean	作業ディレクトリに作成されたルートファイルシステムをクリーンアップしないように指定するために使用します。これにより、ルートファイルシステムを表示することができ、イメージの問題をトラブルシューティングする際に役立ちます。
	--no-validate	**input.yaml**ファイルの入力パラメータを検証しないように指定する場合に使用します。
	-o, **--outdir**	イメージの出力先ディレクトリを指定します。デフォルトは現在の作業ディレクトリです。
	-p, **--pkg**	イメージに含める追加パッケージを指定します。
	-pkg-external	イメージにインストールする、追加の外部パッケージを指定します。
	--rootfs-post-script	ポストスクリプト、つまりイメージを更新するためのビルド後の指示を含むファイルを指定します。Wind River Linuxの**ROOTFS_POSTINSTALL_COMMAND**オプションと同様に、ビルドが完了したら（do_rootfs後）システム更新を行うために使用できます。
	--rootfs-pre-script	イメージを更新するためのプレスクリプト、またはビルド前の指示を含むファイルを指定します。Wind River Linuxの**ROOTFS_PREINSTALL_COMMAND**オプションと同様に、これを使用して、ビルドを実行する前にシステムの更新を行うことができます（pre do_rootfs）。
	-u, **--url**	パッケージフィードのURLやIPアドレスを追加で指定する場合に使用します。
	-w, **--workdir**	現在の作業ディレクトリと異なる場合、作業ディレクトリを指定するために使用します。
genimage	**--env**	イメージを構築する前に、エクスポートする環境を追加で指定する場合に使用します。
	-g, **--gpgpath**	GPGのホームディレクトリを指定するために使用します。オプション引数として設定すると、**input.yaml**ファイルの**gpg_path**オプションより優先されます。 指定しない場合は、**/tmp/.assembly-tool_gnupg** がデフォルトのディレクトリとなります。
	input.yaml	イメージ設定の基礎となる **input.yaml** ファイルを指定するために使用します。
	-n, **--name**	イメージ名を指定するために使用します。**name** オプションにて**input.yaml**ファイルより優先されます。
	--no-clean	作業ディレクトリに作成されたルートファイルシステムをクリーンアップしないように指定するために使用します。これにより、ルートファイルシステムを表示することができ、イメージの問題をトラブルシューティングする際に役立ちます。
	--no-validate	**input.yaml**ファイルの入力パラメータを検証しないように指定する場合に使用します。

Positional Argument	Subcommand Argument(s)	Subcommand Argument 概要
	-o, **--outdir**	イメージの出力先ディレクトリを指定します。デフォルトは現在の作業ディレクトリです。
	-p, **--pkg**	イメージに含める追加パッケージを指定します。
	-pkg-external	イメージにインストールする、追加の外部パッケージを指定します。
	--rootfs-post-script	ポストスクリプト、つまりイメージを更新するためのビルド後の指示を含むファイルを指定します。Wind River Linuxの**ROOTFS_POSTINSTALL_COMMAND**オプションと同様に、ビルドが完了したら（do_rootfs後）システム更新を行うために使用できます。
	--rootfs-pre-script	イメージを更新するためのプレスクリプト、またはビルド前の指示を含むファイルを指定します。Wind River Linuxの**ROOTFS_PREINSTALL_COMMAND**オプションと同様に、これを使用して、ビルドを実行する前にシステムの更新を行うことができます（pre do_rootfs）。
	-t, **--type**	以下のオプションから、イメージの種類を指定するのに使用します。 • all • wic • vmdk • vdi • ostree-repo • ustart **input.yaml**ファイルの**image_type**オプションより優先されます。指定しない場合、デフォルトは**ostree-repo**およびustartです。
	-u, **--url**	パッケージフィードのURLやIPアドレスを追加で指定する場合に使用します。
	-w, **--workdir**	現在の作業ディレクトリと異なる場合、作業ディレクトリを指定するために使用します。
geninitramfs	**--env**	イメージを構築する前に、エクスポートする環境を追加で指定する場合に使用します。
	-g, **--gpgpath**	GPGのホームディレクトリを指定するために使用します。オプション引数として設定すると、**input.yaml**ファイルの**gpg_path**オプションより優先されます。 指定しない場合は、**/tmp/.assembly-tool_gnupg** がデフォルトのディレクトリとなります。
	input.yaml	イメージ設定の基礎となる **input.yaml** ファイルを指定するために使用します。
	-n, **--name**	イメージ名を指定するために使用します。**name** オプションにて**input.yaml**ファイルより優先されます。
	--no-clean	作業ディレクトリに作成されたルートファイルシステムをクリーンアップしないように指定するために使用します。これにより、ルートファイルシステムを表示することができ、イメージの問題をトラブルシューティングする際に役立ちます。

	--no-validate	**input.yaml**ファイルの入力パラメータを検証しないように指定する場合に使用します。

Positional Argument	Subcommand Argument(s)	Subcommand Argument Description
	-o, **--outdir**	イメージの出力先ディレクトリを指定します。デフォルトは現在の作業ディレクトリです。
	-p, **--pkg**	イメージに含める追加パッケージを指定します。
	-pkg-external	イメージにインストールする、追加の外部パッケージを指定します。
	--rootfs-post-script	ポストスクリプト、つまりイメージを更新するためのビルド後の指示を含むファイルを指定します。Wind River Linuxの**ROOTFS_POSTINSTALL_COMMAND**オプションと同様に、ビルドが完了したら（do_rootfs後）システム更新を行うために使用できます。
	--rootfs-pre-script	イメージを更新するためのプレスクリプト、またはビルド前の指示を含むファイルを指定します。Wind River Linuxの**ROOTFS_PREINSTALL_COMMAND**オプションと同様に、これを使用して、ビルドを実行する前にシステムの更新を行うことができます（pre do_rootfs）。
	-u, **--url**	パッケージフィードのURLやIPアドレスを追加で指定する場合に使用します。
	-w, **--workdir**	現在の作業ディレクトリと異なる場合、作業ディレクトリを指定するために使用します。
genrpm	**-f**, **--file**	**input.yaml**ファイルを指定するときに使用します。
	-i, **--installdir**	アプリケーションのインストール先ディレクトリを指定します。
	-o, **--outputdir**	作成されたRPMパッケージの出力先ディレクトリを指定するのに使用します。
	-pkgarch	パッケージのアーキテクチャを指定するために使用します。
gensdk	**-f**, **--file**	**input.yaml**ファイルを指定するときに使用します。
	-o, **--outputdir**	作成されたRPMパッケージの出力先ディレクトリを指定するのに使用します。
publishrpm	**-r**, **--repo**	パッケージリポジトリのURLまたはIPアドレスを定義するために使用します。
genyaml	**--env**	イメージを構築する前に、エクスポートする環境を追加で指定する場合に使用します。
	-g, **--gpgpath**	GPGのホームディレクトリを指定するために使用します。オプション引数として設定すると、**input.yaml**ファイルの**gpg_path**オプションより優先されます。 指定しない場合は、**/tmp/.assembly-tool_gnupg** がデフォルトのディレクトリとなります。
	input.yaml	イメージ設定の基礎となる **input.yaml** ファイルを指定するために使用します。
	-n, **--name**	イメージ名を指定するために使用します。**name** オプションにて**input.yaml**ファイルより優先されます。
	--no-clean	作業ディレクトリに作成されたルートファイルシステムをクリーンアップしないように指定するために使用します。これにより、ルートファイルシステムを表示することができ、イメージの問題をトラブルシューティングする際に役立ちます。

Positional Argument	Subcommand Argument(s)	Subcommand Argument 概要
	--no-validate	**input.yaml**ファイルの入力パラメータを検証しないように指定する場合に使用します。
	-o, **--outdir**	イメージの出力先ディレクトリを指定します。デフォルトは現在の作業ディレクトリです。
	-p, **--pkg**	イメージに含める追加パッケージを指定します。
	-pkg-external	イメージにインストールする、追加の外部パッケージを指定します。
	--rootfs-post-script	ポストスクリプト、つまりイメージを更新するためのビルド後の指示を含むファイルを指定します。Wind River Linuxの**ROOTFS_POSTINSTALL_COMMAND**オプションと同様に、ビルドが完了したら（do_rootfs後）システム更新を行うために使用できます。
	--rootfs-pre-script	イメージを更新するためのプレスクリプト、またはビルド前の指示を含むファイルを指定します。Wind River Linuxの**ROOTFS_PREINSTALL_COMMAND**オプションと同様に、これを使用して、ビルドを実行する前にシステムの更新を行うことができます（pre do_rootfs）。
	-t, **--type**	以下のオプションから、イメージの種類を指定するのに使用します。 • all • container • initramfs • wic • vmdk • vdi • ostree-repo • ustart **input.yaml**ファイルの**image_type**オプションより優先されます。noneを指定した場合、デフォルトは**allです**。
	-u, **--url**	パッケージフィードのURLやIPアドレスを追加で指定する場合に使用します。
	-w, **--workdir**	現在の作業ディレクトリと異なる場合、作業ディレクトリを指定するために使用します。

3.9. Linux Assembly Tool Core input.yaml サンプルファイル

input.yaml ファイルは、イメージの多くの側面を定義するための詳細な情報を提供します。

注 **input.yamlの**ファイル名は参考程度にお考えください。拡張子が**.yamlで**あれば、用途に応じたファイル名を付けることができます。

最初にSDKをインストールしてセットアップするとき、参照用の***.yaml**ファイルは含まれていません。これを作成するには、ソースとなるSDKの作業ディレクトリで次のコマンドを実行します。

```
$ appsdk exampleyamls
appsdk - INFO: Deploy Directory: /opt/windriver/wrlinux-graphics/20.38/exampleyamls
+-------------------+-----------------------------------------+
|     Yaml Type     |                  Name                   |
+===================+=========================================+
| Image             | container-base-intel-x86-64.yaml        |
|                   |  core-image-minimal-intel-x86-64.yaml   |
|                   | core-image-sato-intel-x86-64.yaml       |
|                   | initramfs-ostree-image-intel-x86-64.yaml |
|                   |  wrlinux-image-small-intel-x86-64.yaml  |
|                   |                                         |
+-------------------+-----------------------------------------+
| Feature           | feature/debug-tweaks.yaml               |
|                   | feature/package_management.yaml         |
|                   | feature/set_root_password.yaml          |
|                   | feature/startup-container.yaml          |
|                   | feature/vboxguestdrivers.yaml           |
|                   | feature/xfce_desktop.yaml               |
|                   |                                         |
+-------------------+-----------------------------------------+
| System Definition | sysdef/add-system-user.yaml             |
|  Yamls            | sysdef/add-user-home.yaml               |
|                   | sysdef/contains-container-base.yaml     |
|                   | sysdef/set-dns.yaml                     |
|                   | sysdef/set-hostname.yaml                |
|                   | sysdef/set-ntp.yaml                     |
|                   | sysdef/update-containers.yaml           |
|                   |                                         |
+-------------------+-----------------------------------------+
```

これらのサンプルファイルを元に、独自のファイルを作成してください。

サンプルシステムイメージファイル

以下は、intel-x86-64 BSP用のシステムイメージ**input.yaml**ファイルの例です。

```
name: wrlinux-image-small
machine: intel-x86-64
image_type:
- ostree-repo
- ustart
package_feeds:
- http://XXXX/lat/dist/intel-x86-64/repos/rpm/corei7_64
- http://XXXX/lat/dist/intel-x86-64/repos/rpm/intel_x86_64
- http://XXXX/lat/dist/intel-x86-64/repos/rpm/noarch
ostree:
  OSTREE_CONSOLE: console=ttyS0,115200 console=tty1
  OSTREE_FDISK_BLM: 2506
  OSTREE_FDISK_BSZ: 200
  OSTREE_FDISK_FSZ: 32
  OSTREE_FDISK_RSZ: 1400
  OSTREE_FDISK_VSZ: 0
  OSTREE_GRUB_PW_FILE: $OECORE_NATIVE_SYSROOT/usr/share/bootfs/boot_keys/ostree_grub_pw
  OSTREE_GRUB_USER: root
  ostree_osname: wrlinux
  ostree_remote_url: ''
```

```
  ostree_skip_boot_diff: '2'
  ostree_use_ab: '1'
wic:
  OSTREE_FLUX_PART: fluxdata
  OSTREE_WKS_BOOT_SIZE: ''
  OSTREE_WKS_EFI_SIZE: --size=32M
  OSTREE_WKS_FLUX_SIZE: ''
  OSTREE_WKS_ROOT_SIZE: ''
remote_pkgdatadir: https://distro.windriver.com/release/wrlinux/linux-cd/base/WRLinux-CD-Images/in
tel-x86-64/repos/rpm
features:
  image_linguas: ''
  pkg_globs: null
gpg:
  gpg_path: /tmp/.lat_gnupg
  grub:
    BOOT_GPG_NAME: SecureBootCore
    BOOT_GPG_PASSPHRASE: SecureCore
    BOOT_KEYS_DIR: $OECORE_NATIVE_SYSROOT/usr/share/bootfs/boot_keys
  ostree:
    gpg_password: windriver
    gpgid: Wind-River-Linux-Sample
    gpgkey: $OECORE_NATIVE_SYSROOT/usr/share/genimage/rpm_keys/RPM-GPG-PRIVKEY-Wind-River-Linux-Sa
mple
packages:
- ca-certificates
- glib-networking
- grub-efi
- i2c-tools
- intel-microcode
- iucode-tool
- kernel-modules
- lmsensors
- os-release
- ostree
- ostree-upgrade-mgr
- packagegroup-busybox-replacement
- packagegroup-core-boot
- packagegroup-core-ssh-dropbear
- packagegroup-wr-bsps
- rtl8723bs-bt
- run-postinsts
- systemd
external-packages: []
include-default-packages: '0'
rootfs-pre-scripts:
- echo "run script before do_rootfs in $IMAGE_ROOTFS"
rootfs-post-scripts:
- echo "run script after do_rootfs in $IMAGE_ROOTFS"
environments:
- NO_RECOMMENDATIONS="0"
- KERNEL_PARAMS="key=value"
```

サンプルコンテナファイル

以下は、intel-x86-64 BSP 用のコンテナ **input.yaml** ファイルの例です。

```
name: container-base
machine: intel-x86-64
image_type:
- container
package_feeds:
- http://XXXX/lat/dist/intel-x86-64/repos/rpm/corei7_64
- http://XXXX/lat/dist/intel-x86-64/repos/rpm/intel_x86_64
- http://XXXX/lat/dist/intel-x86-64/repos/rpm/noarch
remote_pkgdatadir: https://distro.windriver.com/release/wrlinux/linux-cd/base/WRLinux-CD-Images/in
tel-x86-64/repos/rpm
features:
  image_linguas: ''
  pkg_globs: null
packages:
- base-files
- base-passwd
- ca-certificates
- dnf
- openssh
- os-release
- packagegroup-busybox-replacement
- rpm
- run-postinsts
- update-alternatives-opkg
external-packages: []
include-default-packages: '0'
rootfs-pre-scripts:
- echo "run script before do_rootfs in $IMAGE_ROOTFS"
rootfs-post-scripts:
- echo "run script after do_rootfs in $IMAGE_ROOTFS"
environments:
- NO_RECOMMENDATIONS="1"
container_oci:
  OCI_IMAGE_ARCH: x86-64
  OCI_IMAGE_AUTHOR: OpenEmbedded
  OCI_IMAGE_AUTHOR_EMAIL: oe.patch@oe
  OCI_IMAGE_ENTRYPOINT: ''
  OCI_IMAGE_ENTRYPOINT_ARGS: /bin/sh
  OCI_IMAGE_ENV_VARS: ''
  OCI_IMAGE_LABELS: ''
  OCI_IMAGE_PORTS: ''
  OCI_IMAGE_RUNTIME_UID: ''
  OCI_IMAGE_TAG: latest
  OCI_IMAGE_WORKINGDIR: ''
container_upload_cmd: '#skopeo copy --dest-tls-verify=false --insecure-policy docker-archive:examp
leyamls/deploy/container-base-intel-x86-64.docker-image.tar.bz2
  docker://URL :5000/container-base-intel-x86-64'
```

サンプルinitramfsファイル

以下は、intel-x86-64 BSP用のinitramfs **input.yaml**ファイルの例です。

```
name: initramfs-ostree-image
machine: intel-x86-64
image_type:
- initramfs

  package_feeds:
  - http://XXXX/lat/dist/intel-x86-64/repos/rpm/corei7_64
  - http://XXXX/lat/dist/intel-x86-64/repos/rpm/intel_x86_64
  - http://XXXX/lat/dist/intel-x86-64/repos/rpm/noarch
  remote_pkgdatadir: https://distro.windriver.com/release/wrlinux/linux-cd/base/WRLinux-CD-
  Images/in tel-x86-64/repos/rpm
  features:
    image_linguas:
    '' pkg_globs:
    null
  gpg:
    gpg_path:
    /tmp/.lat_gnupg
    grub:
      BOOT_GPG_NAME: SecureBootCore
      BOOT_GPG_PASSPHRASE: SecureCore
      BOOT_KEYS_DIR:
    $OECORE_NATIVE_SYSROOT/usr/share/bootfs/boot_keys ostree:
      gpg_password: windriver
      gpgid: Wind-River-Linux-Sample
      gpgkey:  $OECORE_NATIVE_SYSROOT/usr/share/genimage/rpm_keys/RPM-GPG-PRIVKEY-Wind-River-Linux-
      Sample packages:
  - base-passwd
  - bash
  - busybox
  - busybox-udhcpc
  - bzip2
  - ca-certificates
  - curl
  - dosfstools
  - e2fsprogs-e2fsck
  - e2fsprogs-resize2fs
  - e2fsprogs-tune2fs
  - findutils
  - gawk
  - glib-networking
  - gnupg
  - grep
  - gzip
  - initramfs-ostree
  - kbd
  - kmod
  - mdadm
  - mttyexec
  - ostree
  - ostree-switchroot
  - pv
  - rng-tools
  - sed
  - tar
  - udev
  - util-linux
  - util-linux-blkid
  - util-linux-blockdev
  - util-linux-fdisk
  - util-linux-fsck
  - util-linux-lsblk
  - util-linux-mount
```

```
- util-linux-setsid
external-packages: []
include-default-packages: '0'
rootfs-pre-scripts:
- echo "run script before do_rootfs in $IMAGE_ROOTFS"
rootfs-post-scripts:
- echo "run script after do_rootfs in $IMAGE_ROOTFS"
environments:
- NO_RECOMMENDATIONS="1"
```

新しいシステム機能を追加するためのサンプルfeatureファイル

以下は、**exampleyamls/feature/xfce_desktop.yaml** ファイルの一例です。これは、appsdk genimageコマンドで参照されるイメージ***.yaml**ファイルのpackagesセクションに追加される機能のリストであることに注意してください。

このファイルはappsdk genimageコマンドでのみ使用され、**gencontainer**または**geninitramfs**位置引数で使用するべきではありません。このような使い方をすると、Linux Assembly Toolの検証で、イメージ作成中にエラーが報告されます。

```
image_type:
- ostree-repo
- ustart
packages:
- lxdm
- packagegroup-xfce-base
- wr-themes
- gsettings-desktop-schemas
- packagegroup-core-x11-base
- packagegroup-core-x11-xserver
```

4. DISTROイメージのカスタマイズ

4.1. Linux Assembly Toolを使用してソースパッケージを追加する

Linux Assembly Toolをインストールすると、それを使ってソースパッケージを追加し、ネットワーク上のターゲットデバイスで利用できるようにすることができます。

このタスクについて

次の手順は、SDKで「Hello World」プログラムを作成し、Wind River Linux Distroバイナリ・リリースイメージを実行している外部ターゲットデバイスで使用するために、パッケージとして公開する手順を説明します。

始める前に

Linux Assembly Toolのインストールと実行（17ページ） で説明したように、SDKが事前にインストールされている必要があります。

手順

a. パッケージのソースをsdkDirにダウンロードし、アプリケーションをビルドします。

a. パッケージのソースをダウンロードします。

```
$ wget http://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
```

b. 中身を取り出す。

```
$ tar xzvf hello-2.10.tar.gz
```

c. アプリケーションディレクトリに移動します。

```
$ cd hello-2.10
```

d. ビルドフラグを設定します。

```
$ ./configure $CONFIGURE_FLAGS
```

e. アプリケーションのバイナリをビルドします。

```
$ make
```

f. インストールバイナリをビルドします。

```
$ make DESTDIR=/path-to /install-hello install
```

b. RPM パッケージを作成します。

a. **hello**パッケージの要素を定義する**hello.yaml**ファイルを作成します。

Linux Assembly Tool では、***.yaml** ファイルを使用して、さまざまな用途のパッケージとパッケージ関連情報を定義します。**input.yaml** ファイルのサンプルについては、「Linux Assembly Tool Core input.yaml サンプルファイル (48ページ)」を参照してください。

以下の内容を参考に、**hello.yaml** ファイルを作成してください。

```
name: hello
version: '2.10'
release: r0
summary: Hello World Program From Gnu
license: GPLv3

description: |
  A simple hello world program that only does one thing.
  It's from GNU.

dirs:
  - /usr
  - /usr/local
  - /usr/local/bin

files:
  - /usr/local/bin/hello

post_install: |
  #!/bin/sh
  echo "This is the post install script of hello program"
  echo "It only prints some message."
```

この例では、**dirs:** と **files:** セクションは、パッケージ化する特定のディレクトリを定義するのに役立ちます。例えば、次のステップのappsdk genrpmコマンドは、デフォルトですべてのファイルをパッケージ化しますが、***.yaml**ファイルで特定のディレクトリを設定し、そのディレクトリに含まれる特定のファイルをパッケージ化するために使用することができます。

b. Linux Assembly Toolでhelloパッケージを作成します。

```
$ appsdk genrpm -f hello.yaml -i /path-to /install-hello
```

c. SDKのパッケージ更新をシードするために、新しいHTTPサーバーをセットアップします。

SDK はビルド パッケージ リポジトリとは別に使用するように設計されているため、SDK パッケージに追加や変更があった場合は、これらの更新を他のターゲット デバイスで利用できるようにする機能が必要です。これを実現するために、SDK がインストールされている Linux ホスト システム上に HTTP サービスを作成します。

a. パッケージリポジトリ用のディレクトリを作成します。

```
$ mkdir -p /path-to /http_service_data/third_party_repo
```

b. サーバーを起動します。

```
$ python3 -m http.server 8888 --directory /path-to /http_service_data &
```

サーバーが起動すると、ブラウザで http://host-IP-address **:8888/third_party_repo** にアクセスできるようになります。

d. RPM パッケージを公開します。

```
$ appsdk publishrpm -r /path-to /http_service_data/third_party_repo
deploy/rpms/corei7_64/hello-2.10-r0.corei7_64.rpm
```

e. 展開されたターゲットデバイスでRPMパッケージを使用します。
デプロイされたWind River LinuxのDistroイメージ上で、ターミナルから以下のコマンドを実行します。

a. RPMリポジトリをセットアップします。

以下の内容をコピーして、ターゲットのターミナルに貼り付けてください。

```
# cat > /etc/yum.repos.d/test.repo <<EOF
[appsdk-test-repo]
name=appsdk test repo
baseurl=http://host-IP-address :8888/third_party_repo/
gpgcheck=0
EOFeck=0
EOF
```

b. ターゲットのパッケージキャッシュを更新します。

```
# dnf update
```

c. パッケージをインストールします。

```
# dnf install hello -y
Last metadata expiration check: 0:00:00 ago on Tue May 25 18:20:37 2020.
Dependencies resolved.
================================================================================
 Package       Arch          Version     Repository          Size
================================================================================
Installing:
 hello         corei7_64     2.10        appsdk-test-repo    7.5 k

Transaction Summary
================================================================================
Install  1 Package
```

d. helloアプリケーションを実行します。

```
# hello
Hello, world!
```

f. パッケージをイメージに含めることで、毎回別々にインストールする必要がありません。

a. 既存の ***.yaml** ファイルにパッケージを追加します。

これを行うには、**package_feeds** と **external-packages** セクションを更新し、オプションでファイルを新しい名前、例えば **image-with-hello.yaml** で保存します。次の例では、URL/IP アドレスとパッケージ名がどのように追加されているかに注目してください。

```
machine: intel-x86-64
name: custom-image
image_type:
- ostree-repo
- wic
- ustart
package_feeds:
- http://web-server-url :8080/lat/dist/intel-x86-64/repos/rpm/noarch/
- http://web-server-url :8080/lat/dist/intel-x86-64/repos/rpm/corei7_64/
- http://web-server-url :8080/lat/dist/intel-x86-64/repos/rpm/
intel_x86_64/
- http://host-IP-address :8888/third_party_repo
external-packages:
- hello
```

b.イメージを作成する。

```
$ appsdk genimage image-with-hello.yaml
```

そしてオプションで、helloアプリケーションを含む新しいSDKを作成します。

```
$ appsdk gensdk -f image-with-hello.yaml
```

4.2. システムイメージにコンテナを追加する

Linux Assembly Toolは、Dockerの機能を利用して起動時の動作を管理するコンテナをシステムイメージに含めるための方法を提供します。

このタスクについて

コンテナを含める方法は、主に2つあります。

- イメージの一部であるシステムイメージに、単一のコンテナイメージを構築する。
- **input.yaml** ファイルを使用して複数のコンテナイメージを指定し、インストール時にコンテナを起動および展開するコンテナイメージと場所を定義します。

この手順では、**gencontainer**位置引数でコンテナイメージを作成し、**genimage**位置引数でそのコンテナイメージを含むシステムイメージを作成します。このイメージを起動すると、コンテナは自動的に起動します。

さらに、上記の方法でビルドされていないコンテナを追加で含める方法を学びます。このためには、sdkDir **/exampleyamls/feature/startup-container.yaml** ファイルを基にした **input.yaml** ファイルを作成し、含めるコンテナとそのロードおよび起動に関する特定の指示を指定する必要があります。このサポートには以下が含まれます。

- docker loadコマンドで起動するDockerイメージ
- docker importコマンドを使用してシステムに追加する圧縮ファイルシステムイメージ(例: ***.tar.bz2)**
- Dockerレジストリからコンテナを取得するためにskopeo copyコマンドを使用したイメージ。

始める前に

- SDKがインストールされ、ソーシングされ、アプリケーションSDK管理ツールを実行できる必要があります。詳細については、「Linux Assembly Toolのインストールと実行 (17ページ)」を参照してください。
- ローカルディレクトリまたはリポジトリから外部コンテナを含める場合は、参照用にsdkDir **/exampleyamls/feature/startup-container.yaml**ファイルが必要です。詳細については、「 input.yamlサンプルファイルを作成する (31ページ)」を参照してください。

手順

コンテナを含むシステムイメージを作成する場合は、オプションを選択します。

コンテナオプション	実行するコマンド
単一コンテナイメージの作成と実行	a.コンテナイメージを構築する。 `$ appsdk gencontainer` 次のイメージは、典型的なコマンド出力を示しています。 `appsdk - INFO: Input YAML File: my custom-container.yaml` `appsdk - INFO: Machine: intel-x86-64` `appsdk - INFO: Image Name: container-base` `appsdk - INFO: Image Type: container` `appsdk - INFO: Packages Number: 10` `appsdk - INFO: External Packages Number: 0` `appsdk - INFO: Package Feeds:` `http://url/lat/dist/intel-x86-64/repos/rpm/corei7_64` `http://url/lat/dist/intel-x86-64/repos/rpm/intel_x86_64` `http://url/lat/dist/intel-x86-64/repos/rpm/noarch` `appsdk - INFO: enviroments: ['NO_RECOMMENDATIONS="1"']` `appsdk - INFO: Create Rootfs: Started` `appsdk - INFO: No Kernel Modules found, not running depmod` `appsdk - INFO: Create Rootfs: Succeeded(took 35 seconds)` `appsdk - INFO: Create Docker Container: Started` `appsdk - INFO: Create Docker Container: Succeeded(took 4 seconds)` `appsdk - INFO: Deploy Directory: /opt/windriver/wrlinux-graphics/20.38/deploy` `+-----------------------+-----------------------------------------------------+` `| Docker Image | container-base-intel-x86-64.docker-image.tar.bz2 -> |` `| | container-base-intel-x86-64-20201016031435.docker- |` `| | image.tar.bz2 |` `+-----------------------+-----------------------------------------------------+` `| OCI Image Rootfs | container-base-intel-x86-64.rootfs-oci |` `+-----------------------+-----------------------------------------------------+` `| Container Image Doc | container-base-intel-x86-64.container.README.md |` `+-----------------------+-----------------------------------------------------+` `| Yaml file for genimage | container-base-intel-x86-64.startup-container.yaml |` `| to load and run | |` `+-----------------------+-----------------------------------------------------+` 出力は、コンテナの構成情報を含む ***.yaml** ファイルの場所を提供していることに注意してください。このファイルは、次のステップでコンテナを参照するために使用されます。 b. 前のステップで作成した startup-container ***.yaml** ファイルを参照し、新しいシステムイメージを作成します。 `$ appsdk genimage deploy/container-base-intel-x86-64.startup-container.yaml` ビルドが完了すると、コンテナ付きのイメージがsdkDir **/deploy**ディレクトリに配置されます。 c. 前のステップで作成したイメージを展開し、rootでログインします。 詳しくは、Wind River Linux Distro Quick Start：Booting Up the Target System with QEMU 、またはBooting Up the Target System on Hardwareを参照してください。 d. Dockerでコンテナが正常に追加されることを確認します。 `# docker images` `REPOSITORY TAG IMAGE ID CREATED` `SIZE` `container-base-intel-x86-64 latest eb8fe4a1d485 27 minut` `es ago 129MB`

```
# docker ps
CONTAINER ID IMAGE COMMAND CREAT
ED STATUS PORTS NAMES
6e415728f98d container-base-intel-x86-64 "/bin/sh" 12 mi
nutes ago Up 12 minutes contains-base
```

e. コンテナを取り付けて、使い始めてください。

```
# docker attach container-base-intel-x86-64
sh-5.0# cat /etc/issue
Wind River Linux development 22.37 \n \l
```

コンテナオプション	実行するコマンド
複数のロングライフコンテナイメージのロードと実行	ロングライフコンテナに関する追加情報については、「ロングライフDockerコンテナへの対応の概要（6ページ）」を参照してください。

a. sdkDir **/deploy/container-base-arch .startup- container.yaml** をコピーして **my-container-startup.yaml** にリネームしてください。

b. **my-container-startup.yaml** ファイルの **rootfs-post-scripts** セクションを更新して、追加のコンテナを追加し、その Docker 起動オプションを指定します。

Linuxホストシステムまたはオンラインの場所にあるコンテナについては、**system: file**タグを使用して、コンテナの場所（**src**）、宛先（**dst**）、およびファイル**モード**を指定する必要があります。

次の例では、システムイメージに追加する3つのコンテナを指定し、コンテナファイルの場所とイメージへの追加方法を異なる例にしています。

```
image_type:
- ostree-repo
- ustart
packages:
- startup-container
- docker
rootfs-post-scripts:
- echo "contains-base load=/var/docker-images/container-baseintel-
x86-64.docker-image.tar.bz2 image=container-base-intelx86-
64" >> $IMAGE_ROOTFS/etc/sysdef/run_on_upgrade.d/contain
ers.dat
- echo "ubuntu-tar load=/var/docker-images/ubuntu.docker-imag
e.tar.bz2" >> $IMAGE_ROOTFS/etc/sysdef/run_on_upgrade.d/conta
iners.dat
- skopeo copy --src-tls-verify=false --insecure-policy docker
://path_to_repo /ubuntu docker-archive:$IMAGE_ROOTFS/var/dockerimages/
ubuntu.docker-image.tar.bz2:ubuntu-tar
system:
- run_on_upgrade:
   - exampleyamls/sysdef/run_on_upgrade.d/10_update_containers
.sh
- run_always:
   - exampleyamls/sysdef/run_always.d/10_start_containers.sh
- contains:
   - exampleyamls/container-base-intel-x86-64.yaml
- files:
   - file:
       src: deploy/container-base-intel-x86-64.docker-image.ta
r.bz2
       dst: /var/docker-images/
```

コンテナオプション	実行するコマンド
	c. **my-container-startup.yaml を**参照し、新しいシステムイメージを作成します。 ファイルを作成します。 `$ appsdk genimage deploy/my-container-startup.yaml` ビルドが完了すると、コンテナ付きのイメージがsdkDir **/deploy**ディレクトリに配置されます。 d. 前のステップで作成したイメージを展開し、rootでログインします。 詳しくは、Wind River Linux Distro Quick Startを参照してください。QEMUを使用したターゲットシステムの起動またはハードウェア上でのターゲットシステムの起動.を参照してください。 e. Dockerでコンテナが正常に追加されることを確認します。 `# docker images` `REPOSITORY TAG IMAGE ID CREATED` `SIZE` `container-base-intel-x86-64 latest eb8fe4a1d485 4 hours` `ago 129MB` `ubuntu-tar latest a2a15febcdf3 14 month` `s ago 64.2MB` `# docker ps` `CONTAINER ID IMAGE COMMAND CREATE` `D STATUS PORTS NAMES` `caf39a0ecd0b ubuntu-tar "/bin/bash" 2 minu` `tes ago Up 2 minutes ubuntu-tar` `6e415728f98d container-base-intel-x86-64 "/bin/sh" 12 minu` `tes ago Up 12 minutes contains-base` f. コンテナを使用するには、コンテナ名を指定してdocker attachコマンドを使用します。 使用する各コンテナについて、この手順を実行します。次の例では、container-ubuntu コンテナをアタッチしています。 `# docker attach container-ubuntu` `sh-5.0# cat /etc/issue` `Ubuntu 20.04.3 LTS \n \l`

4.3. ユーザーの管理およびカスタマイズ

サンプルの ***.yaml** ファイルを使用して、Wind River Linux Distro システムイメージにユーザーを追加することができます。

このタスクについて

システムイメージにユーザーを追加するには、sdkDir **/exampleyamls/sysdef/add-user-home.yaml** および **add-system-user.yaml** ファイルを参照し、これらの更新ファイルを appsdk genimage コマンドでインクルードしてください。

この2つのファイルは、**/home**ディレクトリの有無にかかわらず、システムユーザーを追加する柔軟性を提供します。デフォルトでは、**/home** ディレクトリの名前はユーザー名になります。

始める前に

- Linux Assembly Toolのインストールと実行 (17ページ)で説明したように、SDKが事前にインストールされている必要があります。
- 参照用に、サンプルの **add-system-user.yaml** および/または **add-user-home.yaml** ファイルを用意する必要があり ます。詳細は、「input.yamlサンプルファイルを作成する (31ページ)」を参照してください。

手順

a. Linuxの開発用ホストで、各ユーザーのパスワード暗号化をSHA-512アルゴリズムで作成します。

この操作は、追加するユーザーごとに行う必要があります。

```
$ mkpasswd -m sha-512 -S `pwgen 12 1` '$3Cre1t@'
$6$Cae2apae2voa$1u9ykdcQ83w8TDM3dsGcZQiwfPrNXWrCwwWpPQc6CUD/eW1ed0Mt9v4NK4sFNhVn7xqL7pYXgPVRvqL
9.h4dz.
```

この情報を、将来のステップで使用するために保存してください。

b. SDKを使用できるように環境を整える。

sdkDirに移動して、環境設定スクリプトを実行します。

```
$ . ./environment-setup-*-wrs-linux
```

c. ユーザー固有の ***.yaml** ファイルの**コピーを作成します。**

例えば、ユーザーがシステムイメージ上でホームディレクトリを必要とする場合、**add-user-home.yaml** ファイルのコピーを作成します。ユーザーがメンテナンスなどの目的でシステム・アクセスのみを必要とする場合、**add-system-user.yaml のコピーを作成します**。

d. 前のステップで作成した ***.yaml** ファイルを更新して、Username と Password のエントリを独自のものに置き換えます。

e. 上記の手順 2 と 3 を繰り返して、必要に応じてユーザーを追加してください。

f. 各ユーザーの ***.yaml** ファイルについて、sdkDir **/exampleyamls/sysdef/run_once.d/20_add_user_home.sh** のコピーを作成するか、または **10_add_system_user.sh** スクリプトは、追加したいユーザーの種類によって異なります。

g. 前の手順で作成した各スクリプトについて、対応するユーザーの ***.yaml** ファイルを更新して、***.yaml** ファイルの **-files** セクションにそのスクリプトを含めます。

たとえば、sdkDir **/exampleyamls/sysdef/my-user-home1.yaml**ファイルを作成し、sdkDir **/exampleyamls/sysdef/run_once.d/20_my_user_home1.sh**スクリプトを作成してユーザー**jdoeを**追加すると、**my-user-home1.yaml**ファイルを編集してこのスクリプトを追加します.

```
image_type:
- ostree-repo
- ustart
packages:
- shadow
- base-passwd
- base-files
- sudo
system:
- run_once:
  - exampleyamls/sysdef/run_once.d/20_my_user_home1.sh
- files:
  - file:
      src: exampleyamls/sysdef/files/sudoers_sudo
      dst: /etc/sudoers.d/sudo
      mode: 644
```

h. スクリプト内のuseraddコマンドを、上記手順1のユーザー名とパスワードに一致するように更新します。

```
useradd jdoe -G sudo --password '$6$Cae2apae2voa$1u9ykdcQ83w8TDM3dsGcZQiwfPrNXWrCwwWpPQc6CUD/eW
1ed0Mt9v4NK4sFNhVn7xqL7pYXgPVRvqL9.h4dz.'
```

この例では、ユーザーjdoeは**$3Cre1t@という**パスワードのハッシュが設定されています。

i. オプションで、ユーザーの ***.yaml** ファイルとシステムイメージの ***.yaml** ファイルを結合します。

この手順では、システムイメージの ***.yaml** ファイルにユーザー設定を追加して、イメージ作成を容易にします。この手順を実行しない場合、appsdk genimageコマンドを使用してシステムイメージを作成するときに、各別のユーザー***.yaml**ファイルを指定する必要があります。詳細については、「既存のファイルから結合されたinput.yamlファイルを作成する（40ページ）」を参照してください。

```
$ appsdk genyaml exampleyamls/wrlinux-image-small-intel-x86-64.yaml
\ exampleyamls/sysdef/my-user-home1.yaml exampleyamls/sysdef/my-system-
user1.yaml
```

コマンドが完了すると、作業ディレクトリにあるイメージファイル名にユーザーの更新情報が含まれるようになります。

j. オプションで、結合された ***.yaml** ファイルを使用して、新しいイメージを作成します。

詳しくは、「システムイメージの作成（36 ページ）」をご覧ください。

4.4. カスタムGPGセキュリティキーの使用

Wind River Linux Distro SDKは、イメージとパッケージのセキュリティを確保するためにGPGセキュリティキーを提供しますが、それらを独自のキーに置き換えることができます。

このタスクについて

Wind River Linux Distro イメージでは、SDK の **sdkDir/sysroots/x86_64-wrlinuxsdk-linux/usr/share/ bootfs/boot_keys** ディレクトリにある GPG キーが、署名済み GRUB パスワードを使用してデバイスが正常に起動するために必要なセキュリティを提供し ます。これらのキーは、例えばイメージの ***.yaml** ファイルの gpg セクションで定義されます。

```
gpg:
  gpg_path: /tmp/.lat_gnupg
  grub:
    BOOT_GPG_NAME: SecureBootCore
    BOOT_GPG_PASSPHRASE: SecureCore
    BOOT_KEYS_DIR: $OECORE_NATIVE_SYSROOT/usr/share/bootfs/boot_keys
  ostree:
    gpg_password: windriver
    gpgid: Wind-River-Linux-Sample
    gpgkey: $OECORE_NATIVE_SYSROOT/usr/share/genimage/rpm_keys/RPM-GPG-PRIVKEY-Wind-River-Linux-Sa
mple
```

ディレクトリ内の対応するgrubファイルには、以下のものがあります。

```
$ tree sysroots/x86_64-wrlinuxsdk-linux/usr/share/bootfs/boot_keys/
sysroots/x86_64-wrlinuxsdk-linux/usr/share/bootfs/boot_keys/
├── boot_cfg_pw
├── BOOT-GPG-KEY-SecureBootCore
├── BOOT-GPG-PRIVKEY-SecureBootCore
├── boot_pub_key
└── ostree_grub_pw
```

対応するパッケージ関連ファイルは、sdkDir /sysroots/x86_64-wrlinuxsdk-linux/usr/share/genimage/rpm_keys/ ディレクトリに存在し、以下を含みます。

```
$ tree sysroots/x86_64-wrlinuxsdk-linux/usr/share/genimage/rpm_keys/
sysroots/x86_64-wrlinuxsdk-linux/usr/share/genimage/rpm_keys/
└── RPM-GPG-PRIVKEY-Wind-River-Linux-Sample
```

独自のGPG鍵を使用するには、これらのディレクトリのファイルを独自のものに置き換え、システムイメージを作成する必要があります。

4.5. ネットワークベースのインストールにWi-Fiサポートを追加する

インストールされたシステムイメージをWi-Fi経由でインストール、アップデートするには、ハードウェアに必要なファームウェアとWi-Fiカーネルモジュールをイメージの設定に含める必要があります。

このタスクについて

デフォルトでは、Linux Assembly Tool で作成したインストール可能なシステムイメージには Wi-Fi サポートが含まれておらず、代わりに QEMU と Linux ホストシステムを組み合わせた固有のネットワーキングが使用されます。インターネットにアクセスするために Wi-Fi を必要とする環境にいる場合、またはインストールしたデバイスが Wi-Fi アクセスを必要とするネットワークベースのインストールを必要とする場合は、この手順を使用して Wi-Fi サポートを追加します。

ウインドリバーは、bcm-2xxx-rpi4およびintel-x86-64 BSPをサポートするために必要なWi-Fiパッケージを提供します。その他のBSPについては、必要なパッケージとファームウェアを提供する必要があります。BSPに必要なものの詳細については、ボードのREADMEファイルを参照してください。

この手順では、Linux Assembly Tool を使用して、インストール可能な initramfs ベースのシステムイメージに Wi-Fi カーネルモジュールと Linux ファームウェアをインストールし、ネットワークとの通信を容易にする事前共有キー (WPA-PSK) とパスワードを含む Wi-Fi 保護アクセス (WPA) セキュリティプロトコルをセットアップして、イメージ上の Wi-Fi クライアントを設定します。完了したら、「Performing a Network-Based Installation Over Wi-Fi (24 ページ)」の説明に従って、Wi-Fi 経由でネットワークベースのインストールを実行するためにイメージを配置することができます。

始める前に

- Linux Assembly Toolのインストールと実行 (17ページ)で説明したように、SDKが事前にインストールされている必要があります。
- bcm-2xxx-rpi4とintel-x86-64以外のBSPのアクセスを追加する場合、必要なパッケージとファームウェアが必要です。
- 接続するネットワークのWi-Fi SSID、事前共有キー（WPA PSK）、WPAパスワードのいずれかが必要です。
- OSTree パッケージリポジトリの IP アドレスが必要です。詳細は、「Distroイメージによるベースコンテナ、SDK、パッケージフィードの構築（23 ページ）」をご覧ください。

手順

a. サンプルの ***.yaml** ファイルを作成します。詳細については、「input.yamlサンプルファイルを作成する (31ページ)」を参照してください。

   完了すると、sdkDir **/ exampleyamls/feature** ディレクトリに、イメージの設定に必要な **set-wifi-eap.yaml** と **set-wifi-psk.yaml** ファイルが用意されます。

b. ファームウェアとカーネルモジュールを含めるために、特定の Wi-Fi ***.yaml** ファイルの **packages** セクションを更新します。

   注　bcm-2xxx-rpi4 および intel-x86-64 ハードウェアでは、このステップは必要ありません。

   たとえば、**set-wifi-eap.yaml** および **set-wifi-psk.yaml パッケージ**セクションには、いずれも intel- x86-64 BSP 用の次のパッケージが含まれています。

```
packages:
- packagegroup-base-wifi
- linux-firmware-iwlwifi-6000g2a-6
- linux-firmware-iwlwifi-6000g2b-6
- linux-firmware-iwlwifi-135-6
- linux-firmware-iwlwifi-8000c
- linux-firmware-iwlwifi-8265
- linux-firmware-iwlwifi-7265d
- linux-firmware-iwlwifi-9000
- linux-firmware-iwlwifi-misc
- kernel-module-iwlmvm
- kernel-module-iwlwifi
```

BSP では、これらのエントリを削除し、**set-wifi-eap.yaml** および **set-wifi-psk.yaml** ファイルの **packages** セクションを更新して、お使いのハードウェアに固有のファームウェアおよびカーネルモジュールパッケージを含めるようにします。

```
packages:
- packagegroup-base-wifi
- your-bsp-wifi-firmware
- your-bsp-wifi-kernel-module
-
```

これは例であり、あなたのBSPはハードウェアの要件をサポートするために追加のファイルを必要とするかもしれないことに注意してください。

c. ネットワークのWi-Fi保護アクセスを設定する。

a. wi-fi protected access pre-shared key（WPA PSK）を設定する。

set-wifi-psk.yamlファイルの**ネットワーク**エントリを更新して、Wi-FiネットワークのSSIDとPSKを含めます。

```
network={
 ssid="YOUR_SSID"
  psk="YOUR_PSK"
}
```

b.wi-fi protected access extensible authentication protocol（WPA EAP）を設定します。

set-wifi-eap.yamlファイルの**ネットワーク**エントリを更新して、Wi-FiネットワークのSSIDとパスワードを追加します。

```
network={
 ssid="YOUR_SSID"
  key_mgmt=WPA-EAP
  identity="YOUR_ID"
  password="YOUR_PASSWORD "
}
```

d. **install-over-wifi-eap.yaml** および/または **install-over-wifi-psk.yaml** ファイルを更新し、OSTree パッケージリポジトリの場所を含めるようにします。

これにより、イメージはインストールプロセスの一部として OSTree パッケージリポジトリにアクセスできるようになります。Python を使用したローカルパッケージリポジトリの作成については、「Distroイメージを使用したベースコンテナ、SDK、およびパッケージフィードの構築（23ページ）」を参照してください。 sdkDirから以下のコマンドを実行し、両ファイルを更新してください。

```
$ sed -i -e "s#YOUR_OSTREE_REMOTE_URL#http://your-host-ip:8888/ostree_repo#g"
\ exampleyamls/feature/install-over-wifi-psk.yaml
\ exampleyamls/feature/install-over-wifi-eap.yaml
```

これでWi-Fiのイメージ設定は完了です。

e. イメージを作成する。

イメージの作成に使用する設定の種類は、ネットワークの最終用途の要件と、デバイスが提供する Wi-Fi サポートの種類に依存します。例えば、次の表では、インストールされたイメージは配備されたハードウェアに使用されるイメージを指し、インストールイメージはインストールを実行するイメージを指します。

image typeを作成するために	コマンドを実行
WPA EAPをサポートするイメージをインストール	$ appsdk --log-dir log genimageexampleyamls/feature/set-wi fi-eap.yaml
WPA PSKをサポートするイメージをインストール	$ appsdk --log-dir log genimage exampleyamls/feature/set-w ifi-psk.yaml
WPA PSKでインストール	$ appsdk --log-dir log genimage exampleyamls/feature/insta ll-over-wifi-psk.yaml
WPA EAPでインストール	$ appsdk --log-dir log genimage exampleyamls/feature/insta ll-over-wifi-eap.yaml
WPA PSKでインストールし、WPA PSKをサポートしたイメージをインストール	$ appsdk --log-dir log genimage exampleyamls/feature/insta ll-over-wifi-psk.yaml \ exampleyamls/feature/set-wifi-psk.yaml
WPA EAPでインストールし、WPA EAPをサポートしたイメージをインストール	$ appsdk --log-dir log genimage exampleyamls/feature/insta ll-over-wifi-eap.yaml \ exampleyamls/feature/set-wifi-eap.yaml

イメージの作成が完了したら、Wi-Fiとネットワーク設定をテストするために展開することができます。詳細については、Wi-Fiを利用したネットワークベースのインストールを実行する（30ページ）を参照してください。