# Red Hat Software Collections 3

## パッケージガイド

Red Hat Enterprise Linux の Software Collections をパッケージ化するガイド

Last Updated: 2023-08-24

# Red Hat Software Collections 3 パッケージガイド

Red Hat Enterprise Linux の Software Collections をパッケージ化するガイド

Petr Kovář
Red Hat Customer Content Services
pkovar@redhat.com

## 法律上の通知



## 概要

パッケージングガイドでは、Software Collections の概要と、構築およびパッケージ化の方法を説明します。RPM パッケージを使用するソフトウェアパッケージの基本を理解する開発者およびシステム管理者は、Software Collections の概念を初めて理解している開発者やシステム管理者は、Software Collections を使い始めることができます。

# 目次

# 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、弊社の CTO、 Chris Wright のメッセージ を参照してください。

# 第1章 SOFTWARE COLLECTIONS の概要

本章では、Software Collections または SCL の概念および使用方法を紹介します。

## 1.1. RPM を使用したパッケージソフトウェアの理由

RPM Package Manager (RPM) は、Red Hat Enterprise Linux で実行できるパッケージ管理システムです。RPM を使用すると、Red Hat Enterprise Linux 用に作成したソフトウェアの配布、管理、および更新が容易になります。多くのソフトウェアベンダーは、従来のアーカイブファイル (tarball など) でソフトウェアを配布します。ただし、RPM パッケージにソフトウェアをパッケージ化すると、いくつかの利点があります。これらの利点は以下のとおりです。

**RPM を使用すると、以下が可能になります。**

**パッケージのインストール、再インストール、削除、アップグレード、および検証。**

ユーザーは、標準のパッケージ管理ツール (**Yum** や **PackageKit** など) を使用して、RPM パッケージのインストール、再インストール、削除、アップグレード、および検証を行うことができます。

**インストール済みのパッケージのデータベースを使用した、パッケージのクエリーおよび検証。**

RPM はインストールしたパッケージとそのファイルのデータベースを維持するので、ユーザーは、システムのパッケージを簡単にクエリーし、検証することができます。

**メタデータを使用して、パッケージ、それらのインストール手順などを記述します。**

各 RPM パッケージには、パッケージのコンポーネント、バージョン、リリース、サイズ、プロジェクト URL、インストール手順などを記述するメタデータが含まれています。

**ソースパッケージとバイナリーパッケージへの元のソフトウェアソースのパッケージ化**

RPM を使用すると、元のソフトウェアのソースを取得して、ユーザーのためにソースパッケージやバイナリーパッケージにパッケージ化することができます。ソースパッケージでは、使用しているパッチと、完全なビルド命令とともに、元のソースを使用できます。この設計により、新しいバージョンのソフトウェアがリリースされると、パッケージのメンテナンスが簡単になります。

**Yum リポジトリーへのパッケージの追加。**

**Yum** リポジトリーにパッケージを追加すると、クライアントがソフトウェアを見つけ、デプロイできるようになります。

**パッケージへのデジタル署名**

GPG 署名鍵を使用すると、パッケージにデジタル署名し、ユーザーがパッケージの信頼性を検証できるようになります。

RPM の概要と使用方法は、Red Hat Enterprise Linux 7 システム管理者のガイド または Red Hat Enterprise Linux 6 デプロイメントガイド を参照してください。

## 1.2. SOFTWARE COLLECTIONS とは

Software Collections を使用すると、システムに同じソフトウェアコンポーネントの複数のバージョンを構築し、同時にインストールできます。Software Collections は、従来の RPM パッケージ管理ユーティリティーによってインストールされるパッケージのシステムバージョンには影響を与えません。

**Software Collections**

**システムファイルを上書きしない**

Software Collections は、複数のコンポーネントセットとして配布され、システムファイルを上書きせずに完全な機能を提供します。

**システムファイルとの競合を避けるように設計されています。**

Software Collections は、単一の Software Collection とベースシステムインストールの競合を避けるために、特別なファイルシステム階層を利用します。

**RPM パッケージマネージャーへの変更を必要としません。**

Software Collections では、ホストシステムに存在する RPM パッケージマネージャーを変更する必要はありません。

**spec ファイルへのマイナーな変更のみが必要**

従来のパッケージを単一の Software Collection に変換するには、パッケージ spec ファイルにマイナーな変更を加えるだけで済みます。

**1 つのスペックファイルで従来のパッケージと Software Collection パッケージを構築できるようにします。**

従来のパッケージと Software Collection パッケージの両方を構築できます。

**含まれるすべてのパッケージの一意に名前を付ける**

Software Collection の名前空間では、Software Collection に含まれるすべてのパッケージが一意に名前が付けられます。

**更新されたパッケージと競合しないようにしてください。**

Software Collection の名前空間により、システムでパッケージを更新すると競合が発生することはありません。

**他の Software Collections に依存**

1 つの Software Collection は他のソフトウェアコレクションに依存する可能性があるため、複数の依存関係レベルを定義できます。

## 1.3. SOFTWARE COLLECTIONS のサポートの有効化

Software Collections を有効にしてビルドできるように、システムで Software Collections のサポートを有効にするには、scl-utils パッケージおよび scl-utils-build パッケージをインストールする必要があります。

scl-utils パッケージおよび scl-utils-build パッケージと、システムにインストールされていない場合は、シェルプロンプトで root 権限で次のコマンドを実行します。

```
# yum install scl-utils scl-utils-build
```

この scl-utils パッケージは、システムで Software Collections を有効にできる **scl** ツールを提供します。Software Collections の有効化に関する詳細は、「ソフトウェアコレクションの有効化」を参照してください。

この scl-utils-build パッケージは、Software Collections の構築に不可欠のマクロを提供します。Software Collections の構築に関する詳細は、「Software Collection の構築」を参照してください。

**重要**

Red Hat Enterprise Linux システムで利用可能なサブスクリプションによっては、**Optional** チャンネルを有効にして scl-utils-build パッケージをインストールする必要がある場合があります。

## 1.4. ソフトウェアコレクションのインストール

Software Collection がシステムにあることを確認するには、Software Collection のメタパッケージをインストールします。Software Collections は RPM Package Manager と完全に互換性があるため、このタスクには **Yum** や **PackageKit** などの従来のツールを使用できます。

たとえば、**software_collection_1** という名前のメタパッケージを使用して Software Collection をインストールするには、以下のコマンドを実行します。

```
# yum install software_collection_1
```

このコマンドは、ユーザーが Software Collection で最も一般的なタスクを実行するために必要な、Software Collection 内のすべてのパッケージを自動的にインストールします。

Software Collections を使用すると、使用するパッケージのサブセットのみをインストールできます。たとえば、rh-ruby23 Software Collection から Ruby インタープリターを使用するには、その Software Collection から rh-ruby23-ruby パッケージをインストールする必要があります。

Software Collection に依存するアプリケーションをインストールすると、その Software Collection はアプリケーションの残りの依存関係とともにインストールされます。

Software Collection のメタパッケージの詳細は、「メタパッケージ」 を参照してください。

**Yum** および **PackageKit** の使用に関する詳細は、Red Hat Enterprise Linux 7 システム管理者のガイド または Red Hat Enterprise Linux 6 デプロイメントガイド を参照してください。

## 1.5. インストールされた SOFTWARE COLLECTIONS の一覧表示

システムにインストールされている Software Collections の一覧を取得するには、以下のコマンドを実行します。

```
scl --list
```

指定した Software Collection に含まれるインストール済みパッケージの一覧を取得するには、以下のコマンドを実行します。

```
scl --list software_collection_1
```

## 1.6. ソフトウェアコレクションの有効化

**scl** ツールは、Software Collection を有効にし、Software Collection 環境でアプリケーションを実行するために使用されます。

**scl** ツールの一般的な使用方法は、以下の構文を使用して記述できます。

```
scl action software_collection_1 software_collection_2 command
```

***command*** を複数の引数で実行している場合は、コマンドとその引数を引用符で囲むようにしてください。

```
scl action software_collection_1 software_collection_2 'command --argument'
```

または、**--** コマンド区切り文字を使用して、複数の引数を指定して ***command*** を実行します。

```
scl action software_collection_1 software_collection_2 -- command --argument
```

**以下の点に留意してください。**

- **scl** ツールを実行すると、現在のシェルの子プロセス (subshell) を作成します。コマンドを再度実行すると、サブシェルのサブシェルが作成されます。
- 現在のサブシェルの有効な Software Collections を一覧表示できます。詳細は、「有効な Software Collections の一覧表示」 を参照してください。
- 最初に有効な Software Collection を無効にして、再度有効にする必要があります。Software Collection を無効にするには、Software Collections を有効にする際に作成したサブシェルを終了します。
- **scl** ツールを使用して Software Collection を有効にする場合は、一度に有効な Software Collection で 1 つのアクションのみを実行できます。別のアクションを実行する前に、有効な Software Collection を無効にする必要があります。

### 1.6.1. アプリケーションの直接的な実行

たとえば、**software_collection_1** という名前の Software Collection で、 **--versions** オプションで **Perl** を直接実行するには、次のコマンドを実行します。

```
scl enable software_collection_1 'perl --version'
```

または、Software Collection 環境でコマンドを実行することをより便利にする sypaths サブパッケージを指定します。sypaths サブパッケージの詳細は、「sypaths サブパッケージの提供」 を参照してください。

### 1.6.2. 複数の Software Collections が有効になっているシェルの実行

複数の Software Collections が有効になっている環境で **Bash** シェルを実行するには、次のコマンドを実行します。

```
scl enable software_collection_1 software_collection_2 bash
```

上記のコマンドは、**software_collection_1** および **software_collection_2** という名前の 2 つの Software Collections を有効にします。

### 1.6.3. ファイルに保存されたコマンドの実行

ファイルに保存されたコマンドを多数実行するには、Software Collection 環境内で以下のコマンドを実行します。

```
cat cmd | scl enable software_collection_1 -
```

上記のコマンドは、**software_collection_1** という名前の Software Collection の環境にある **cmd** ファイルに保存されているコマンドを実行します。

## 1.7. 有効な SOFTWARE COLLECTIONS の一覧表示

現行セッションで有効になっている Software Collections の一覧を取得するには、以下のコマンドを実行して **$X_SCLS** 環境変数を出力します。

```
echo $X_SCLS
```

## 1.8. SOFTWARE COLLECTIONS のアンインストール

Software Collections は RPM Package Manager と完全に互換性があるため、Software Collection をアンインストールするときに **Yum** や **PackageKit** などの従来のツールを使用できます。たとえば、**software_collection_1** という名前の Software Collection に含まれるパッケージおよびサブパッケージをすべてアンインストールするには、以下のコマンドを実行します。

```
yum remove software_collection_1\*
```

**yum remove** コマンドを使用して、**scl** ユーティリティーを削除することもできます。

**Yum** および **PackageKit** の使用に関する詳細は、Red Hat Enterprise Linux 7 システム管理者のガイド または Red Hat Enterprise Linux 6 デプロイメントガイド を参照してください。

# 第2章 SOFTWARE COLLECTIONS のパッケージ化

本章では、Software Collections のパッケージ化について説明します。

## 2.1. 独自のソフトウェアコレクションの作成

通常、以下の 2 つの方法のいずれかを使用して、既存の Software Collection に依存するアプリケーションをデプロイすることができます。

- 必要な Software Collections およびパッケージをすべて手動でインストールしてから、アプリケーションをデプロイする、または
- アプリケーションの新しい Software Collection を作成します。

**アプリケーション用に新しい Software Collection を作成する場合:**

**Software Collection メタパッケージの作成**

各 Software Collection にはメタパッケージが含まれており、ユーザーは Software Collection で最も一般的なタスクを実行するために必要な Software Collection パッケージのサブセットをインストールします。メタパッケージの作成に関する詳細は、「メタパッケージ」 を参照してください。

**Software Collection の root ディレクトリーの場所を指定することを検討する**

Software Collection の spec ファイルに **%_scl_prefix** マクロを設定して、Software Collection の root ディレクトリーの場所を指定することが推奨されます。詳細は、「Software Collection Root ディレクトリー」 を参照してください。

**Software Collection パッケージの名前の接頭辞付けを検討する**

Software Collection パッケージの名前の前には、ベンダーおよび Software Collection の名前を付けることが推奨されます。詳細は、「Software Collection 接頭辞」 を参照してください。

**アプリケーションに必要なすべての Software Collections およびその他のパッケージを依存関係として指定します。**

アプリケーションに必要なすべての Software Collections およびその他のパッケージが、Software Collection の依存関係として指定されていることを確認します。詳細は、「Software Collection を別の Software Collection に依存させる」 を参照してください。

**既存の従来のパッケージを変換または新しい Software Collection パッケージの作成**

Software Collection パッケージの spec ファイルのすべてのマクロが条件を使用していることを確認します。既存パッケージの spec ファイルを変換する方法は、「従来の spec ファイルの変換」 を参照してください。

**Software Collection の構築**

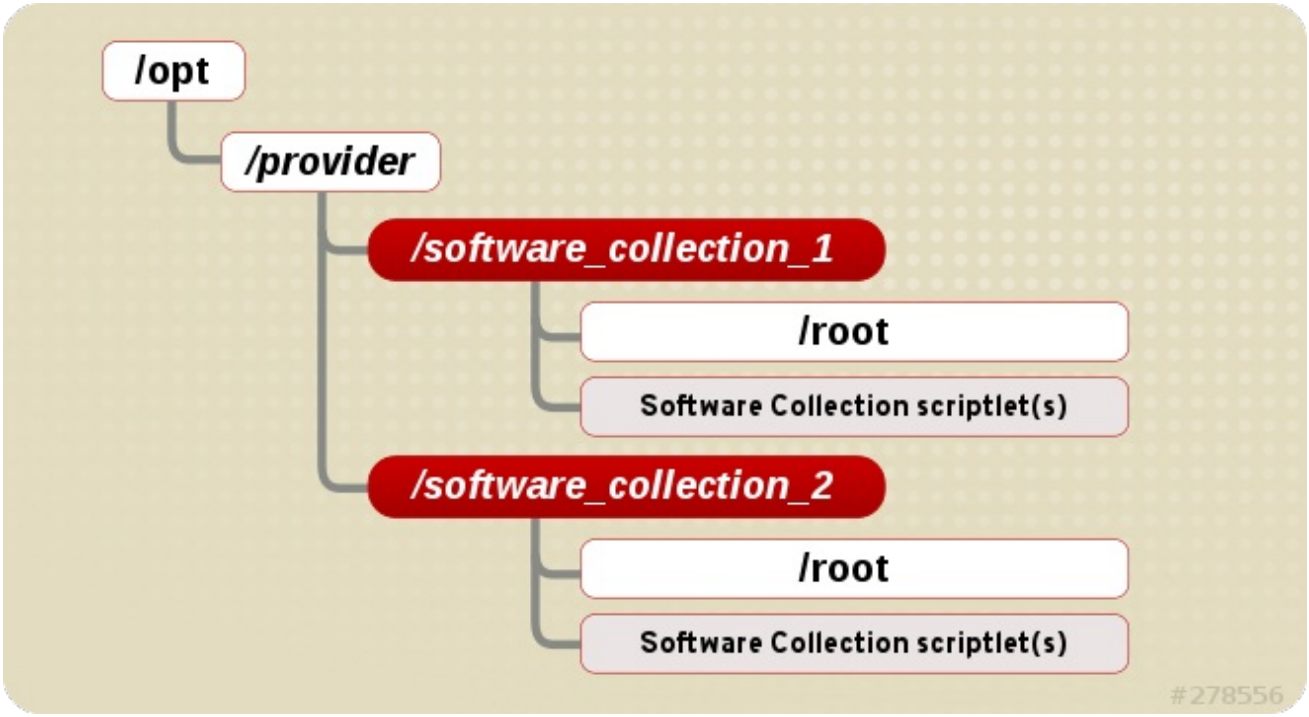
Software Collection のメタパッケージを作成し、Software Collection のパッケージを変換または作成したら、**rpmbuild** ユーティリティーを使用して Software Collection を構築できます。詳細は、「Software Collection の構築」 を参照してください。

## 2.2. ファイルシステム階層

Software Collections と ベースシステムインストールの競合を避けるため、Software Collections のルートディレクトリーは通常 **/opt/** にあります。**/opt/** ディレクトリーの使用は、ファイルシステム階層標準 (FHS) によって推奨されます。

以下は、*software_collection_1* と *software_collection_2* の 2 つの Software Collections を持つファイルシステム階層レイアウトの例です。

**図2.1 Software Collection File System 階層**




上記のように、Software Collections の各ディレクトリーには、Software Collection の root ディレクトリーと、1 つ以上の Software Collection スクリプトレットが含まれます。Software Collection スクリプトレットの詳細は、「Software Collection スクリプトレット」を参照してください。

## 2.3. SOFTWARE COLLECTION ROOT ディレクトリー

以下の例のように、spec ファイルに **%_scl_prefix** マクロを設定して、root ディレクトリーの場所を変更できます。

```
%global _scl_prefix /opt/provider
```

ここで、*provider* は登録されているプロバイダー (ベンダー) の名前で、該当する場合は Linux Foundation と下位の Linux LANANA (Assigned Names and Numbers Authority) で、ファイルシステム階層規格に準拠します。

Software Collections を構築して配布する各組織またはプロジェクトは、独自のプロバイダー名を使用する必要があります。これは、ファイルシステム階層標準 (FHS) に準拠し、Software Collections とベースシステムインストールの競合の発生を防ぎます。

ファイルシステム階層は、以下のレイアウトに適合させることが推奨されます。

```
/opt/provider/prefix-application-version/
```

**注記**

spec ファイルの **_scl_prefix** マクロの上に **%scl_package** マクロを定義する必要があります。

ファイルシステム階層標準の詳細は、http://www.pathname.com/fhs/ を参照してください。

Linux の割り当て名および番号機関の詳細は、http://www.lanana.org/ を参照してください。

## 2.4. SOFTWARE COLLECTION 接頭辞

Software Collection に名前を付けるときは、Software Collection の一部であるパッケージのシステムバージョンと名前が競合する可能性を回避するために、以下に説明するように Software Collection の名前に接頭辞を付けることをお勧めします。

Software Collection の接頭辞は、以下の 2 つの部分で設定されます。

- *provider* の名前を定義するプロバイダーの部分
- Software Collection 自体の名前

Software Collection の接頭辞のこれらの 2 つは、以下の例のようにハイフン (**-**) で区切られています。

```
myorganization-ruby193
```

この例では、*myorganization* がプロバイダーの名前で、*ruby193* は Software Collection の名前になります。

最終的にベンダーとディストリビューターの決定は、プロバイダーの名前を接頭辞に指定するかどうかに関わらず、指定することが強く推奨されます。

主な例外は、Red Hat Software Collections 1.x に最初に同梱された Software Collections であり、接頭辞にはプロバイダーの名前を指定しません。Red Hat Software Collections 2.0 以降に追加された新しい Software Collections は **rh** をプロバイダーの名前として使用します。以下に例を示します。

```
rh-ruby23
```

## 2.5. SOFTWARE COLLECTION パッケージ名

Software Collection パッケージ名は、以下の 2 つの部分で設定されます。

- *接頭* 辞の部分 (「Software Collection 接頭辞」 を参照)
- Software Collection に含まれるアプリケーションの名前とバージョン番号。

Software Collection パッケージ名のこれらの 2 つの部分は、以下の例のようにハイフン (**-**) で区切ります。

```
myorganization-ruby193-foreman-1.1
```

この例では、*myorganization-ruby193* が接頭辞で、*foreman-1.1* はアプリケーションの名前とバージョン番号です。

## 2.6. SOFTWARE COLLECTION スクリプトレット

Software Collection スクリプトレットは、現在のシステム環境を変更する簡単なシェルスクリプトで、Software Collection のパッケージグループが、システムにインストールされている従来のパッケージのグループよりも優先されます。

Software Collection スクリプトレットを使用するには、scl-utils パッケージに含まれる **scl** ツールを使用します。**scl** の詳細は、「ソフトウェアコレクションの有効化」 を参照してください。

1 つの Software Collection には、複数の Software Collection スクリプトレットを含めることができます。これらのスクリプトレットは、Software Collection パッケージの **/opt/provider/software_collection/** ディレクトリーにあります。Software Collection に 1 つのスクリプトレットのみを配布する必要がある場合には、そのスクリプトレットの名前として **enable** を使用することが強く推奨されます。ユーザーが **scl enable *software_collection コマンド*** を実行して Software Collection 環境でコマンドを実行すると、**/opt/provider/software_collection/enable** スクリプトレットは検索パスの更新などに使用されます。

Software Collection スクリプトレットは、**scl-enable** コマンドを実行して作成されたサブシェルにシステム環境のみを設定できることに注意してください。サブシェルは、コマンドが実行される期間にのみアクティブです。

## 2.7. パッケージレイアウト

各ソフトウェアコレクションのレイアウトは、他のパッケージのサブセットをインストールするメタパッケージと、Software Collection の名前空間内にインストールされる多数の Software Collection のパッケージで設定されます。

### 2.7.1. メタパッケージ

各 Software Collection にはメタパッケージが含まれており、ユーザーは Software Collection で最も一般的なタスクを実行するために必要な Software Collection パッケージのサブセットをインストールします。たとえば、必須パッケージは Perl 言語インタープリターを提供しますが、Perl 拡張モジュールはありません。メタパッケージには、基本的なファイルシステム階層が含まれており、Software Collection のスクリプトレットを多数提供します。

メタパッケージの目的は、Software Collection の必須パッケージをすべて適切にインストールし、Software Collection を有効にすることができるようにすることです。

メタパッケージは、Software Collection の一部でもある次のパッケージを生成します。

**メインパッケージ**: *%name*

Software Collection のメインパッケージには、Software Collection に含まれるベースパッケージの依存関係が含まれています。main パッケージにはファイルが含まれません。

Software Collection のパッケージの依存関係を指定する場合は、Software Collection の他のパッケージがメインパッケージに依存していないことを確認してください。メインパッケージの目的は、ユーザーが Software Collection で最も一般的なタスクを実行するために必要なパッケージのみをインストールすることです。

通常、メインパッケージはビルド時の依存関係 (たとえば、別の Software Collection のパッケージのビルド時間依存関係のみのパッケージ) を指定しないようにします。

たとえば、Software Collection の名前が **myorganization-ruby193** の場合、メインパッケージマクロは以下に展開されます。

```
myorganization-ruby193
```

**ランタイムサブパッケージ: *%name*-runtime**

Software Collection の runtime サブパッケージは Software Collection のファイルシステムを所有し、Software Collection のスクリプトレットを配布します。ユーザーが Software Collection を使用できるようにするには、このパッケージをインストールする必要があります。

たとえば、Software Collection の名前が **myorganization-ruby193** の場合、ランタイムパッケージマクロは以下に展開されます。

```
myorganization-ruby193-runtime
```

**ビルドサブパッケージ: *%name*-build**

Software Collection の build サブパッケージは、Software Collection のビルド設定を提供します。Software Collection にパッケージを構築するのに必要な RPM マクロが含まれます。build サブパッケージは任意で、Software Collection から除外できます。

たとえば、Software Collection の名前が **myorganization-ruby193** の場合、ビルドサブパッケージマクロは以下に展開されます。

```
myorganization-ruby193-build
```

**myorganization-ruby193-build** サブパッケージの内容を以下に示します。

```
$ cat /etc/rpm/macros.ruby193-config
%scl myorganization-ruby193
```

**syspaths サブパッケージ: *%name*-syspaths**

Software Collection の syspaths サブパッケージは、便利なシェルラッパーとシンボリックリンクを標準パスにインストールして、ベースシステムのインストールを変更するオプションの方法を提供します。ただし、Software Collection パッケージでバイナリーファイルをより簡単に作成できます。

たとえば、Software Collection の名前が **myorganization-ruby193** の場合、syspaths パッケージマクロは以下に展開されます。

```
myorganization-ruby193-syspaths
```

syspaths サブパッケージの詳細は、「syspaths サブパッケージの提供」を参照してください。

**scldevel サブパッケージ: *%name*-scldevel**

%name Software Collection の scldevel サブパッケージには開発ファイルが含まれています。このファイルは、%name Software Collection に依存する別の Software Collection のパッケージを開発する際に便利です。scldevel サブパッケージは任意で、%name Software Collection から除外できます。

たとえば、Software Collection の名前が **myorganization-ruby193** の場合、scldevel サブパッケージマクロは以下に展開されます。

```
myorganization-ruby193-scldevel
```

scldevel サブパッケージの詳細は、「scldevel サブパッケージの提供」 を参照してください。

### 2.7.2. メタパッケージの作成

**新しいメタパッケージを作成する場合は、以下を行います。**

- **%scl_package** マクロの上に、メタパッケージ仕様ファイルの先頭に以下のマクロを定義します。
  - **scl_name_prefix** Software Collection 名の接頭辞として使用するプロバイダーの名前を指定します (例: *myorganization-*)。これは **_scl_prefix** とは異なるもので、Software Collection の root を指定しますが、プロバイダーの名前も使用します。詳細は、「Software Collection 接頭辞」 を参照してください。
  - **scl_name_base** Software Collection のベース名を指定します (例: *ruby*)。
  - **scl_name_version** Software Collection のバージョンを指定します (例: *193*)。
- 設定ファイルと状態ファイルの場所を変更して、NFS で Software Collection を使用できるようにする Software Collection マクロを定義する **nfsmountable** ことが推奨されます。詳細は、「NFS での Software Collections の使用」 を参照してください。
- Software Collection のランタイムに不可欠なすべてのパッケージをメタパッケージの依存関係として指定することを検討してください。これにより、Software Collection メタパッケージでパッケージがインストールされているようになります。
- build サブパッケージに **Requires: scl-utils-build** を追加することが推奨されます。
- メタパッケージで、Software Collection 固有のマクロに条件を使用する必要はありません。
- Software Collection のパッケージが **enable** スクリプトレットで必要となる可能性があるパスの再定義を含めます。

  一般的に使用されるパスの再定義に関する情報は、「一般的に使用されるパスの再定義」 を参照してください。
- メタパッケージに **%prep** セクションに **%setup** マクロが含まれていることを確認してください。そうしないと、Software Collection の構築に失敗します。**%setup** マクロで特定のオプションを使用する必要がない場合は、**%setup -c -T** コマンドを **%prep** セクションに追加します。

  これは、**%setup** マクロが **%buildsubdir** ディレクトリーを定義し、作成するためです。これは通常、ビルド時に一時ファイルを保存するために使用されます。Software Collection パッケージで **%setup** を定義しない場合は、**%buildsubdir** ディレクトリー内のファイルは上書きされ、ビルドは失敗します。
- build サブパッケージの **macros.%{scl}-config** ファイルに使用するマクロを追加します。

**メタパッケージの例**

*myorganization-ruby193* と呼ばれる Software Collection の典型的なメタパッケージを理解するには、以下の例を参照してください。

```
%global scl_name_prefix myorganization-
%global scl_name_base ruby
%global scl_name_version 193
```

```
%global scl %{scl_name_prefix}%{scl_name_base}%{scl_name_version}

# Optional but recommended: define nfsmountable
%global nfsmountable 1

%global _scl_prefix /opt/myorganization
%scl_package %scl

Summary: Package that installs %scl
Name: %scl_name
Version: 1
Release: 1%{?dist}
License: GPLv2+
Requires: %{scl_prefix}less
BuildRequires: scl-utils-build

%description
This is the main package for %scl Software Collection.

%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils

%description runtime
Package shipping essential scripts to work with %scl Software Collection.

%package build
Summary: Package shipping basic build configuration
Requires: scl-utils-build

%description build
Package shipping essential configuration macros to build %scl Software Collection.

# This is only needed when you want to provide an optional scldevel subpackage
%package scldevel
Summary: Package shipping development files for %scl

%description scldevel
Package shipping development files, especially useful for development of
packages depending on %scl Software Collection.

%prep
%setup -c -T

%install
%scl_install

cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export PATH="%{_bindir}:%{_sbindir}\${PATH:+:\${PATH}}"
export LD_LIBRARY_PATH="%{_libdir}\${LD_LIBRARY_PATH:+:\${LD_LIBRARY_PATH}}"
export MANPATH="%{_mandir}:\${MANPATH:-}"
export PKG_CONFIG_PATH="%
{_libdir}/pkgconfig\${PKG_CONFIG_PATH:+:\${PKG_CONFIG_PATH}}"
EOF
```

```
# This is only needed when you want to provide an optional scldevel subpackage
cat >> %{buildroot}%{_root_sysconfdir}/rpm/macros.%{scl_name_base}-scldevel << EOF
%%scl_%{scl_name_base} %{scl}
%%scl_prefix_%{scl_name_base} %{scl_prefix}
EOF

# Install the generated man page
mkdir -p %{buildroot}%{_mandir}/man7/
install -p -m 644 %{scl_name}.7 %{buildroot}%{_mandir}/man7/

%files

%files runtime -f filelist
%scl_files

%files build
%{_root_sysconfdir}/rpm/macros.%{scl}-config

%files scldevel
%{_root_sysconfdir}/rpm/macros.%{scl_name_base}-scldevel

%changelog
* Fri Aug 30 2013 John Doe &lt;jdoe@example.com&gt; 1-1
- Initial package
```

## 2.8. SOFTWARE COLLECTION マクロ

Software Collection パッケージマクロ **scl** は、Software Collection のファイル構造の再配置先を定義します。移動したファイル構造は、Software Collection 専用で使用されるファイルシステムです。

**%scl_package** マクロは、Software Collection のメタパッケージの所有者を定義し、Software Collection 環境で使用する追加のパッケージマクロを提供します。

従来のパッケージと Software Collection パッケージを 1 つのスペックファイルで構築できるようにするには、以下の例のように **%{?scl:*macro*}** の説得時を Software Collection マクロに付けます。

```
%{?scl:Requires: %scl_runtime}
```

上記の例では、**%scl_runtime** マクロは **Requires** タグの値です。マクロとタグの両方が **%{?scl:** 接頭辞を使用します。

### 2.8.1. Software Collection に固有のマクロ

以下の表は、特定の Software Collection に固有のマクロの一覧を示しています。すべてのマクロにはデフォルト値があり、ほとんどの場合で変更する必要はありません。

**表2.1 Software Collection 固有のマクロ**

| マクロ | 詳細 | 値の例 |
|---|---|---|
| **%scl_name** | Software Collection の名前 | **software_collection_1** |

| マクロ | 詳細 | 値の例 |
|---|---|---|
| **%scl_prefix** | 末尾にダッシュが追加された Software Collection の名前 | **software_collection_1-** |
| **%pkg_name** | 元のパッケージの名前 | **perl** |
| **%_scl_prefix** | Software Collection の root (パッケージの root ではない) | **/opt/provider/** |
| **%_scl_scripts** | Software Collection のスクリプトレットの場所 | **/opt/provider/software_collection_1/** |
| **%_scl_root** | パッケージのインストール root (install-root) | **/opt/provider/software_collection_1/root/** |
| **%scl_require_package software_collection_1 package_2** | 特定の Software Collection の特定のパッケージに依存します。 | **software_collection_1-package_2** |

### 2.8.2. Software Collection に指定されていないマクロ

以下の表は、特定の Software Collection に固有しないマクロの一覧を示しています。これらのマクロは移動されず、Software Collection ファイルシステムを参照しないため、システムの root ファイルシステムを参照できます。これらのマクロは、**_root** を接頭辞として使用します。

すべてのマクロにはデフォルト値があり、ほとんどの場合で変更する必要はありません。

**表2.2 非 Software Collection 固有のマクロ**

| マクロ | 詳細 | 再配置 | 値の例 |
|---|---|---|---|
| **%_root_prefix** | Software Collection の **%_prefix** マクロ | いいえ | **/usr/** |
| **%_root_exec_prefix** | Software Collection の **%_exec_prefix** マクロ | いいえ | **/usr/** |
| **%_root_bindir** | Software Collection の **%_bindir** マクロ | いいえ | **/usr/bin/** |
| **%_root_sbindir** | Software Collection の **%_sbindir** マクロ | いいえ | **/usr/sbin/** |
| **%_root_datadir** | Software Collection の **%_datadir** マクロ | いいえ | **/usr/share/** |
| **%_root_sysconfdir** | Software Collection の **%_sysconfdir** マクロ | いいえ | **/etc/** |

| マクロ | 詳細 | 再配置 | 値の例 |
|---|---|---|---|
| **%_root_libexecdir** | Software Collection の **%_libexecdir** マクロ | いいえ | **/usr/libexec/** |
| **%_root_sharedstatedir** | Software Collection の **%_sharedstatedir** マクロ | いいえ | **/usr/com/** |
| **%_root_localstatedir** | Software Collection の **%_localstatedir** マクロ | いいえ | **/usr/var/** |
| **%_root_includedir** | Software Collection の **%_includedir** マクロ | いいえ | **/usr/include/** |
| **%_root_infodir** | Software Collection の **%_infodir** マクロ | いいえ | **/usr/share/info/** |
| **%_root_mandir** | Software Collection の **%_mandir** マクロ | いいえ | **/usr/share/man/** |
| **%_root_initddir** | Software Collection の **%_initddir** マクロ | いいえ | **/etc/rc.d/init.d/** |
| %_**root_libdir** | Software Collection の **%_libdir** マクロ (Software Collection のメタパッケージがプラットフォームに依存していない場合、このマクロは機能しません) | いいえ | **/usr/lib/** |

### 2.8.3. nfsmountable マクロ

Software Collection マクロ **nfsmountable** を使用すると、**_sysconfdir**、**_sharedstatedir**、および **_localstatedir** マクロの値を変更できるため、Software Collection が、状態ファイルと設定ファイルが、Software Collection の **/opt** ファイルシステム階層外に置かれます。これにより、NFS で Software Collection を使用する場合に、ファイルの管理が容易になり、必要になります。

NFS を介した Software Collections のサポートが必要ない場合は、**nfsmountable** の使用は任意ですが推奨されます。詳細は、「NFS での Software Collections の使用」 を参照してください。

## 2.9. 一般的に使用されるパスの再定義

本セクションでは、Software Collection 環境を設定するために **enable** スクリプトレットのパスを再定義するのに一般的に使用される環境変数の一覧を紹介します。また、Software Collection のファイルシステム階層内の Software Collection コンポーネントの場所を指定するためにも使用されます。

**enable** スクリプトレットでパスの再定義を指定する必要があるかどうかは、Software Collection に含まれるパッケージにより異なります。通常、環境変数は以下のパターンに従います。

```
$ENV_VAR=$SCL_ENV_VAR:$ENV_VAR
```

### 2.9.1. 言語固有のパスの再定義

#### GEM_PATH

**GEM_PATH** 環境変数で Ruby gems の場所を指定します。したがって、rh-ruby23 Software Collection を拡張する Software Collections でも使用されます。詳細は、「rh-ruby23 Software Collection の拡張」 を参照してください。

**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export GEM_PATH="\${GEM_PATH:=%{gem_dir}:\`scl enable %{scl_ruby} -- ruby -e "print Gem.path.join(':')"\`}"
```

#### GOPATH

**GOPATH** 環境変数で、Go ソースおよびバイナリーファイルの場所を指定します。**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export GOPATH="%{gopath}\${GOPATH:+:\${GOPATH}}"
```

#### JAVACONFDIRS

**JAVACONFDIRS** 環境変数は、**java.conf** 設定ファイルの場所を指定するために使用されます。**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export JAVACONFDIRS="%{_sysconfdir}/java\${JAVACONFDIRS:+:}\${JAVACONFDIRS:-}"
```

#### PERL5LIB

**PERL5LIB** 環境変数は、**%{?_scl_root}** 接頭辞でインストールできるように、カスタム Perl モジュールの場所を指定するのに使用されます。**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export PERL5LIB="%{_scl_root}%{perl_vendorlib}\${PERL5LIB:+:\${PERL5LIB}}"
```

#### PYTHONPATH

**PYTHONPATH** 環境変数で、カスタム Python ライブラリーの場所を指定します。**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export PYTHONPATH="%{_scl_root}%{python_sitearch}:%{_scl_root}%{python_sitelib}\${PYTHONPATH:+:}\${PYTHONPATH:-}"
```

### 2.9.2. 他のパスの再定義

#### CPATH

**CPATH** 環境変数は、GCC コンパイラーが使用するパスを指定します。**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export CPATH="%{_includedir}\${CPATH:+:\${CPATH}}"
```

#### INFOPATH

**INFOPATH** 環境変数は、Info ファイルが含まれるディレクトリーを指定します。**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export INFOPATH="%{_infodir}\${INFOPATH:+:\${INFOPATH}}"
```

LD_LIBRARY_PATH

**LD_LIBRARY_PATH** 環境変数で、ライブラリーの場所を指定します。詳細は、「Software Collection Library のサポート」 を参照してください。

**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export LD_LIBRARY_PATH="%{_libdir}\${LD_LIBRARY_PATH:+:\${LD_LIBRARY_PATH}}"
```

LIBRARY_PATH

**LIBRARY_PATH** 環境変数は、GCC が使用する特別なリンカーファイルまたは通常のライブラリーの場所を指定します。**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export LIBRARY_PATH="%{_libdir}\${LIBRARY_PATH:+:\${LIBRARY_PATH}}"
```

MANPATH

**MANPATH** 環境変数で man ページの場所を指定します。詳細は、「Software Collection MANPATH サポート」 を参照してください。

**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export MANPATH="%{_mandir}:\${MANPATH:-}"
```

PATH

**PATH** 環境変数で、バイナリーファイルの場所を指定します。**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export PATH="%{_bindir}:%{_sbindir}\${PATH:+:\${PATH}}"
```

PCP_DIR

**PCP_DIR** 環境変数は、PCP が使用するファイルおよびディレクトリーの場所を指定します。**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export PCP_DIR="%{_scl_root}"
```

PKG_CONFIG_PATH

**PKG_CONFIG_PATH** 環境変数は、**pkg-config** プログラムによって使用される **.pc** ファイルの場所を指定します。詳細は、「Software Collection .pc ファイルのサポート」 を参照してください。

**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export PKG_CONFIG_PATH="%
{_libdir}/pkgconfig\${PKG_CONFIG_PATH:+:\${PKG_CONFIG_PATH}}"
```

XDG_CONFIG_DIRS

**XDG_CONFIG_DIRS** 環境変数は、freedesktop.org 仕様に従ってデスクトップ設定ファイルの場所を指定します。**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export XDG_CONFIG_DIRS="%{_sysconfdir}/xdg:\${XDG_CONFIG_DIRS:-/etc/xdg}"
```

XDG_DATA_DIRS

**XDG_DATA_DIRS** 環境変数は、freedesktop.org 仕様に従ってデスクトップデータファイルの場所を指定します。Software Collections によっては、Software Collections 固有のスクリプトを検索したり、bash の完了を有効にしたりするために使用されます。

**enable** スクリプトレットに以下を追加して、環境変数を再定義します。

```
export XDG_DATA_DIRS="%{_datadir}:\${XDG_DATA_DIRS:-/usr/local/share:%{_root_datadir}}"
```

## 2.10. 従来の SPEC ファイルの変換

本項では、変換した spec ファイルが、従来のパッケージと Software Collection の両方で使用できるようにするため、従来の spec ファイルを Software Collection の spec ファイルに変換する方法を説明します。

### 2.10.1. 変換スペックファイルの例

従来の spec ファイルと変換された spec ファイルの比較は、以下の例を参照してください。

```
--- a/less.spec
+++ b/less.spec
@@ -1,10 +1,14 @@
+%{?scl:%global _scl_prefix /opt/provider}
+%{?scl:%scl_package less}
+%{!?scl:%global pkg_name %{name}}
+
 Summary: A text file browser similar to more, but better
-Name: less
+Name: %{?scl_prefix}less
 Version: 444
 Release: 7%{?dist}
 License: GPLv3+
 Group: Applications/Text
-Source: http://www.greenwoodsoftware.com/less/%{name}-%{version}.tar.gz
+Source: http://www.greenwoodsoftware.com/less/%{pkg_name}-%{version}.tar.gz
 Source1: lesspipe.sh
 Source2: less.sh
 Source3: less.csh
@@ -19,6 +22,7 @@ URL: http://www.greenwoodsoftware.com/less/
 Requires: groff
 BuildRequires: ncurses-devel
 BuildRequires: autoconf automake libtool
-Obsoletes: lesspipe < 1.0
+Obsoletes: %{?scl_prefix}lesspipe < 1.0
+%{?scl:Requires: %scl_runtime}

 %description
```

```
 The less utility is a text file browser that resembles more, but has
@@ -31,7 +35,7 @@ You should install less because it is a basic utility for viewing text
 files, and you'll use it frequently.

 %prep
-%setup -q
+%setup -q -n %{pkg_name}-%{version}
 %patch1 -p1 -b .Foption
 %patch2 -p1 -b .search
 %patch4 -p1 -b .time
@@ -51,16 +55,16 @@ make CC="gcc $RPM_OPT_FLAGS -D_GNU_SOURCE -
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOU
 %install
 rm -rf $RPM_BUILD_ROOT
 make DESTDIR=$RPM_BUILD_ROOT install
-mkdir -p $RPM_BUILD_ROOT/etc/profile.d
+mkdir -p $RPM_BUILD_ROOT%{_sysconfdir}/profile.d
 install -p -c -m 755 %{SOURCE1} $RPM_BUILD_ROOT/%{_bindir}
-install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT/etc/profile.d
-install -p -c -m 644 %{SOURCE3} $RPM_BUILD_ROOT/etc/profile.d
-ls -la $RPM_BUILD_ROOT/etc/profile.d
+install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{_sysconfdir}/profile.d
+install -p -c -m 644 %{SOURCE3} $RPM_BUILD_ROOT%{_sysconfdir}/profile.d
+ls -la $RPM_BUILD_ROOT%{_sysconfdir}/profile.d

 %files
 %defattr(-,root,root,-)
 %doc LICENSE
-/etc/profile.d/*
+%{_sysconfdir}/profile.d/*
 %{_bindir}/*
 %{_mandir}/man1/*
```

### 2.10.2. タグおよびマクロ定義の変換

以下の手順では、従来のスペックファイルのタグおよびマクロ定義を Software Collection の spec ファイルに変換する方法を説明します。

**手順2.1 タグおよびマクロ定義の変換**

1. **%_scl_prefix** の上に **%scl_package macro** マクロを定義することで、root ディレクトリーの場所を変更できます。

   ```
   %{?scl:%global _scl_prefix /opt/provider}
   ```

2. **%scl_package** マクロを spec ファイルに追加します。次のように、スペックファイルのプリアンブルの前にマクロを配置します。

   ```
   %{?scl:%scl_package package_name}
   ```

3. Software Collection 用にパッケージを構築していない場合は、spec ファイルのプリアンブルで **%pkg_name** マクロを定義することが推奨されます。

   ```
   %{!?scl:%global pkg_name %{name}}
   ```

したがって、**%pkg_name** マクロを使用して、従来のパッケージと Software Collection の両方を構築するのに使用できるスペックファイルで、パッケージの元の名前を定義することができます。

4. spec ファイルの Preamble で **Name** タグを以下のように変更します。

```
Name: %{?scl_prefix}package_name
```

5. 他の Software Collection パッケージを構築したり、リンクしたりする場合は、以下のように **Requires** タグおよび **BuildRequires** タグにある パッケージの名前に **%{?scl_prefix}** を付けます。

```
Requires: %{?scl_prefix}ifconfig
```

パッケージのシステムバージョンによっては、バージョン管理 **Requires** または **BuildRequires** を使用しないでください。システムで更新できるパッケージに依存する必要がある場合は、Software Collection にそのパッケージを含めるか、システムパッケージの更新時に Software Collection を再ビルドすることを忘れないようにしてください。

6. Software Collection の基本的なパッケージがすべてメインのメタパッケージの依存関係であることを確認するには、spec ファイルの **BuildRequires** タグまたは **Requires** タグの後に以下のマクロを追加します。

```
%{?scl:Requires: %scl_runtime}
```

7. **Obsoletes**、**Conflicts** および **BuildConflicts** タグの前に **%{?scl_prefix}** を付けます。これは、たとえば、ベースシステムのインストールから **Obsolete** 削除されるなどして、Software Collection を使用して、古いシステムに新しいパッケージをデプロイできるようにします。以下に例を示します。

```
Obsoletes: %{?scl_prefix}lesspipe < 1.0
```

8. 以下の例のように、**Provides** タグの前に **%{?scl_prefix}** を付けます。

```
Provides: %{?scl_prefix}more
```

### 2.10.3. サブパッケージの変換

**-n** オプションで名前を定義するサブパッケージの場合は、以下の例のようにその名前の前に **%{?scl_prefix}** を付けます。

```
%package -n %{?scl_prefix}more
```

接頭辞は **%package** マクロだけでなく、**%description** および **%files** に対しても適用されます。以下に例を示します。

```
%description -n %{?scl_prefix}rubygems
RubyGems is the Ruby standard for publishing and managing third party
libraries.
```

サブパッケージにメインパッケージが必要な場合は、タグが **%{?scl_prefix}%{pkg_name}** を使用するようにサブパッケージの **Requires** タグも調整してください。以下に例を示します。

```
Requires: %{?scl_prefix}%{pkg_name} = %{version}-%{release}
```

### 2.10.4. RPM スクリプトの変換

本セクションでは、従来の spec ファイルの **%prep**、**%build**、**%install**、**%check**、**%pre**、および **%post** セクションで頻繁に検索できる RPM スクリプトを変換する一般的なルールを説明します。

- **%name** はすべて **%pkg_name** に置き換えます。最も重要な点として、これには **%setup** マクロの調整が含まれます。
  - スペックファイルの **%prep** セクションで **%setup** マクロを調整し、そのマクロが Software Collection 環境で異なるパッケージ名を処理できるようにします。

    ```
    %setup -q -n %{pkg_name}-%{version}
    ```

    **%setup** マクロが必要なため、Software Collection を正常に構築するには **-n** オプションとともにマクロを使用する必要があります。
- **%_root_** マクロのいずれかを使用してシステムファイルシステム階層を参照する場合は、これらのマクロに条件を使用する必要があります。これにより、従来のパッケージと Software Collection の両方を構築するために spec ファイルを使用することができます。以下の例のようにマクロを編集します。

  ```
  mkdir -p %{?scl:%_root_sysconfdir}%{?!scl:%_sysconfdir}
  ```
- 他の Software Collection パッケージに依存する Software Collection パッケージを構築する場合は、**scl enable** 機能リンクを正しく実行したり、適切なバイナリーを実行することが重要になることがよくあります。これが必要な例は、Software Collection ライブラリーに対してコンパイルするか、Software Collection でインタープリターを使用してインタープリターを実行する例です。

  以下の例のように、**%{?scl:** 接頭辞を使用してスクリプトをラップします。

  ```
  %{?scl:scl enable %scl - << \EOF}
   set -e
   ruby example.rb
   RUBYOPT="-Ilib" ruby bar.rb
   # The rest of the script contents goes here.
  %{?scl:EOF}
  ```

  スクリプトで **set -e** を指定することが重要です。これにより、スクリプトが **rpm** シェルまたは **scl** 環境で実行されるかどうかに関わらず、スクリプトの動作が一貫性を保つことが重要です。
- Software Collection パッケージのインストール時に実行されるスクリプトに注意してください。以下に例を示します。
  - **%pretrans**、**%pre**、
  - **%post**、**%postun**、**%posttrans**、
  - **%triggerin**、**%triggerun**、および **%triggerpostun**

  このスクリプトの **scl enable** 機能を使用する場合は、ベースシステムのインストールとの意図しない競合を避けるために、空の環境から開始することが推奨されます。

これを行うには、以下の例のように、Software Collection を有効にする前に **env -i -** を使用します。

```
%posttrans
%{?scl:env -i - scl enable %{scl} - << \EOF}
%vagrant_plugin_register %{vagrant_plugin_name}
%{?scl:EOF}
```

- RPM スクリプトで見つかったハードコーディングされたパスはすべて適切なマクロに置き換える必要があります。たとえば、すべての **/usr/share** を **%{_datadir}** で置き換えます。これは、**$RPM_BUILD_ROOT** 変数と **%{build_root}** マクロが **scl** マクロで移動しないため、必要です。

### 2.10.5. Software Collection の自動 Provides および Requires ならびにフィルタリングサポート

**重要**

このセクションで説明する機能は、Red Hat Enterprise Linux 6 では利用できません。

Red Hat Enterprise Linux 7 の RPM は **Provides**、自動 **Requires** およびフィルタリングをサポートします。たとえば、すべての Python ライブラリーでは、RPM は自動的に以下の **Requires** を追加します。

```
Requires: python(abi) = (version)
```

「従来の spec ファイルの変換」の説明にあるように、従来の RPM パッケージを変換する際に、**Requires** の前に **%{?scl_prefix}** を付ける必要があります。

```
Requires: %{?scl_prefix}python(abi) = (version))
```

元の RPM スクリプトは拡張できないため、これらの依存関係を検索するスクリプトを Software Collection に対して書き換える必要があるため、場合によってはフィルタリングを使用することができません。たとえば、Python の **Provides** と **Requires** の自動書き込みを行い、**macros.%{scl}-config** マクロファイルに以下の行を追加します。

```
%__python_provides /usr/lib/rpm/pythondeps-scl.sh --provides %{_scl_root} %{scl_prefix}
%__python_requires /usr/lib/rpm/pythondeps-scl.sh --requires %{_scl_root} %{scl_prefix}
```

**/usr/lib/rpm/pythondeps-scl.sh** ファイルは、従来のパッケージの **pythondeps.sh** ファイルをベースとしており、検索パスを調整します。

**pkg_config Provides** など、調整が必要な **Provides** または **Requires** がある場合には、2 つの方法があります。

- Software Collection のすべてのパッケージに適用されるように、**macros.%{scl}-config** マクロファイルに次の行を追加します。

```
%_use_internal_dependency_generator 0
%__deploop() while read FILE; do /usr/lib/rpm/rpmdeps -%{1} ${FILE}; done | /bin/sort -u
%__find_provides /bin/sh -c "%{?__filter_prov_cmd} %{__deploop P} %{?__filter_from_prov}"
%__find_requires /bin/sh -c "%{?__filter_req_cmd}  %{__deploop R} %{?__filter_from_req}"
```

```
# Handle pkgconfig's virtual Provides and Requires
%__filter_from_req | %{__sed} -e 's|pkgconfig|%{?scl_prefix}pkgconfig|g'
%__filter_from_prov | %{__sed} -e 's|pkgconfig|%{?scl_prefix}pkgconfig|g'
```

- **Provides** または **Requires** にフィルター処理を行うすべての spec ファイルのタグ定義の後に以下の行を追加します。

```
%{?scl:%filter_from_provides s|pkgconfig|%{?scl_prefix}pkgconfig|g}
%{?scl:%filter_from_requires s|pkgconfig|%{?scl_prefix}pkgconfig|g}
%{?scl:%filter_setup}
```

**重要**

フィルターを使用する場合は、変更する自動依存関係に注意する必要があります。たとえば、従来のパッケージに含まれるのが **Requires: pkgconfig(package_1)** および **Requires: pkgconfig(package_2)** であり、package_2 は Software Collection に含まれる場合は、package_1 の **Requires** タグにフィルターを適用しないようにしてください。

### 2.10.6. Software Collection マクロファイルのサポート

場合によっては、Software Collection パッケージでマクロファイルを配信しないといけない場合があります。これらは **%{?scl:%{_root_sysconfdir}}%{!?scl:%{_sysconfdir}}/rpm/** ディレクトリーにあります。これは、従来のパッケージの **/etc/rpm/** ディレクトリーに対応します。マクロファイルを読み込む際には、以下を確認します。

- **.%{scl}** を名前に追加してマクロファイルの名前を変更する場合は、ベースシステムインストールのファイルと競合しないようにします。
- マクロファイル内のマクロは、以下の例のように、拡張されていないか、条件を使用しているかのいずれかになります。

```
%__python2 %{_bindir}/python
%python2_sitelib %(%{?scl:scl enable %scl '}%{__python2} -c "from distutils.sysconfig import get_python_lib; print(get_python_lib())"%{?scl:'})
```

別の例として、Software Collection python26 に依存する Software Collection mypython を作成する必要がある場合もあります。python26 Software Collection は上記のサンプルのように **%{__python2}** マクロを定義します。このマクロは **/opt/provider/mypython/root/usr/bin/python2** に評価されますが、**python2** バイナリーは python26 Software Collection (**/opt/provider/python26/root/usr/bin/python2**) でのみ利用できます。

mypython Software Collection 環境でソフトウェアを構築するには、以下を確認します。

- python26-python-devel パッケージの一部である **macros.python.python26** マクロファイルには、以下の行が含まれます。

```
%__python26_python2 /opt/provider/python26/root/usr/bin/python2
```

- また、python26-build サブパッケージ内のマクロファイルと、Software Collection の build サブパッケージには、以下の行が含まれます。

```
%scl_package_override() {%global __python2 %__python26_python2}
```

これにより、対応する Software Collection のビルドサブパッケージが存在する場合にのみ **%{__python2}** マクロが再定義されます。これは通常、その Software Collection のソフトウェアを構築することを意味します。

### 2.10.7. Software Collection シバンのサポート

シバンは、インタープリターディレクティブとして使用されるスクリプトの最初の文字のシーケンスです。シバンは自動依存関係ジェネレーターによって処理され、場合によってはシステムの root ファイルシステム内の特定の場所を参照します。

自動依存関係ジェネレーターがシバンを処理すると、それが参照するインタープリターに応じて依存関係が追加されます。Software Collection の観点からは、以下の 2 種類のシバンがあります。

**#!/usr/bin/env *example***

このシバンは、**/usr/bin/env** プログラムにインタープリターの実行を指示します。

自動依存関係ジェネレーターは、予想通りに **/usr/bin/env** プログラムに依存関係を作成します。

**enable** スクリプトレットで **$PATH** 環境変数が適切に再定義された場合、*example* インタープリターは期待通りに Software Collection ファイルシステム階層にあります。

Software Collection パッケージでシバンを書き直し、シバンが Software Collection ファイルシステム階層にあるインタープリターへの完全パスを指定することが推奨されます。

**#!/usr/bin/*example***

このシバンはインタープリターへの直接パスを指定します。

自動依存関係ジェネレーターは、Software Collection ファイルシステム階層外にある **/usr/bin/example** インタープリターの依存関係を作成します。ただし、Software Collection のパッケージを構築する場合は、多くの場合、Software Collection ファイルシステム階層にある **%{?_scl_root}/usr/bin/example** インタープリターに依存します。

**$PATH** 環境変数を適切に再定義しても、インタープリターが使用されるものには影響しないことに注意してください。Software Collection ファイルシステム階層外にあるインタープリターのシステムバージョンは常に使用されます。大半の場合、これは望ましくありません。

このタイプのシバンを使用していて、シバンが Software Collection パッケージの構築時に Software Collection ファイルシステム階層を参照するようにするには、以下のようなコマンドを使用します。

```
find %{buildroot} -type f | \
  xargs sed -i -e '1 s"^#!/usr/bin/example"#!%{?_scl_root}/usr/bin/example"'
```

*/usr/bin/example* は、使用するインタープリターになります。

### 2.10.8. Software Collection を別の Software Collection に依存させる

Software Collection を別の Software Collection のパッケージに依存するようにするには、これらのタグが依存関係を適切に定義できるように、依存する Software Collection のスペックファイルの **BuildRequires** タグおよび **Requires** タグを調整する必要があります。

たとえば、**software_collection_1** と **software_collection_2** という名前の 2 つの Software Collections の依存関係を定義するには、以下の 3 つの行をアプリケーションの spec ファイルに追加します。

```
BuildRequires: scl-utils-build
Requires: %scl_require software_collection_1
Requires: %scl_require software_collection_2
```

以下のように、spec ファイルの Preamble の前に **%scl_package** マクロも含まれていることを確認してください。

```
%{?scl:%scl_package less}
```

**%scl_package** マクロは、Software Collection の spec ファイルに含める必要があることに注意してください。

**%scl_require_package** マクロを使用して、以下の例にあるように、特定の Software Collection から特定のパッケージの依存関係を定義することもできます。

```
BuildRequires: scl-utils-build
Requires: %scl_require_package software_collection_1 package_name
```

## 2.11. すべての SOFTWARE COLLECTION ディレクトリーのアンインストール

この **yum remove** コマンドは、Software Collection パッケージおよび Software Collection runtime サブパッケージの削除後に削除されたサブパッケージが提供するディレクトリーをアンインストールしません。

すべてのディレクトリーをアンインストールするには、そのパッケージとサブパッケージは、runtime サブパッケージに依存します。これを行うには、**%scl_runtime** マクロで、以下の行を、各パッケージおよびサブパッケージの spec ファイルに追加します。

```
%{?scl:Requires: %scl_runtime}
```

上記の行を追加すると、runtime サブパッケージがパッケージやサブパッケージに依存しない限り、そのパッケージおよびサブパッケージが提供するすべてのディレクトリーが正しく削除されます。

## 2.12. SOFTWARE COLLECTION の構築

「従来の spec ファイルの変換」 の説明に従って、Software Collection の従来の spec ファイルを正しく変換した場合は、Software Collection と従来のビルドルートの両方で、生成されるパッケージを構築できます。従来のビルドルートで変換されたパッケージを構築すると、従来のベースシステム RPM パッケージが生成されますが、%{scl}-build を含む Software Collection ビルドルートでビルドすると、Software Collection パッケージが生成されます。

システムで Software Collection を構築するには、以下のコマンドを実行します。

```
rpmbuild -ba package.spec --define 'scl name'
```

上記のコマンドと、従来のパッケージ (**rpmbuild -ba *package*.spec**) を構築する標準コマンドの違いは、Software Collection を構築する際に、**rpmbuild** コマンドに **--define** オプションを追加する必要があります。

**--define** オプションは、**scl** マクロを定義します。これは、Software Collection の spec ファイル (***package*.spec**) で設定された Software Collection を使用します。

標準コマンド **rpmbuild -ba *package*.spec** を使用して Software Collection を構築するには、**package.spec** ファイルで以下を指定します。

```
BuildRequires: software_collection-build
```

ここで、*software_collection* は Software Collection の名前になります。

### 2.12.1. サブパッケージを構築せずにソフトウェアコレクションの再構築

ビルドサブパッケージ (*software_collection*-build) がない Software Collection を再構築する場合は、Software Collection メタパッケージを再ビルドして、この **rpmbuild -ba *package*.spec --define 'scl *name*'** コマンドを回避して、ビルドサブパッケージを作成できます。

システムに scl-utils-build パッケージをインストールする必要があります。そうしないと、**rpmbuild** コマンドで Software Collection メタパッケージを再構築すると失敗します。

scl-utils-build パッケージの詳細は、「Software Collections のサポートの有効化」を参照してください。

### 2.12.2. debuginfo ファイルの競合の回避

同じ **Source** タグを指定する 2 つの Software Collection パッケージ (または従来の RPM パッケージと Software Collection パッケージ) を構築すると、**%_builddir** ディレクトリーにあるソースファイルを同じディレクトリーに展開すると、その **debuginfo** パッケージによりファイルの競合が発生します。このような競合により、ユーザーは両方のパッケージを同時に同じシステムにインストールすることができません。

これらのファイルの競合を回避するには、いずれかのパッケージの spec ファイルを変更して、アップストリームのソースを一意の名前のトップディレクトリーに展開する必要があります。これにより、**%_builddir** ディレクトリーの下にあるビルドツリーにディレクトリーレベルが 1 つ以上追加されます。これを実行することで、**debuginfo** パッケージ生成スクリプトは、他の **debuginfo** パッケージのファイルと競合しない **debuginfo** ファイルを作成します。

元のスペックファイルと変更されたスペックファイルを比較した diff ファイルがどのように見えるかを確認するには、次の例を参照してください。

```
--- a/tbb.spec
+++ b/tbb.spec
@@ -66,11 +66,13 @@ PDF documentation for the user of the Threading Building Block (TBB)
 C++ library.

 %prep
-%setup -q -n %{sourcebasename}
+%setup -q -c -n %{name}
+cd %{sourcebasename}
 %patch1 -p1
 %patch2 -p1

 %build
+cd %{sourcebasename}
 %{?scl:scl enable %{scl} - << \EOF}
 make %{?_smp_mflags} CXXFLAGS="$RPM_OPT_FLAGS" tbb_build_prefix=obj
 %{?scl:EOF}
@@ -81,6 +83,7 @@ done
```

```
 %install
 rm -rf $RPM_BUILD_ROOT
+cd %{sourcebasename}
 mkdir -p $RPM_BUILD_ROOT/%{_libdir}
 mkdir -p $RPM_BUILD_ROOT/%{_includedir}

@@ -108,20 +111,20 @@ done

 %files
 %defattr(-,root,root,-)
-%doc COPYING doc/Release_Notes.txt
+%doc %{sourcebasename}/COPYING %{sourcebasename}/doc/Release_Notes.txt
 %{_libdir}/*.so.2

 %files devel
 %defattr(-,root,root,-)
-%doc CHANGES
+%doc %{sourcebasename}/CHANGES
 %{_includedir}/tbb
 %{_libdir}/*.so
 %{_libdir}/pkgconfig/*.pc

 %files doc
 %defattr(-,root,root,-)
-%doc doc/Release_Notes.txt
-%doc doc/html
+%doc %{sourcebasename}/doc/Release_Notes.txt
+%doc %{sourcebasename}/doc/html

 %changelog
 * Wed Nov 13 2013 John Doe <jdoe@example.com> - 4.1-5.20130314
```

# 第3章 高度なトピック

本章では、Software Collections のパッケージ化に関する高度なトピックを説明します。

## 3.1. NFS での SOFTWARE COLLECTIONS の使用

一部の環境では、この要件には、ユーザーが希望するアプリケーションやツールのバージョンをインストールするのではなく、アプリケーションやツールがどのように配布されるかについての集中型モデルが必要になることがよくあります。このようにして NFS は、集中管理されたソフトウェアをマウントする一般的な方法です。

NFS 経由で Software Collection を使用する **nfsmountable** には、Software Collection マクロを定義する必要があります。Software Collection の構築時にマクロが定義されている場合には、生成される Software Collection には、Software Collection の **/opt** ファイルシステム階層外にある状態ファイルと設定ファイルがあります。これにより、NFS に **/opt** ファイルシステム階層を読み取り専用としてマウントできます。また、状態ファイルと設定ファイルの管理が容易になります。

NFS を介した Software Collections のサポートが必要ない場合は、**nfsmountable** の使用は任意ですが推奨されます。

**nfsmountable** マクロを定義するには、Software Collection のメタパッケージの spec ファイルに以下の行が含まれていることを確認してください。

```
%global nfsmountable 1

%scl_package %scl
```

上記のように、**nfsmountable** マクロを定義する前に **%scl_package** マクロを定義する必要があります。これは、**nfsmountable** マクロが定義されているかどうかに応じて、**%scl_package** マクロが、**_sysconfdir**、**_sharedstatedir**、**_localstatedir** のマクロを再定義するためです。再定義されたマクロで **nfsmountable** が変更する値は、以下の表で説明されています。

**表3.1 Software Collection マクロの値の変更**

| マクロ | 元の定義 | 元の定義に対する拡張された値 | 変更された定義 | 元の定義に対する拡張された値 |
|---|---|---|---|---|
| **_sysconfdir** | %{_scl_root}/etc | /opt/provider/%{scl}/root/etc | %{_root_sysconfdir}%{_scl_prefix}/%{scl} | /etc/opt/provider/%{scl} |
| **_sharedstatedir** | %{_scl_root}/var/lib | /opt/provider/%{scl}/root/var/lib | %{_root_localstatedir}%{_scl_prefix}/%{scl}/lib | /var/opt/provider/%{scl}/lib |
| **_localstatedir** | %{_scl_root}/var | /opt/provider/%{scl}/root/var | %{_root_localstatedir}%{_scl_prefix}/%{scl} | /var/opt/provider/%{scl} |

### 3.1.1. ディレクトリー構造およびファイル所有者の変更

**nfsmountable** マクロは、**scl_install** および **scl_files** マクロがディレクトリー構造の作成方法にも影響し、**rpmbuild** コマンドの実行時にファイルの所有権を設定します。

たとえば、定義された **nfsmountable** マクロを持つ **software_collection** という名前の Software Collection のディレクトリー構造は、以下のようになります。

```
$ rpmbuild -ba software_collection.spec --define 'scl software_collection'
...
$ rpm -qlp software_collection-runtime-1-1.el6.x86_64
/etc/opt/provider/software_collection
/etc/opt/provider/software_collection/X11
/etc/opt/provider/software_collection/X11/applnk
/etc/opt/provider/software_collection/X11/fontpath.d
...
/opt/provider/software_collection/root/usr/src
/opt/provider/software_collection/root/usr/src/debug
/opt/provider/software_collection/root/usr/src/kernels
/opt/provider/software_collection/root/usr/tmp
/var/opt/provider/software_collection
/var/opt/provider/software_collection/cache
/var/opt/provider/software_collection/db
/var/opt/provider/software_collection/empty
...
```

### 3.1.2. Software Collections の登録および登録解除

Software Collection が NFS で共有されているが、お使いのシステムにローカルにインストールされていない場合には、その Software Collection を登録して **scl** ツールに認識させる必要があります。

Software Collection の登録は、以下の **scl register** コマンドを実行して行います。

```
$ scl register /opt/provider/software_collection
```

*/opt/provider/software_collection* は、登録する Software Collection のファイルシステム階層への絶対パスです。パスのディレクトリーに、有効な Software Collection ファイルシステム階層と見なされる **enable** スクリプトレットと **root/** ディレクトリーに含まれる必要があります。

Software Collection の登録解除は、**scl** ツールが登録済み Software Collection を認識させないようにする際に実行する逆の操作です。

Software Collection の登録を解除するには、**scl** コマンドの実行時に **deregister** スクリプトレットを呼び出します。

```
$ scl deregister software_collection
```

ここで、*software_collection* は、登録を解除する Software Collection の名前に置き換えます。

#### 3.1.2.1. Software Collection Metapackage での (de)register スクリプトレットの使用

Software Collection メタパッケージで (de)register スクリプトレットの有効化方法と同様に、(de)register スクリプトレットを指定できます。スクリプトセットを指定する場合は、メタパッケージ spec ファイルの **%file** セクションに明示的に追加するようにしてください。

(de)register スクリプトセットの指定例は、以下のサンプルコードを参照してください。

```
%install
%scl_install

cat >> %{buildroot}%{_scl_scripts}/enable << EOF
# Contents of the enable scriptlet goes here
...
EOF

cat >> %{buildroot}%{_scl_scripts}/register << EOF
# Contents of the register scriptlet goes here
...
EOF

cat >> %{buildroot}%{_scl_scripts}/deregister << EOF
# Contents of the deregister scriptlet goes here
...
EOF
...
%files runtime -f filelist
%scl_files
%{_scl_scripts}/register
%{_scl_scripts}/deregister
```

register スクリプトレットでは、**/etc/opt/** や **/var/opt/** にファイルを作成するコマンドなど、Software Collection の登録時に実行するコマンドをオプションで指定できます。

## 3.2. SOFTWARE COLLECTION スクリプトレットの環境モジュールへの変換

環境モジュールを使用すると、たとえば、シェル環境を動的に変更することで、さまざまなバージョンのアプリケーションを管理できます。環境モジュールシステムで Software Collection を使用するには、**/usr/share/Modules/bin/createmodule.sh** スクリプトで Software Collection の **enable** スクリプトレットを環境モジュールに変換します。

**手順3.1 enable スクリプトレットの環境モジュールへの変換**

1. environment-modules パッケージがシステムにインストールされていることを確認します。

   ```
   # yum install environment-modules
   ```

2. **/usr/share/Modules/bin/createmodule.sh** スクリプトを実行して、Software Collection の **enable** スクリプトレットを環境モジュールに変換します。

   ```
   /usr/share/Modules/bin/createmodule.sh /path/to/enable/scriptlet
   ```

   */path/to/enable/scriptlet* は、変換する **enable** スクリプトレットのファイルパスに置き換えます。

3. Software Collection メタパッケージの **%pre** セクションの **/usr/share/Modules/bin/createmodule.sh /path/to/enable/scriptlet** コマンド (**enable** スクリプトレットを生成するコード) を追加します。

Software Collection パッケージのいずれかに、**enable** スクリプトレットをファイルとしてパッケージ化されている場合は、**%post** セクションに **/usr/share/Modules/bin/createmodule.sh /*path/to/enable/scriptlet*** コマンドを追加します。

環境モジュールの詳細は、**module**(1) man ページを参照してください。

## 3.3. SYSPATHS サブパッケージの提供

Software Collection のパッケージを使用するには、ユーザーは従来の RPM パッケージを使用する場合とは異なる特定のタスクを実行する必要があります。たとえば、別の場所にインストールされたバイナリーを見つけるために、**PATH**、**LD_LIBRARY_PATH** などの環境変数を変更する **scl enable** 呼び出しを使用する必要があります。systemd サービスの代替名を使用する必要もあります。一部のスクリプトは、**/usr/bin/mysql** など、完全なパスを使用してバイナリーを呼び出すこともできます。その結果、これらのスクリプトは Software Collection で機能しない可能性があります。

上述の問題に対処するための推奨ソリューションは、syspaths サブパッケージを使用することです。基本的な概念は、ユーザーがベースシステムのインストールに影響を及ぼさずに、異なるバージョンの同じパッケージを使用できるようにすることですが、Software Collection パッケージを従来の RPM パッケージであるかのように使用するオプションを使用すると、Software Collection を簡単に使用できるようになります。

オプションの syspaths サブパッケージ (rh-mariadb102-syspaths など) は、標準パス (通常は **/usr/bin**) にインストールされるシェルラッパーおよびシンボリックリンクを提供します。つまり、syspaths サブパッケージのインストールを選択すると、ユーザーがベースシステムのインストールを意図的に変更し、一度に複数のバージョンの同じパッケージをインストールし、実行する必要のないユーザーに適した syspaths サブパッケージになります。これは特にデータベースを使用する場合です。

syspaths サブパッケージを使用すると、Software Collection パッケージでスクリプトを調整する必要がなくなり、スクリプトを簡単に使用できるようになります。syspaths サブパッケージは、ベースシステムインストールのパッケージと競合するため、従来のパッケージを syspaths サブパッケージとともにインストールすることはできません。懸念される場合は、コンテナーベースの技術を使用して、ベースシステムのインストールから syspaths サブパッケージを分離することを検討してください。

### 3.3.1. syspaths サブパッケージの命名

syspaths サブパッケージの概念を使用する各 Software Collection には、通常 syspaths サブパッケージが複数含まれています。syspaths サブパッケージは、ラッパーまたはシンボリックリンクで提供できるファイルとともに、各パッケージで利用できます。

その上には、software_collection_1 という名前の Software Collection メタパッケージがあります。ここで software_collection_1-syspaths、は Software Collection の名前になります。この software_collection_1-syspaths サブパッケージには、Software Collection が含まれる他の syspaths サブパッケージが必要です。これにより、software_collection_1-syspaths サブパッケージをインストールすると、その他の syspaths パッケージがすべてインストールされます。

たとえば、software_collection_1-package_1 パッケージに含まれるバイナリーファイル **binary_1** と、software_collection_1-package_2 パッケージに含まれるバイナリーファイル **binary_2** のラッパーを含めたい場合は、software_collection_1 Software Collection に以下の 3 つの syspaths サブパッケージを作成します。

```
software_collection_1-syspaths
software_collection_1-package_1-syspaths
software_collection_1-package_2-syspaths
```

### 3.3.2. syspaths サブパッケージに含まれるファイル

syspaths サブパッケージに組み込むのに適したファイルは、ユーザーが対話するバイナリーのシェルラッパーです。

以下は、software_collection_1 に含まれるバイナリーファイルの **binary_1** のラッパーの例です。これは、**/opt/rh/software_collection_1/root/usr/bin/binary_1** にあります。

```
#!/bin/bash
source scl_source enable software_collection_1
exec "/opt/rh/software_collection_1/root/usr/bin/binary_1" "$@"
```

**/usr/bin/binary_1** にこのラッパーをインストールする場合は、**scl enable software_collection_1** を付けなくても **binary_1** コマンドを実行できます。**/usr/bin/** にインストールされているラッパーは正しい環境を設定し、**/opt/provider/%{scl}** ファイルシステム階層とともに配置されるターゲットバイナリーを実行します。

### 3.3.3. syspaths Wrapper の制限

syspaths ラッパーがシェルスクリプトであるため、ユーザーはターゲットバイナリーと同様にラッパーを使用してすべてのタスクを実行できないことを意味します。たとえば、**gdb** を使用してバイナリーをデバッグする場合、**/opt/provider/%{scl}** はラッパーシェルスクリプトでは機能しないため、**gdb** ファイルシステム階層を指定するフルパスを使用する必要があります。

### 3.3.4. syspaths サブパッケージのシンボリックリンク

バイナリーファイルのラッパー以外に、**/opt**、**/etc/opt/**、または **/var/opt/** ディレクトリー以外のインストールに適したファイルが多数存在するため、syspaths サブパッケージで提供できます。たとえば、データベースファイル (通常は **/var/opt/provider/%{scl}** に配置) へのパスを作成して、**/var/lib/** にあるシンボリックリンクを簡単に検出できます。ただし、一部のシンボリックリンクでは、ベースシステムインストールの従来の RPM パッケージ名が競合する可能性があるため、元の名前で **/var/lib/** にインストールしないことが推奨されます。

シンボリックリンク **/var/lib/software_collection_1-original_name** または同様の名前を付けることが推奨されます。ログファイルの場合は、推奨される名前は **/var/log/software_collection_1-original_name** または同様の名前になります。名前自体は重要ではないことに注意してください。ここでの設計目標は、これらのファイルを **/var/lib/** または **/var/log/** ディレクトリーで簡単に検索することです。

設定ファイルにも同じように適用されるゴールは、シンボリックリンクを **/etc** ディレクトリーで簡単に検出することです。

### 3.3.5. 接頭辞のないサービス

systemd および SysV init サービスは、デーモンサービスとのユーザー対話の例です。通常、サービスの起動時にコマンド **scl enable** に追加する必要はありません。サービスには、クリーンな環境で設計が開始されるためです。ただし、ユーザーは正しいサービス名を使用する必要があります。通常は、Software Collection 名 (例: **rh-mariadb102-mariadb**) で始まる名前になります。

syspaths サブパッケージを使用すると、適切な syspaths サブパッケージがインストールされている場合は、**mariadb**、**mongod**、**postgresql** などの従来のサービス名を使用できます。これを行うには、従来のサービスファイルを参照するシンボリックリンク名に Software Collection 名を含めずに、シンボリックリンクを作成します。

たとえば、**/etc/rc.d/init.d/software_collection_1-service_1** ファイルが通常提供する software_collection_1 Software Collection のサービス **_1** は、以下のシンボリックリンクを作成して **service_1** としてアクセスできます。

```
/etc/rc.d/init.d/service_1 -> /etc/rc.d/init.d/software_collection_1-service_1
```

systemd ユニットファイルの場合は、以下のようになります。

```
/usr/lib/systemd/system/service_1 -> /usr/lib/systemd/system/software_collection_1-service_1
```

## 3.4. SOFTWARE COLLECTIONS でのサービス管理

Software Collection のパッケージ化を行う際、ユーザーは、Red Hat Enterprise Linux 6 の **service**、**chkconfig**、Red Hat Enterprise Linux 7 の **systemctl** などの Software Collection が提供するサービス (daemon)、またはシステムデフォルトツールを使用した関連アプリケーションの 1 つを直接管理できます。

Red Hat Enterprise Linux 6 の Software Collections では、以下のように spec ファイルの **%install** セクションを調整して、Software Collection に含まれるサービスのシステムバージョンと名前が競合しないようにします。

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/rc.d/init.d/%{?scl_prefix}service_name
```

*service_name* を、サービスの実際の名前に置き換えます。

Red Hat Enterprise Linux 7 の Software Collections では、以下のように spec ファイルの **%install** セクションを調整します。

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{_unitdir}/%{?
scl_prefix}service_name.service
```

この設定が導入されると、以下のように Software Collection に含まれるサービスのバージョンを参照できます。

```
%{?scl_prefix}service_name
```

ユーザーの環境から SysV init スクリプト (または Red Hat Enterprise Linux 7 の systemd サービスファイル) に環境変数が伝播されることはありません。これは想定されており、サービスが常にクリーンな環境で起動されるようにします。ただし、SysV init スクリプト (または systemd サービスファイル) で実行するプロセスの Software Collection 環境を適切に設定する必要があります。

### 3.4.1. サービス環境の設定

サービスに対して有効にする Software Collection を設定することが推奨されます。本セクションの方向は、*software_collection* という名前の Software Collection を設定可能にする方法を説明します。

**手順3.2 Red Hat Enterprise Linux 6 でサービス用の環境の設定**

1. 以下の内容で、**/opt/provider/*software_collection*/service-environment** に設定ファイルを作成します。

   ```
   [SCLNAME]_SCLS_ENABLED="software_collection"
   ```

*SCLNAME* を、(大文字で書かれた) Software Collection の名前の一意の識別子に置き換えます。

*software_collection* を、 **%scl_name** マクロで定義した Software Collection の名前に置き換えます。

2. SysV init スクリプトの最初に以下の行を追加します。

```
source /opt/provider/software_collection/service-environment
```

3. SysV init スクリプトで、**/opt/provider/** ファイルシステム階層にあるバイナリーを実行するコマンドを決定します。これらのコマンドの前に **scl enable $[SCLNAME]_SCLS_ENABLED** (Software Collection 環境でコマンドを実行する場合と同様) を指定してください。

たとえば、以下の行を置き換えます。

```
/usr/bin/daemon_binary --argument-1 --argument-2
```

上記の行を、以下のように置き換えます。

```
scl enable $[SCLNAME]_SCLS_ENABLED -- /usr/bin/daemon_binary --argument-1 --
argument-2
```

4. **su** や **runuser** などの一部のコマンドは、環境変数をクリアします。したがって、これらのコマンドが SysV init スクリプトで使用されている場合は、このコマンドの実行後に Software Collection を再度有効にします。

たとえば、以下の行を置き換えます。

```
su - user_name -c '/usr/bin/daemon_binary --argument-1 --argument-2'
```

上のコマンドを、下のコマンドに置き換えます。

```
su - user_name -c '\
  source /opt/provider/software_collection/service-environment \
  scl enable $SCLNAME_SCLS_ENABLED -- /usr/bin/daemon_binary --argument-1 --
argument-2'
```

**手順3.3 Red Hat Enterprise Linux 7 でサービス用の環境の設定**

1. 以下の内容で、**/opt/provider/*software_collection*/service-environment** に設定ファイルを作成します。

```
[SCLNAME]_SCLS_ENABLED="software_collection"
```

*SCLNAME* を、(大文字で書かれた) Software Collection の名前の一意の識別子に置き換えます。

*software_collection* を、 **%scl_name** マクロで定義した Software Collection の名前に置き換えます。

2. systemd サービスファイルに以下の行を追加して、設定ファイルを読み込みます。

```
EnvironmentFile=/opt/provider/software_collection/service-environment
```

3. systemd サービスファイルでは、Software Collection 環境でコマンドを実行するときと同様に、**ExecStartPre**、**ExecStart**、および同様のディレクティブで指定されたすべてのコマンドの前に **scl enable $[SCLNAME]_SCLS_ENABLED** を付けてください。

```
ExecStartPre=/usr/bin/scl enable $[SCLNAME]_SCLS_ENABLED --
/opt/provider/software_collection/root/usr/bin/daemon_helper_binary --argument-1 --
argument-2
ExecStart=/usr/bin/scl enable $[SCLNAME]_SCLS_ENABLED --
/opt/provider/software_collection/root/usr/bin/daemon_binary --argument-1 --argument-2
```

**警告**

**ExecStart*** コマンドの前に **scl enable ...** を使用すると、サービスが、SELinux ポリシーで示されるターゲットの SELinux コンテキストに移行できなくなります。SELinux を使用してこのサービスを制限する場合は、systemd サービスファイルでバイナリーを直接実行する必要があります。バイナリーが Software Collection に含まれる共有ライブラリーにリンクされている場合、**DT_RUNPATH** 属性は **scl enable ...** ラッパーを使用せずに、ランタイム時にこれらの共有ライブラリーにアクセスできるようにするのに役立ちます。詳細は、「Red Hat Enterprise Linux 7 での SELinux サポート」 を参照してください。

## 3.5. SOFTWARE COLLECTION LIBRARY のサポート

Software Collection 環境でのみ使用するライブラリー、またはシステムで利用可能なライブラリーに加えて、以下のように、**enable** スクリプトレットの **LD_LIBRARY_PATH** 環境変数を更新します。

```
export LD_LIBRARY_PATH="%{_libdir}\${LD_LIBRARY_PATH:+:\${LD_LIBRARY_PATH}}"
```

この設定は、Software Collection が有効な場合に、システムで利用可能なライブラリーのバージョンよりも、Software Collection のライブラリーのバージョンが推奨されます。

**注記**

Software Collection でプライベート共有ライブラリーを配布する場合は、**LD_LIBRARY_PATH** 環境変数の代わりに **DT_RUNPATH** 属性を使用して、Software Collection 環境でプライベートの共有ライブラリーにアクセスできるようにすることを検討してください。

### 3.5.1. Software Collection 以外のライブラリーの使用

Software Collection 環境外で使用するライブラリーを配布する場合は、この目的で **/etc/ld.so.conf.d/** ディレクトリーを使用できます。

**警告**

システムで利用できるライブラリー **/etc/ld.so.conf.d/** には使用しないでください。**/etc/ld.so.conf.d/** を使用すると、Software Collection のライブラリーのバージョンが、ライブラリーのシステムバージョンよりも優先される可能性があるため、システムで利用できないライブラリーにのみの使用が推奨されます。これにより、予期せぬ終了やデータ損失などの、アプリケーションのシステムバージョンの動作が望ましくない可能性があります。

**手順3.4 Software Collection のライブラリーに /etc/ld.so.conf.d/ を使用**

1. **%{?scl_prefix}libs.conf** という名前のファイルを作成し、spec ファイル設定を適宜調整します。

```
SOURCE2: %{?scl_prefix}libs.conf
```

2. **%{?scl_prefix}libs.conf** ファイルに、Software Collection に関連付けられたライブラリーのバージョンがあるディレクトリーの一覧を含めます。以下に例を示します。

```
/opt/provider/software_collection_1/root/usr/lib64/
```

上記の例では、Software Collection **software_collection_1** に含まれる **/usr/lib64/** ディレクトリーが一覧に含まれます。

3. 以下のように **%{?scl_prefix}libs.conf** ファイルがインストールされているように、spec ファイルの **%install** セクションを編集します。

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/ld.so.conf.d/
```

### 3.5.2. Software Collection 名を使用したライブラリーメジャー soname の接頭辞

Software Collection に含まれるライブラリーを使用する場合は、ベースシステムのインストールの一部として、同じメジャー soname を持つライブラリーがすでにシステムで使用できることに注意してください。そのため、Software Collection に含まれるライブラリーに対してアプリケーションを構築する場合は、**scl enable** コマンドの使用を忘れないでください。これを実行しないと、ライブラリーの誤ったシステムバージョンにリンクされた、誤った環境でアプリケーションが実行される可能性があります。

**警告**

誤った環境でアプリケーションを実行することや (Software Collection 環境ではなくシステム環境など)、アプリケーションを誤ったライブラリーにリンクすると、予期せぬ終了やデータ損失など、アプリケーションの望ましくない動作が発生する可能性があることに注意してください。

**LD_LIBRARY_PATH** 環境変数が正しく設定されていない場合でも、アプリケーションが誤ったライブラリーにリンクされていないことを確認するには、Software Collection に含まれるライブラリーのメジャー soname を変更します。メジャーの soname を変更するには、メジャーの soname バージョン番号の前に Software Collection 名を付けることが推奨されます。

以下は、**mysql55-** 接頭辞が付いた MySQL クライアントライブラリーの例です。

```
$ rpm -ql mysql55-mysql-libs | grep 'lib.*so'
/opt/provider/mysql55/root/usr/lib64/mysql/libmysqlclient.so.mysql55-18
/opt/provider/mysql55/root/usr/lib64/mysql/libmysqlclient.so.mysql55-18.0.0
```

同じシステムでは、MySQL クライアントライブラリーのシステムバージョンを以下に示します。

```
$ rpm -ql mysql-libs | grep 'lib.*so'
/usr/lib64/mysql/libmysqlclient.so.18
/usr/lib64/mysql/libmysqlclient.so.18.0.0
```

この **rpmbuild** ユーティリティーは、バージョン付けされた共有ライブラリーを含むパッケージ用の自動 **Provides** タグを生成します。上記のように soname の接尾辞を付けない場合は、 mysql パッケージの場合の **Provides** の例は **libmysqlclient.so.18()(64bit)** となります。この **Provides** を使用して、RPM は誤った RPM パッケージを選択することができるため、アプリケーションには要件がなくなります。

上記のように soname の前に付ける場合は、mysql の場合に、生成した **Provides** の例は、**libmysqlclient.so.mysql55-18()(64bit)** となります。この **Provides** では、RPM で正しい RPM 依存関係が選択され、アプリケーションの要件が適用されます。

通常、Software Collection パッケージは、ベースシステムインストールのパッケージによりすでに提供されているシンボルを提供しないようにしてください。このルールの例外の 1 つは、ベースシステムインストールのパッケージでシンボルを使用する場合です。

### 3.5.3. Red Hat Enterprise Linux 7 での Software Collection Library のサポート

Red Hat Enterprise Linux 7 の Software Collection を構築する場合は、**%__provides_exclude_from** マクロを使用して、自動的に生成される RPM シンボル用に特定のファイルをスキャンしないようにします。

たとえば、**%{_libdir}** ディレクトリー内の **.so** ファイルのスキャンを防ぐには、Software Collection の spec ファイルに、**BuildRequires** タグまたは **Requires** タグの前に以下の行を追加します。

```
%if %{?scl:1}%{!?scl:0}
  # Do not scan .so files in %{_libdir}
  %global __provides_exclude_from ^%{_libdir}/.*.so.*$
%endif
```

この機能は、自動の **Provides** および **Requires** の RPM サポートの一部です。詳細は「Software Collection の自動 Provides および Requires ならびにフィルタリングサポート」を参照してください。

## 3.6. SOFTWARE COLLECTION .PC ファイルのサポート

.pc ファイルは、**pkg-config** プログラムが使用する特別なメタデータファイルで、システムで利用できるライブラリーに関する情報を保存します。

Software Collection 環境でのみ使用する .pc ファイル、またはシステムにインストールされている .pc

ファイルも一緒に使用する場合は、**PKG_CONFIG_PATH** 環境変数を更新します。.pc ファイルで定義されている内容に応じて、**%{_libdir}** マクロの **PKG_CONFIG_PATH** 環境変数 (ライブラリーディレクトリー (通常は **/usr/lib/** または **/usr/lib64/**) に拡張)、または **%{_datadir}** マクロ (通常は共有ディレクトリー **/usr/share/** に拡張) を更新します。

ライブラリーディレクトリーが .pc ファイルに定義されている場合は、以下のように Software Collection の spec ファイルの **%install** セクションを調整して、**PKG_CONFIG_PATH** 環境変数を更新します。

```
%install
cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export PKG_CONFIG_PATH="%
{_libdir}/pkgconfig\${PKG_CONFIG_PATH:+:\${PKG_CONFIG_PATH}}"
EOF
```

共有ディレクトリーが .pc ファイルに定義されている場合は、以下のように Software Collection の spec ファイルの **%install** セクションを調整して、**PKG_CONFIG_PATH** 環境変数を更新します。

```
%install
cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export PKG_CONFIG_PATH="%
{_datadir}/pkgconfig\${PKG_CONFIG_PATH:+:\${PKG_CONFIG_PATH}}"
EOF
```

上記の 2 つの例は、Software Collection が有効な場合に、Software Collection の .pc ファイルがシステムで利用可能な .pc ファイルよりも優先されるように、**enable** スクリプトレットを設定します。

Software Collection は、たとえば **/usr/bin** ディレクトリーなど、Software Collection を有効にするためにシステムが認識できるラッパースクリプトを提供します。この場合は、Software Collection が無効であっても .pc ファイルがシステムに表示されていることを確認します。

無効になっている Software Collection からの .pc ファイルを使用できるようにするには、Software Collection に関連付けられた .pc ファイルへのパスで **PKG_CONFIG_PATH** 環境変数を更新します。.pc ファイルで定義されている内容に応じて、**%{_libdir}** マクロ (ライブラリーディレクトリーに拡張)、または**%{_datadir}** マクロ (共有ディレクトリーに拡張) の **PKG_CONFIG_PATH** 環境変数を更新します。

**手順3.5 %{_libdir} の PKG_CONFIG_PATH 環境変数の更新**

1. **%{_libdir}** マクロの **PKG_CONFIG_PATH** 環境変数を更新するには、カスタムスクリプト **/etc/profile.d/*name.sh*** を作成します。このスクリプトは、システムでシェルが起動するとあらかじめ読み込みます。

   たとえば、以下のファイルを作成します。

   ```
   %{?scl_prefix}pc-libdir.sh
   ```

2. **PKG_CONFIG_PATH** 変数を変更して .pc ファイルを参照するように短いスクリプト **pc-libdir.sh** を使用します。

   ```
   export PKG_CONFIG_PATH="%
   {_libdir}/pkgconfig:/opt/provider/software_collection/path/to/your/pc_files"
   ```

3. このファイルを Software Collection パッケージの spec ファイルに追加します。

```
SOURCE2: %{?scl_prefix}pc-libdir.sh
```

4. Software Collection パッケージの spec ファイルの **%install** セクションを調整して、このファイルをシステムの **/etc/profile.d/** ディレクトリーにインストールします。

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/profile.d/
```

**手順3.6 %{_datadir} の PKG_CONFIG_PATH 環境変数の更新**

1. **%{_datadir}** マクロの **PKG_CONFIG_PATH** 環境変数を更新するには、カスタムスクリプト **/etc/profile.d/*name.sh*** を作成します。このスクリプトは、システムでシェルが起動するとあらかじめ読み込みます。

   たとえば、以下のファイルを作成します。

```
%{?scl_prefix}pc-datadir.sh
```

2. **PKG_CONFIG_PATH** 変数を変更して .pc ファイルを参照するように短いスクリプト **pc-datadir.sh** を使用します。

```
export PKG_CONFIG_PATH="%
{_datadir}/pkgconfig:/opt/provider/software_collection/path/to/your/pc_files"
```

3. このファイルを Software Collection パッケージの spec ファイルに追加します。

```
SOURCE2: %{?scl_prefix}pc-datadir.sh
```

4. Software Collection パッケージの spec ファイルの **%install** セクションを調整して、このファイルをシステムの **/etc/profile.d/** ディレクトリーにインストールします。

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/profile.d/
```

## 3.7. SOFTWARE COLLECTION MANPATH サポート

有効な Software Collection からの man ページを表示できるようにシステムで **man** コマンドを使用できるようにするには、Software Collection に関連付けられた man ページへのへのパスで **MANPATH** 環境変数を更新します。

**MANPATH** 環境変数を更新するには、Software Collection の spec ファイルの **%install** セクションに以下を追加します。

```
%install
cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export MANPATH="%{_mandir}:\${MANPATH:-}"
EOF
```

これにより、**MANPATH** 環境変数を更新する **enable** スクリプトレットが設定されます。その後、Software Collection が有効になっていない限り、Software Collection に関連付けられた man ページは表示されません。

Software Collection は、たとえば **/usr/bin** ディレクトリーなど、Software Collection を有効にするためにシステムが認識できるラッパースクリプトを提供します。この場合は、Software Collection が無効であっても、man ページがシステムに表示されていることを確認します。

システムの **man** コマンドが、無効になっている Software Collection の man ページを表示できるようにするには、Software Collection に関連する man ページのパスで **MANPATH** 環境変数を更新します。

**手順3.7 無効化されたソフトウェアコレクションの MANPATH 環境変数の更新**

1. **MANPATH** 環境変数を更新するには、カスタムスクリプト **/etc/profile.d/*name.sh*** を作成します。このスクリプトは、システムでシェルが起動するとあらかじめ読み込みます。

   たとえば、以下のファイルを作成します。

   ```
   %{?scl_prefix}manpage.sh
   ```

2. **MANPATH** 変数を変更して man パスディレクトリーを参照するように **manpage.sh** 短いスクリプトを使用します。

   ```
   export MANPATH="/opt/provider/software_collection/path/to/your/man_pages:${MANPATH}"
   ```

3. このファイルを Software Collection パッケージの spec ファイルに追加します。

   ```
   SOURCE2: %{?scl_prefix}manpage.sh
   ```

4. Software Collection パッケージの spec ファイルの **%install** セクションを調整して、このファイルをシステムの **/etc/profile.d/** ディレクトリーにインストールします。

   ```
   %install
   install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
   scl:%_sysconfdir}/profile.d/
   ```

## 3.8. SOFTWARE COLLECTION CRONJOB サポート

Software Collection を使用すると、専用サービスまたは cronjobs を使用して、システムで定期的なタスクを実行できます。専用サービスを使用する場合は、Software Collection 環境で initscripts の使用方法について「Software Collections でのサービス管理」を参照してください。

**手順3.8 cronjobs で定期的なタスクの実行**

1. 定期的なタスクを実行するために cronjobs を使用するには、Software Collection の **crontab** ファイルを、Software Collection の名前のある **/etc/cron.d** ディレクトリーに配置します。

   たとえば、以下のファイルを作成します。

   ```
   %{?scl_prefix}crontab
   ```

2. 以下の例のように、**crontab** ファイルの内容が標準の **crontab** ファイル形式に準拠することを確認します。

```
0 1 * * Sun root scl enable software_collection
'/opt/provider/software_collection/root/usr/bin/cron_job_name'
```

ここで、*software_collection* は Software Collection の名前で、**/opt/*provider*/*software_collection*/root/usr/bin/*cron_job_name*** は定期的に実行するコマンドです。

3. このファイルを Software Collection パッケージの spec ファイルに追加します。

```
SOURCE2: %{?scl_prefix}crontab
```

4. Software Collection パッケージの spec ファイルの **%install** セクションを調整して、システムディレクトリー **/etc/cron.d/** にインストールします。

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/cron.d/
```

## 3.9. SOFTWARE COLLECTION ログファイルのサポート

デフォルトでは、Software Collection にパッケージ化されているプログラムにより、その **/opt/*provider*/%{scl}/root/var/log/** ディレクトリーにログファイルが作成されます。

ログファイルにアクセスしやすく、管理を容易にするには、**_localstatedir** マクロを再定義する **nfsmountable** マクロを使用することが推奨されます。これにより、**/var/opt/*provider*/%{scl}/log/** ディレクトリーの下にログファイルが作成されます。これは、**/opt/*provider*/%{scl}** ファイルシステム階層外にあることになります。

たとえば、*mydaemon* サービスは、通常、ログファイルをベースシステムインストールの **/var/log/mydaemon/mydaemond.log** に保存します。*mydaemon* が *software_collection* Software Collection としてパッケージ化され、**nfsmountable** マクロが定義されている場合には、*software_collection* 内のログファイルへのパスは以下のようになります。

```
/var/opt/provider/software_collection/log/mydaemon/mydaemond.log
```

**nfsmountable** マクロの使用方法は、「NFS での Software Collections の使用」 を参照してください。

## 3.10. SOFTWARE COLLECTION LOGROTATE サポート

Software Collection または Software Collection に関連付けられたアプリケーションを使用して、**logrotate** プログラムでログファイルを管理できます。

**手順3.9 logrotate でログファイルの管理**

1. **logrotate** を使用してログファイルを管理するには、**logrotate** ジョブ **/etc/logrotate.d/** のシステムディレクトリーに、Software Collection のカスタム **logrotate** ファイルを置きます。

   たとえば、以下のファイルを作成します。

```
%{?scl_prefix}logrotate
```

2. **logrotate** ファイルの内容が、以下のように標準の **logrotate** ファイル形式に準拠することを確認します。

```
/opt/provider/software_collection/var/log/your_application_name.log {
    missingok
    notifempty
    size 30k
    yearly
    create 0600 root root
}
```

3. このファイルを Software Collection パッケージの spec ファイルに追加します。

```
SOURCE2: %{?scl_prefix}logrotate
```

4. Software Collection パッケージの spec ファイルの **%install** セクションを調整して、システムディレクトリー **/etc/logrotate.d/** にインストールします。

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/logrotate.d/
```

## 3.11. SOFTWARE COLLECTION /VAR/RUN/ ファイルのサポート

PID ファイルは通常 **/var/run/*package_name*/** ディレクトリーの下にあるファイルの 1 つの例です。PID ファイルを Software Collection にパッケージ化する場合は、**nfsmountable** マクロを使用し、以下のディレクトリーに PID ファイルを保存することが推奨されます。

```
/var/run/software_collection-package_name/
```

ここで、*software_collection* は Software Collection の名前で、 *package_name* は Software Collection に含まれるパッケージの名前になります。

この命名規則に従って、ベースシステムインストールとのファイルの競合を避ける一方で、Software Collection が PID ファイルの **tmpfs** ファイルシステムなどの **/var/run/** 機能を使用できるようにします。

**nfsmountable** マクロの使用方法は、「NFS での Software Collections の使用」 を参照してください。

## 3.12. SOFTWARE COLLECTION のロックファイルのサポート

デフォルトでは、Software Collection にパッケージ化されたプログラムは、**/opt/*provider*/%{scl}/root/var/lock/** ディレクトリーにロックファイルを作成します。

ロックファイルにアクセスし、管理しやすくするには、**_localstatedir** マクロを再定義する **nfsmountable** マクロを使用することが推奨されます。これにより、**/var/opt/*provider*/%{scl}/lock/** ディレクトリーの下にログファイルが作成されます。これは、**/opt/*provider*/%{scl}** ファイルシステム階層外にあることになります。

Software Collection にパッケージ化されているアプリケーションやサービスが **/var/opt/*provider*/%{scl}/lock/** ディレクトリーの下にあるロックを書き込むと、これらのアプリケーションとサービスは、システムバージョンと同時に実行できます (Software Collection のアプリケーションやサービスのリソースがシステムバージョンのリソースと競合することはありません)。

たとえば、ロックファイル **mylockfile.lock** は、通常ベースシステムインストールの **/var/lock/** ディレクトリーに作成されます。ロックファイルが *software_collection* Software Collection の一部で、**nfsmountable** マクロが定義されている場合は、*software_collection* のロックファイルへのパスは以下のようになります。

```
/var/opt/provider/software_collection/lock/mylockfile.lock
```

**nfsmountable** マクロの使用方法は、「NFS での Software Collections の使用」 を参照してください。

プログラムが継続的に実行されないようにする
該当するアプリケーションまたはサービスのシステムバージョンの実行中に、Software Collection のアプリケーションやサービスを実行しないようにする場合は、ロックを必要とするアプリケーションやサービスが、システムディレクトリー **/var/lock** にロックを書き込むことを確認してください。これにより、アプリケーションまたはサービスのロックファイルは上書きされません。ロックファイルの名前は変更されず、名前はシステムバージョンと同じままになります。

### 3.12.1. Software Collection SysV init Lock File のサポート

init スクリプトがサービスを開始すると、init スクリプトと同じ名前を持つ **/var/lock/subsys/** ディレクトリー内でロックファイルが作成されます。「Software Collections でのサービス管理」 で説明されているように、サービス名には Software Collection の接頭辞が含まれます。**/var/lock/subsys/** 以下でファイルの同じ命名規則を使用して、ロックファイル名がベースシステムのインストールと競合しないようにします。

## 3.13. SOFTWARE COLLECTION 設定ファイルのサポート

デフォルトでは、Software Collection の設定ファイルは **/opt/*provider*/%{scl}** ファイルシステム階層に保存されます。

設定ファイルにアクセスしやすく、管理を容易にするには、**_sysconfdir** マクロを再定義する **nfsmountable** マクロを使用することが推奨されます。これにより、設定ファイルは、**/etc/opt/*provider*/%{scl}/** ファイルシステム階層外にある **/opt/*provider*/%{scl}** ディレクトリーの下に作成されます。

たとえば、設定ファイル *example.conf* は、通常ベースシステムインストールの **/etc** ディレクトリーに保存されます。設定ファイルが *software_collection* Software Collection の一部で、**nfsmountable** マクロが定義されている場合には、*software_collection* 内の設定ファイルへのパスは以下のようになります。

```
/etc/opt/provider/software_collection/example.conf
```

**nfsmountable** マクロの使用方法は、「NFS での Software Collections の使用」 を参照してください。

## 3.14. SOFTWARE COLLECTION カーネルモジュールのサポート

Linux カーネルモジュールは通常、特定のバージョンの Linux カーネルに関連付けられているため、カーネルモジュールを Software Collection にパッケージ化する場合は注意してください。これは、更新されたバージョンの Linux カーネルがインストールされている場合に、Red Hat Enterprise Linux のパッケージ管理システムが、更新されたバージョンのカーネルモジュールを自動的に更新またはインストールしないためです。Software Collection へのカーネルモジュールのパッケージ化を容易にするには、以下の推奨事項を参照してください。以下の点を確認してください。

1. カーネルモジュールパッケージの名前には、カーネルバージョンが含まれます。

2. タグ **Requires** は、カーネルモジュールの spec ファイルにあります。これには、(**kernel-*version-revision*** 形式の) カーネルのバージョンおよびリビジョンが含まれます。

## 3.15. SOFTWARE COLLECTION SELINUX サポート

Software Collection は、別のディレクトリーに Software Collection パッケージをインストールするように設計されているため、SELinux が代替ディレクトリーを認識できるように、必要な SELinux ラベルを設定します。

Software Collection パッケージのファイルシステム階層が、対応する従来のパッケージのファイルシステム階層を省略した場合は、**semanage fcontext** および **restorecon** コマンドを実行して SELinux ラベルを設定できます。

たとえば、Software Collection パッケージの **/opt/provider/software_collection_1/root/usr/** ディレクトリーが従来のパッケージの **/usr/** ディレクトリーを省略する場合は、以下のように SELinux ラベルを設定します。

```
semanage fcontext -a -e /usr /opt/provider/software_collection_1/root/usr
```

```
restorecon -R -v /opt/provider/software_collection_1/root/usr
```

上記のコマンドは、**/usr/** ディレクトリー内のディレクトリーとファイルがすべて、**/opt/provider/software_collection_1/root/usr/** ディレクトリーにあるかのように SELinux によってラベル付けされるようにします。

### 3.15.1. Red Hat Enterprise Linux 7 での SELinux サポート

Red Hat Enterprise Linux 7 の Software Collection をパッケージ化する際に、以下を Software Collection メタパッケージの **%post** セクションに追加し、SELinux ラベルを設定します。

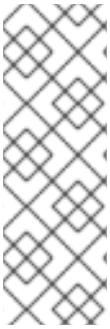
```
semanage fcontext -a -e /usr /opt/provider/software_collection_1/root/usr
```

```
restorecon -R -v /opt/provider/software_collection_1/root/usr
```

```
selinuxenabled && load_policy || :
```

最後のコマンドは、新たに作成された SELinux ポリシーが適切に読み込まれ、Software Collection のパッケージによりインストールされたファイルが、正しい SELinux コンテキストで作成されるようにします。メタパッケージでこのコマンドを使用すると、Software Collection のすべてのパッケージに **restorecon** コマンドを含める必要がありません。

この **semanage fcontext** コマンドは policycoreutils-python パッケージにより提供されるため、Software Collection メタパッケージの **Requires** に **policycoreutils-python** を追加することが重要です。



**注記**

Red Hat Enterprise Linux 7 では、サービスを起動する SELinux の機能が大幅に 変更 になりました。最も重要な点として、systemd サービスファイルで **scl enable ...** ラッパーを使用すると、**unconfined_service_t** コンテキストを使用してサービスが 制限のないプロセス として実行されます。このコンテキストには設計による移行ルールがないため、SELinux ポリシーにより示されるターゲットの SELinux コンテキストに移行することができません。つまり、**scl enable ...** は、開始するサービスが SELinux を使用して制限される場合は、Red Hat Enterprise Linux 7 では使用できません。

## 3.16. RED HAT ENTERPRISE LINUX 6 と 7 の相違点

Red Hat Enterprise Linux 7 の RPM Package Manager には、Red Hat Enterprise Linux 6 に含まれる旧バージョンの RPM Package Manager では提供されない多くの機能変更が含まれています。

本セクションでは、両方のシステムに Software Collection パッケージを構築する際に影響を受ける可能性のある変更の詳細を説明します。

ライブラリーサポートの相違点は、「Red Hat Enterprise Linux 7 での Software Collection Library のサポート」を参照してください。SELinux サポートの相違点は、「Red Hat Enterprise Linux 7 での SELinux サポート」に記載されています。

### 3.16.1. %license マクロ

**%license** マクロを使用すると、パッケージにインストールするライセンスファイルを指定できます。このマクロは、Red Hat Enterprise Linux 7 の RPM Package Manager でのみサポートされます。Red Hat Enterprise Linux 6 および 7 の両方で Software Collection パッケージを構築する場合は、以下のように Red Hat Enterprise Linux 6 の **%license** マクロを宣言します。

```
%{!?_licensedir:%global license %%doc}
```

### 3.16.2. ランタイムのサブパッケージ依存関係がない

Red Hat Enterprise Linux 7 では、**scl** ツールは、Software Collection runtime サブパッケージに必要な **Requires** を自動的に生成します。これは、Red Hat Enterprise Linux 6 では機能しません。Software Collection をそのシステム用に構築する場合は、各 Software Collection パッケージの runtime サブパッケージの依存関係を明示的に指定する必要があります。

```
Requires: %{?scl_prefix}runtime
```

### 3.16.3. scl-package() の Provides

Software Collection パッケージを構築すると、多くの **Provide: scl-package()** タグが生成されます。この目的は、構築されたパッケージを特定の Software Collection に属するものとして内部的に特定することです。タグの詳細を以下の表に示します。

**表3.2 Red Hat Enterprise Linux 7 の Provides**

| Software Collection パッケージ | Provides |
|---|---|
| **${software_collection_1}** | **scl-package(software_collection_1)** |

| Software Collection パッケージ | Provides |
| --- | --- |
| **${software_collection_1}-build** | **scl-package(software_collection_1)** |
| **${software_collection_1}-runtime** | **scl-package(software_collection_1)** |

Red Hat Enterprise Linux 6 には旧バージョンの RPM Package Manager が同梱されているため、Red Hat Enterprise Linux 6 で同じパッケージを構築すると、以下の表で説明する **Provide: scl-package()** タグは 1 つだけです。これは予想される動作であり、違いは **scl** ツールによって内部的に処理されます。

**表3.3 Red Hat Enterprise Linux 6 の Provide**

| Software Collection パッケージ | Provides |
| --- | --- |
| **${software_collection_1}** | **scl-package(software_collection_1)** |

このような内部生成された依存関係を使用して、特定の Software Collection に属するパッケージを一覧表示しないでください。Software Collection パッケージを適切に一覧表示する方法は「インストールされた Software Collections の一覧表示」を参照してください。

# 第4章 RED HAT SOFTWARE COLLECTIONS の再構築

本章では、Red Hat Software Collections オファリングの一部である Software Collections の一部を拡張する方法を説明します。

## 4.1. SCLDEVEL サブパッケージの提供

scldevel サブパッケージの目的は、多くの汎用マクロファイルを提供することにより、依存する Software Collections の作成プロセスを簡素化することです。パッケージャーは、既存の Software Collections を拡張するときにこれらのマクロファイルを使用します。scldevel は、Software Collection メタパッケージのサブパッケージとして提供されます。

### 4.1.1. scldevel サブパッケージの作成

次のセクションでは、ruby193 および ruby200 の 2 つの例の Ruby Software Collections の scldevel サブパッケージを作成する方法を説明します。

**手順4.1 独自の scldevel サブパッケージを提供**

1. Software Collection のメタパッケージで、名前、要約、および説明を定義して scldevel サブパッケージを追加します。

```
%package scldevel
Summary: Package shipping development files for %scl
Provides: scldevel(%{scl_name_base})

%description scldevel
Package shipping development files, especially useful for development of
packages depending on %scl Software Collection.
```

依存する Software Collections のパッケージを構築する際には、仮想 **Provides: scldevel(%{scl_name_base})** を使用することが推奨されます。これにより、以下の手順で指定されているように、**%{scl_name_base}** Software Collection とそのマクロのバージョンが利用できるようになります。

2. Software Collection のメタパッケージの **%install** セクションで、scldevel サブパッケージに含まれる **macros.%{scl_name_base}-scldevel** ファイルを作成し、以下を追加します。

```
cat >> %{buildroot}%{_root_sysconfdir}/rpm/macros.%{scl_name_base}-scldevel << EOF
%%scl_%{scl_name_base} %{scl}
%%scl_prefix_%{scl_name_base} %{scl_prefix}
EOF
```

同じ **%{scl_name_base}** 名を共有するすべての Software Collections の間で、提供される **macros.%{scl_name_base}-scldevel** ファイルが競合する必要があることに注意してください。これにより、複数のバージョンの **%{scl_name_base}** Software Collections のインストールは許可されません。たとえば、ruby193-scldevel サブパッケージがインストールされている場合は、ruby200-scldevel サブパッケージをインストールすることができません。

### 4.1.2. 依存する Software Collection での scldevel サブパッケージの使用

ruby200 Software Collection に依存する Software Collection で scldevel サブパッケージを使用するには、以下のように、依存する Software Collection のメタパッケージを更新します。

**手順4.2 依存するソフトウェアコレクションで独自の scldevel サブパッケージの使用**

1. メタパッケージの spec ファイルの最初に以下を追加することを検討してください。

```
%{!?scl_ruby:%global scl_ruby ruby200}
%{!?scl_prefix_ruby:%global scl_prefix_ruby %{scl_ruby}-}
```

これら 2 行は任意です。これは、依存する Software Collection が ruby200 Software Collection に依存するように設計されている視覚的なヒントとしてのみ意図されています。ビルドルートで利用できる scldevel サブパッケージが他にない場合は、ruby200-scldevel サブパッケージがビルド要件として使用されます。

これらの行を以下の行に置き換えることができます。

```
%{?scl_prefix_ruby}
```

2. 以下のビルド要件をメタパッケージに追加します。

```
BuildRequires: %{scl_prefix_ruby}scldevel
```

このビルド要件を指定すると、scldevel サブパッケージがビルドルートにあり、デフォルト値が使用されていないことを確認します。このパッケージを省略すると、後続のパッケージのビルド時間で要求が破損する可能性があります。

3. メタパッケージの spec ファイルの **%package runtime** に以下の行が含まれていることを確認します。

```
%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils
Requires: %{scl_prefix_ruby}runtime
```

4. メタパッケージの spec ファイルの **%package build** に、以下の行を含めることを検討してください。

```
%package build
Summary: Package shipping basic build configuration
Requires: %{scl_prefix_ruby}scldevel
```

**Requires: %{scl_prefix_ruby}scldevel** を指定すると、Software Collection のすべてのパッケージでマクロが利用可能になります。

これを追加することは、依存する Software Collection のパッケージが scldevel サブパッケージが提供するマクロを使用するなど、特定のユースケースで **Requires** のみ有効であることに注意してください。

## 4.2. PYTHON27 および RH-PYTHON35 SOFTWARE COLLECTIONS の拡張

本セクションでは、依存する Software Collection を作成して python27 および rh-python35 の Software Collections を拡張する方法を説明します。

Red Hat Software Collections 3.8 では、**scl** ツールはマクロ **%scl_package_override()** をサポートするように拡張されました。これにより、依存する Software Collection のパッケージ化が容易になりました。

### 4.2.1. vt191 Software Collection

以下は、依存する Software Collection を構築する例です。Software Collection には **vt191** という名前が付けられ、versiontools Python パッケージバージョン 1.9.1 が同梱されています。

vt191 Software Collection のメタパッケージでは、以下の点に注意してください。

- vt191 Software Collection メタパッケージには、以下のビルド依存関係が設定されています。

  ```
  BuildRequires: %{scl_prefix_python}scldevel
  ```

  これは、たとえば python27-scldevel などに展開されます。

  python27-scldevel サブパッケージには、2 つの重要なマクロ **%scl_python** および **%scl_prefix_python** が同梱されています。このマクロは、メタパッケージの spec ファイルの上部に定義されていることに注意してください。定義は必須ではありませんが、vt191 Software Collection が python27 Software Collection に構築されるように設計されているという視覚的なヒントを提供します。フォールバック値としても機能します。

- **site-packages** ディレクトリーを正しく設定するには、**%python27python_sitelib** マクロの値を使用して、**python27** を **vt191** に置き換えます。Software Collection を別のプロバイダー (例: **/opt/myorganization/** instead of **/opt/rh/**) を使用して構築する場合は、これらも変更する必要があります。

  **重要**

  **/opt/rh/** プロバイダーは Red Hat が提供する Software Collections のインストールに使用するため、競合の可能性を回避するために別のプロバイダーを使用することを強く推奨します。詳細は、「Software Collection Root ディレクトリー」を参照してください。

- vt191-build サブパッケージには、以下の依存関係セットがあります。

  ```
  Requires: %{scl_prefix_python}scldevel
  ```

  これは、たとえば python27-scldevel などに展開されます。この依存関係の目的は、vt191 Software Collection のパッケージを構築する際にマクロが常に存在することを確認することです。

- vt191 Software Collection の **enable** スクリプトレットは、以下の行を使用します。

  ```
  . scl_source enable %{scl_python}
  ```

  行の先頭にあるドットに注意してください。この行では、vt191 Software Collection が起動すると、Python Software Collection が暗黙的に起動されます。これにより、ユーザーは **scl enable python27 vt191 *command*** のかわりに、**scl enable vt191 *command*** のみを入力して、Software Collection 環境で *command* を実行できます。

- マクロファイル **macros.vt191-config** は、**%__os_install_post**、Python 依存関係ジェネレーター、および他のパッケージの spec ファイルで使用される特定の Python 固有のマクロを適切に上書きする **%scl_package_override** を呼び出します。

```
# define name of the scl
%global scl vt191
%scl_package %scl

# Defaults for the values for the python27/rh-python35 Software Collection. These
# will be used when python27-scldevel (or rh-python35-scldevel) is not in the
# build root
%{!?scl_python:%global scl_python python27}
%{!?scl_no_vendor:%global scl_no_vendor python27}
%{!?scl_prefix_python:%global scl_prefix_python %{scl_python}-}

# Only for this build, you need to override default __os_install_post,
# because the default one would find /opt/.../lib/python2.7/ and try
# to bytecompile with the system /usr/bin/python2.7
%global __os_install_post %{%{scl_no_vendor}_os_install_post}
# Similarly, override __python_requires for automatic dependency generator
%global __python_requires %{%{scl_no_vendor}_python_requires}

# The directory for site packages for this Software Collection
%global vt191_sitelib %(echo %{python27python_sitelib} | sed 's|%{scl_python}|%{scl}|')

Summary: Package that installs %scl
Name: %scl_name
Version: 1
Release: 1%{?dist}
License: GPLv2+
BuildRequires: scl-utils-build
# Always make sure that there is the python27-sclbuild (or rh-python35-sclbuild)
# package in the build root
BuildRequires: %{scl_prefix_python}scldevel
# Require python27-python-devel, you will need macros from that package
BuildRequires: %{scl_prefix_python}python-devel
Requires: %{scl_prefix}python-versiontools

%description
This is the main package for %scl Software Collection.

%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils
Requires: %{scl_prefix_python}runtime

%description runtime
Package shipping essential scripts to work with %scl Software Collection.

%package build
Summary: Package shipping basic build configuration
Requires: scl-utils-build
# Require python27-scldevel (or rh-python35-scldevel) so that there is always access
# to the %%scl_python and %%scl_prefix_python macros in builds for this Software
# Collection
Requires: %{scl_prefix_python}scldevel
```

```
%description build
Package shipping essential configuration macros to build %scl Software Collection.

%prep
%setup -c -T

%install
%scl_install

# Create the enable scriptlet that:
# - Adds an additional load path for the Python interpreter.
# - Runs scl_source so that you can run:
#     scl enable vt191 "bash"
#   instead of:
#     scl enable python27 vt191 "bash"

cat >> %{buildroot}%{_scl_scripts}/enable << EOF
. scl_source enable %{scl_python}
export PYTHONPATH="%{vt191_sitelib}\${PYTHONPATH:+:\${PYTHONPATH}}"
EOF

mkdir -p %{buildroot}%{vt191_sitelib}

# - Enable Software Collection-specific bytecompilation macros from
#   the python27-python-devel package.
# - Also override the %%python_sitelib macro to point to the vt191 Software
#   Collection.
# - If you have architecture-dependent packages, you will also need to override
#   the %%python_sitearch macro.

cat >> %{buildroot}%{_root_sysconfdir}/rpm/macros.%{scl}-config << EOF
%%scl_package_override() %%{expand:%{?python27_os_install_post:%%global __os_install_post
%%python27_os_install_post}
%%global __python_requires %%python27_python_requires
%%global __python_provides %%python27_python_provides
%%global __python %python27__python
%%global python_sitelib %vt191_sitelib
%%global python2_sitelib %vt191_sitelib
}
EOF

%files

%files runtime -f filelist
%scl_files
%vt191_sitelib

%files build
%{_root_sysconfdir}/rpm/macros.%{scl}-config

%changelog
* Wed Jan 22 2014 John Doe <jdoe@example.com> - 1-1
- Initial package.
```

### 4.2.2. python-versiontools パッケージ

以下は、python-versiontools パッケージの spec ファイルのコメントされた例です。spec ファイルでは、以下の点に注意してください。

- **BuildRequires** タグの接頭辞は、**%{scl_prefix}** ではなく、**%{?scl_prefix_python}** になります。
- **%install** セクションは、**--install-purelib** を明示的に指定します。

```
%{?scl:%scl_package python-versiontools}
%{!?scl:%global pkg_name %{name}}

%global pypi_name versiontools

Name:           %{?scl_prefix}python-versiontools
Version:        1.9.1
Release:        1%{?dist}
Summary:        Smart replacement for plain tuple used in __version__

License:        LGPLv3
URL:            https://launchpad.net/versiontools
Source0:        http://pypi.python.org/packages/source/v/versiontools/versiontools-1.9.1.tar.gz

BuildArch:      noarch
BuildRequires:  %{?scl_prefix_python}python-devel
BuildRequires:  %{?scl_prefix_python}python-setuptools
%{?scl:BuildRequires: %{scl}-build %{scl}-runtime}
%{?scl:Requires: %{scl}-runtime}

%description
Smart replacement for plain tuple used in __version__

%prep
%setup -q -n %{pypi_name}-%{version}

%build
%{?scl:scl enable %{scl} "}
%{__python} setup.py build
%{?scl:"}

%install
# Explicitly specify --install-purelib %{python_sitelib}, which is now overriden
# to point to vt191, otherwise Python will try to install into the python27
# Software Collection site-packages directory
%{?scl:scl enable %{scl} "}
%{__python} setup.py install -O1 --skip-build --root %{buildroot} --install-purelib %{python_sitelib}
%{?scl:"}

%files
%{python_sitelib}/%{pypi_name}*

%changelog
* Wed Jan 22 2014 John Doe <jdoe@example.com> - 1.9.1-1
- Built for vt191 SCL.
```

### 4.2.3. vt191 Software Collection の構築

vt191 Software Collection を構築するには、以下を実行します。

1. python27 Software Collection に含まれる python27-scldevel サブパッケージおよび python27-python-devel サブパッケージをインストールします。
2. **vt191.spec** を構築して、vt191-runtime パッケージおよび vt191-build パッケージをインストールします。
3. versiontools のビルド要件である python27-python-setuptools パッケージをインストールします。
4. **python-versiontools.spec** をビルドします。

### 4.2.4. vt191 Software Collection のテスト

vt191 Software Collection をテストするには、以下を実行します。

1. vt191-python-versiontools パッケージをインストールします。
2. 次のコマンドを実行します。

```
$ scl enable vt191 "python -c 'import versiontools; print(versiontools.__file__)'"
```

3. 出力に以下の行が含まれることを確認します。

```
/opt/rh/vt191/root/usr/lib/python2.7/site-packages/versiontools/__init__.pyc
```

パスのプロバイダー **rh** は、**%_scl_prefix** マクロの再定義により異なる可能性があることに注意してください。詳細は、「Software Collection Root ディレクトリー」 を参照してください。

## 4.3. RH-RUBY23 SOFTWARE COLLECTION の拡張

Red Hat Software Collections 3.8 では、依存パッケージを追加して rh-ruby23 Software Collection を拡張できます。rh-ruby23 Software Collection が提供する Ruby 2.3 に構築される Ruby on Rails 4.2 (rh-ror42) Software Collection は、このような拡張機能の 1 つです。

本セクションでは、rh-ror42 Software Collection に含まれる rh-ror42-rubygem-bcrypt メタパッケージと rh-ror42 パッケージの詳細情報を提供します。

### 4.3.1. rh-ror42 Software Collection

このセクションでは、rh-ror42 Software Collection の Ruby on Rails 4.2 メタパッケージのコメントされた例を説明します。rh-ror42 Software Collection は、rh-ruby23 Software Collection によって異なります。

rh-ror42 Software Collection のメタパッケージの例を以下に示します。

- rh-ror42 Software Collection の spec ファイルには、以下のビルド依存関係セットがあります。

```
BuildRequires: %{scl_prefix_ruby}scldevel
BuildRequires: %{scl_prefix_ruby}rubygems-devel
```

これは、rh-ruby23-scldevel、rh-ruby23-rubygems-devel などに展開されます。

この rh-ruby23-scldevel サブパッケージには、重要なマクロが 2 つ (**%scl_ruby** および **%scl_prefix_ruby**) あります。rh-ruby23-scldevel サブパッケージはビルドルートで利用できる必要があります。Ruby Software Collections が複数ある場合は、rh-ruby23-scldevel は、利用可能な Software Collections をどれを使用するかを決定します。

**%scl_ruby** および **%scl_prefix_ruby** マクロは、spec ファイルの上部でも定義されることに注意してください。定義は必須ではありませんが、rh-ror42 Software Collection が rh-ruby23 Software Collection に構築されるように設計されているという視覚的なヒントを提供します。フォールバック値としても機能します。

- rh-ror42-runtime サブパッケージは、依存する Software Collection の runtime サブパッケージに依存する必要があります。この依存関係は以下のように指定されます。

```
%package runtime
Requires: %{scl_prefix_ruby}runtime
```

パッケージが rh-ruby23 Software Collection に対して構築されると、rh-ruby23-runtimeに展開されます。

- rh-ror42-build サブパッケージは、依存する Software Collection の scldevel サブパッケージに依存する必要があります。これは、この Software Collection の他のすべてのパッケージが同じマクロを定義しているため、同じ Ruby バージョンに対して構築されるようにするためです。

```
%package build
Requires: %{scl_prefix_ruby}scldevel
```

rh-ruby23 Software Collection の場合には、rh-ruby23-scldevelに展開されます。

- rh-ror42 Software Collection の **enable** スクリプトレットには、以下の行が含まれます。

```
. scl_source enable %{scl_ruby}
```

行の先頭にあるドットに注意してください。この行では、rh-ror42 Software Collection が起動すると、Ruby Software Collection が暗黙的に開始するため、ユーザーは **scl enable rh-ruby23 rh-ror42 *command*** の代わりに、**scl enable rh-ror42 *command*** のみを実行して、Software Collection 環境で *command* を実行できます。

- この rh-ror42-scldevel サブパッケージは、rh-ror42 Software Collection を拡張する Software Collection を構築する必要がある場合は利用できるようにすることができます。このパッケージは、rh-ror42 Software Collection を拡張するために使用できる **%{scl_ror}** および **%{scl_prefix_ror}** マクロを提供します。

- rh-ror42 Software Collection の gems は別のルートディレクトリー構造にインストールされるため、rubygems ディレクトリーの正しい所有権が設定されていることを確認する必要があります。これは、スニペットを使用してファイル一覧 rubygems_filesystem.list を生成することで行います。

root ファイルシステムに置かれた場合は、別の runtime パッケージが所有するディレクトリーをすべて所有するように設定することが推奨されます。rh-ror42 Software Collection の場合におけるそのようなディレクトリーの 1 つが Rubygem ディレクトリー構造です。

```
%global scl_name_prefix rh-
%global scl_name_base ror
%global scl_name_version 41

%global scl %{scl_name_prefix}%{scl_name_base}%{scl_name_version}

# Fallback to rh-ruby23. rh-ruby23-scldevel is unlikely to be available in
# the build root.
%{!?scl_ruby:%global scl_ruby rh-ruby23}
%{!?scl_prefix_ruby:%global scl_prefix_ruby %{scl_ruby}-}

# Do not produce empty debuginfo package.
%global debug_package %{nil}

# Support SCL over NFS.
%global nfsmountable 1

%{!?install_scl: %global install_scl 1}

%scl_package %scl

Summary: Package that installs %scl
Name: %scl_name
Version: 2.0
Release: 5%{?dist}
License: GPLv2+

%if 0%{?install_scl}
Requires: %{scl_prefix}rubygem-therubyracer
Requires: %{scl_prefix}rubygem-sqlite3
Requires: %{scl_prefix}rubygem-rails
Requires: %{scl_prefix}rubygem-sass-rails
Requires: %{scl_prefix}rubygem-coffee-rails
Requires: %{scl_prefix}rubygem-jquery-rails
Requires: %{scl_prefix}rubygem-sdoc
Requires: %{scl_prefix}rubygem-turbolinks
Requires: %{scl_prefix}rubygem-bcrypt
Requires: %{scl_prefix}rubygem-uglifier
Requires: %{scl_prefix}rubygem-jbuilder
Requires: %{scl_prefix}rubygem-spring
%endif
BuildRequires: help2man
BuildRequires: scl-utils-build
BuildRequires: %{scl_prefix_ruby}scldevel
BuildRequires: %{scl_prefix_ruby}rubygems-devel

%description
This is the main package for %scl Software Collection.

%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils
# The enable scriptlet depends on the ruby executable.
Requires: %{scl_prefix_ruby}ruby

%description runtime
```

```
Package shipping essential scripts to work with %scl Software Collection.

%package build
Summary: Package shipping basic build configuration
Requires: scl-utils-build
Requires: %{scl_runtime}
Requires: %{scl_prefix_ruby}scldevel

%description build
Package shipping essential configuration macros to build %scl Software Collection.

%package scldevel
Summary: Package shipping development files for %scl
Provides: scldevel(%{scl_name_base})

%description scldevel
Package shipping development files, especially usefull for development of
packages depending on %scl Software Collection.

%prep
%setup -c -T

%install
%scl_install

cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export PATH="%{_bindir}:%{_sbindir}\${PATH:+:\${PATH}}"
export LD_LIBRARY_PATH="%{_libdir}\${LD_LIBRARY_PATH:+:\${LD_LIBRARY_PATH}}"
export MANPATH="%{_mandir}:\${MANPATH:-}"
export PKG_CONFIG_PATH="%
{_libdir}/pkgconfig\${PKG_CONFIG_PATH:+:\${PKG_CONFIG_PATH}}"
export GEM_PATH="\${GEM_PATH:=%{gem_dir}:\`scl enable %{scl_ruby} -- ruby -e "print
Gem.path.join(':')"\`}"

. scl_source enable %{scl_ruby}
EOF

cat >> %{buildroot}%{_root_sysconfdir}/rpm/macros.%{scl_name_base}-scldevel << EOF
%%scl_%{scl_name_base} %{scl}
%%scl_prefix_%{scl_name_base} %{scl_prefix}
EOF


scl enable %{scl_ruby} - << \EOF
set -e

# Fake rh-ror42 Software Collection environment.
GEM_PATH=%{gem_dir}:`ruby -e "print Gem.path.join(':')"` \
X_SCLS=%{scl} \
ruby -rfileutils > rubygems_filesystem.list << \EOR
  # Create the RubyGems file system.
  Gem.ensure_gem_subdirectories '%{buildroot}%{gem_dir}'
  FileUtils.mkdir_p File.join '%{buildroot}', Gem.default_ext_dir_for('%{gem_dir}')

  # Output the relevant directories.
  Gem.default_dirs['%{scl}_system'.to_sym].each { |k, p| puts p }
```

```
EOR
EOF

%files

%files runtime -f rubygems_filesystem.list
%scl_files

%files build
%{_root_sysconfdir}/rpm/macros.%{scl}-config

%files scldevel
%{_root_sysconfdir}/rpm/macros.%{scl_name_base}-scldevel


%changelog
* Thu Jan 16 2015 John Doe <jdoe@example.com> - 1-1
- Initial package.
```

### 4.3.2. rh-ror42-rubygem-bcrypt パッケージ

以下は、rh-ror42-rubygem-bcrypt パッケージの spec ファイルのコメントされた例です。このパッケージは、bcrypt Ruby gem を提供します。bcrypt の詳細は、以下の Web サイトを参照してください。

- http://rubygems.org/gems/bcrypt-ruby

rh-ror42-rubygem-bcrypt パッケージの spec ファイルと通常の Software Collection の spec ファイルの大きな違いは、以下のとおりです。

- **BuildRequires** タグの接頭辞は、**%{scl_prefix}** ではなく、**%{?scl_prefix_ruby}** になります。

```
%{?scl:%scl_package rubygem-%{gem_name}}
%{!?scl:%global pkg_name %{name}}

%global gem_name bcrypt

Summary: Wrapper around bcrypt() password hashing algorithm
Name: %{?scl_prefix}rubygem-%{gem_name}
Version: 3.1.9
Release: 2%{?dist}
Group: Development/Languages
# ext/* - Public Domain
# spec/TestBCrypt.java - ISC
License: MIT and Public Domain and ISC
URL: https://github.com/codahale/bcrypt-ruby
Source0: http://rubygems.org/downloads/%{gem_name}-%{version}.gem
Requires: %{?scl_prefix_ruby}ruby(release)
Requires: %{?scl_prefix_ruby}ruby(rubygems)
BuildRequires: %{?scl_prefix_ruby}rubygems-devel
BuildRequires: %{?scl_prefix_ruby}ruby-devel
BuildRequires: %{?scl_prefix}rubygem(rspec)
Provides: %{?scl_prefix}rubygem(bcrypt) = %{version}

%description
```

```
bcrypt() is a sophisticated and secure hash algorithm designed by The
OpenBSD project for hashing passwords. bcrypt provides a simple,
humane wrapper for safely handling passwords.

%package doc
Summary: Documentation for %{pkg_name}
Group: Documentation
Requires: %{?scl_prefix}%{pkg_name} = %{version}-%{release}

%description doc
Documentation for %{pkg_name}.

%prep
%setup -n %{pkg_name}-%{version} -q -c -T
%{?scl:scl enable %{scl} - << \EOF}
%gem_install -n %{SOURCE0}
%{?scl:EOF}

%build

%install
mkdir -p %{buildroot}%{gem_dir}
cp -pa .%{gem_dir}/* \
        %{buildroot}%{gem_dir}/

mkdir -p %{buildroot}%{gem_extdir_mri}
cp -pa .%{gem_extdir_mri}/* %{buildroot}%{gem_extdir_mri}/

# Prevent a symlink with an invalid target in -debuginfo (BZ#878863).
rm -rf %{buildroot}%{gem_instdir}/ext/

%check
%{?scl:scl enable %{scl} - << \EOF}
pushd .%{gem_instdir}
# 2 failutes due to old RSpec
# https://github.com/rspec/rspec-expectations/pull/284
rspec -I$(dirs +1)%{gem_extdir_mri} spec |grep '34 examples, 2 failures' || exit 1
popd
%{?scl:EOF}

%files
%dir %{gem_instdir}
%exclude %{gem_instdir}/.*
%{gem_libdir}
%{gem_extdir_mri}
%exclude %{gem_cache}
%{gem_spec}
%doc %{gem_instdir}/COPYING

%files doc
%doc %{gem_docdir}
%doc %{gem_instdir}/README.md
%doc %{gem_instdir}/CHANGELOG
%{gem_instdir}/Rakefile
%{gem_instdir}/Gemfile*
%{gem_instdir}/%{gem_name}.gemspec
```

```
%{gem_instdir}/spec


%changelog
* Fri Mar 21 2015 John Doe <jdoe@example.com> - 3.1.2-4
- Initial package.
```

### 4.3.3. rh-ror42 Software Collection の構築

rh-ror42 Software Collection を構築するには、以下を実行します。

1. rh-ruby23 Software Collection に含まれる rh-ruby23-scldevel サブパッケージをインストールします。
2. **rh-ror42.spec** を構築して、ror42-runtime パッケージおよび ror42-build パッケージをインストールします。
3. **rubygem-bcrypt.spec** をビルドします。

### 4.3.4. rh-ror42 Software Collection のテスト

rh-ror42 Software Collection をテストするには、以下を実行します。

1. rh-ror42-rubygem-bcrypt パッケージをインストールします。
2. 次のコマンドを実行します。

```
$ scl enable rh-ror42 -- ruby -r bcrypt -e "puts BCrypt::Password.create('my password')"
```

3. 出力に以下の行が含まれることを確認します。

```
$2a$10$s./ReniLY.wXPHVBQ9npoeyZf5KzywfpvI5lhjG6Ams3u0hKqwVbW
```

## 4.4. RH-PERL524 SOFTWARE COLLECTION の拡張

本セクションでは、独自の依存する Software Collection を構築して、rh-perl524 Software Collection を拡張する方法を説明します。

**重要**

本項で説明する例は、rh-perl524 Software Collection を以下のパッケージで拡張する場合にのみ期待どおりに機能します。

- Perl モジュールを提供しないでください。
- rh-perl524 Software Collection が提供する Perl モジュールのみに依存します。

### 4.4.1. h2m144 Software Collection

本セクションは、依存する Software Collection のメタパッケージに関するコメントされた例を説明します。依存する Software Collection は h2m144 という名前で、help2man Perl パッケージバージョン 1.44.1 が含まれています。h2m144 Software Collection は、rh-perl524 Software Collection によって異なります。

h2m144 Software Collection のメタパッケージでは、以下の点に注意してください。

- h2m144 Software Collection メタパッケージには、以下のビルド依存関係セットがあります。

  ```
  BuildRequires: %{scl_prefix_perl}scldevel
  ```

  これは、rh-perl524-scldevel に展開されます。

  rh-perl524-scldevel サブパッケージには 2 つの重要なマクロ (**%scl_perl** および **%scl_prefix_perl**) が含まれます。Perl 依存関係ジェネレーターも提供します。マクロは、メタパッケージの spec ファイルの上部に定義されていることに注意してください。定義は必須ではありませんが、h2m144 Software Collection が rh-perl524 Software Collection に構築されるように設計されているという視覚的なヒントを提供します。フォールバック値としても機能します。

- h2m144-build サブパッケージには、以下の依存関係セットがあります。

  ```
  Requires: %{scl_prefix_perl}scldevel
  ```

  これは、rh-perl524-scldevel に展開されます。この依存関係の目的は、h2m144 Software Collection のパッケージを構築する際にマクロと依存関係ジェネレーターを常に存在させることです。

- h2m144 Software Collection の **enable** スクリプトレットには、以下の行が含まれます。

  ```
  . scl_source enable %{scl_perl}
  ```

  行の先頭にあるドットに注意してください。この行では、h2m144 Software Collection が起動すると Perl Software Collection が暗黙的に起動されます。これにより、ユーザーは **scl enable rh-perl524 h2m144 *command*** のかわりに、**scl enable h2m144 *command*** だけを実行して Software Collection 環境で *command* を実行することができます。

- マクロファイル **macros.h2m144-config** は Perl 依存関係ジェネレーターと、他のパッケージの spec ファイルで使用される特定の Perl 固有のマクロを呼び出します。

```
%global scl h2m144
%scl_package %scl

# Default values for the rh-perl524 Software Collection. These
# will be used when rh-perl524-scldevel is not in the build root.
%{!?scl_perl:%global scl_perl rh-perl524}
%{!?scl_prefix_perl:%global scl_prefix_perl %{scl_perl}-}

# Only for this build, override __perl_requires for the automatic dependency
# generator.
%global __perl_requires /usr/lib/rpm/perl.req.stack

Summary: Package that installs %scl
Name:    %scl_name
Version: 1
Release: 1%{?dist}
License: GPLv2+
BuildRequires: scl-utils-build
# Always make sure that there is the rh-perl524-scldevel
# package in the build root.
```

```
BuildRequires: %{scl_prefix_perl}scldevel
# Require rh-perl524-perl-macros; you will need macros from that package.
BuildRequires: %{scl_prefix_perl}perl-macros
Requires: %{scl_prefix}help2man

%description
This is the main package for %scl Software Collection.

%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils
Requires: %{scl_prefix_perl}runtime

%description runtime
Package shipping essential scripts to work with %scl Software Collection.

%package build
Summary: Package shipping basic build configuration
Requires: scl-utils-build
# Require rh-perl524-scldevel so that there is always access to the %%scl_perl
# and %%scl_prefix_perl macros in builds for this Software Collection.
Requires: %{scl_prefix_perl}scldevel

%description build
Package shipping essential configuration macros to build %scl Software Collection.

%prep
%setup -c -T

%build

%install
%scl_install

# Create the enable scriptlet that:
# - Adds an additional load path for the Perl interpreter.
# - Runs scl_source so that you can run:
#     scl enable h2m144 'bash'
#   instead of:
#     scl enable rh-perl524 h2m144 'bash'

cat >> %{buildroot}%{_scl_scripts}/enable << EOF
. scl_source enable %{scl_perl}
export PATH="%{_bindir}:%{_sbindir}\${PATH:+:\${PATH}}"
export MANPATH="%{_mandir}:\${MANPATH:-}"
EOF

cat >> %{buildroot}%{_root_sysconfdir}/rpm/macros.%{scl}-config << EOF
%%scl_package_override() %%{expand:%%global __perl_requires /usr/lib/rpm/perl.req.stack
%%global __perl_provides /usr/lib/rpm/perl.prov.stack
%%global __perl %{_scl_prefix}/%{scl_perl}/root/usr/bin/perl
}
EOF

%files
```

```
%files runtime -f filelist
%scl_files

%files build
%{_root_sysconfdir}/rpm/macros.%{scl}-config

%changelog
* Tue Apr 22 2014 John Doe <jdoe@example.com> - 1-1
- Initial package.
```

### 4.4.2. help2man パッケージ

以下は、help2man パッケージの spec ファイルのコメントされた例です。spec ファイルでは、以下の点に注意してください。

- **BuildRequires** タグの接頭辞は、**%{scl_prefix}** ではなく、**%{?scl_prefix_perl}** になります。

```
%{?scl:%scl_package help2man}
%{!?scl:%global pkg_name %{name}}

# Supported build option:
#
# --with nls ... build this package with --enable-nls
%bcond_with nls

Name:           %{?scl_prefix}help2man
Summary:        Create simple man pages from --help output
Version:        1.44.1
Release:        1%{?dist}
Group:          Development/Tools
License:        GPLv3+
URL:            http://www.gnu.org/software/help2man
Source:         ftp://ftp.gnu.org/gnu/help2man/help2man-%{version}.tar.xz
%{!?with_nls:BuildArch: noarch}

BuildRequires:  %{?scl_prefix_perl}perl(Getopt::Long)
BuildRequires:  %{?scl_prefix_perl}perl(POSIX)
BuildRequires:  %{?scl_prefix_perl}perl(Text::ParseWords)
BuildRequires:  %{?scl_prefix_perl}perl(Text::Tabs)
BuildRequires:  %{?scl_prefix_perl}perl(strict)
%{?with_nls:BuildRequires: %{?scl_prefix_perl}perl(Locale::gettext) /usr/bin/msgfmt}
%{?with_nls:BuildRequires: %{?scl_prefix_perl}perl(Encode)}
%{?with_nls:BuildRequires: %{?scl_prefix_perl}perl(I18N::Langinfo)}
Requires:   %{?scl_prefix_perl}perl(:MODULE_COMPAT_%(%{?scl:scl enable %{scl_perl} '}eval
"`perl -V:version`"; echo $version%{?scl:'}))

Requires(post): /sbin/install-info
Requires(preun): /sbin/install-info

%description
help2man is a script to create simple man pages from the --help and
--version output of programs.

Since most GNU documentation is now in info format, this provides a
way to generate a placeholder man page pointing to that resource while
```

```
still providing some useful information.

%prep
%setup -q -n help2man-%{version}

%build
%configure --%{!?with_nls:disable}%{?with_nls:enable}-nls --libdir=%{_libdir}/help2man
%{?scl:scl enable %{scl} "}
make %{?_smp_mflags}
%{?scl:"}

%install
%{?scl:scl enable %{scl} "}
make install_l10n DESTDIR=$RPM_BUILD_ROOT
%{?scl:"}
%{?scl:scl enable %{scl} "}
make install DESTDIR=$RPM_BUILD_ROOT
%{?scl:"}
%find_lang %pkg_name --with-man

%post
/sbin/install-info %{_infodir}/help2man.info %{_infodir}/dir 2>/dev/null || :

%preun
if [ $1 -eq 0 ]; then
  /sbin/install-info --delete %{_infodir}/help2man.info \
    %{_infodir}/dir 2>/dev/null || :
fi

%files -f %pkg_name.lang
%doc README NEWS THANKS COPYING
%{_bindir}/help2man
%{_infodir}/*
%{_mandir}/man1/*

%if %{with nls}
%{_libdir}/help2man
%endif

%changelog
* Tue Apr 22 2014 John Doe <jdoe@example.com> - 1.44.1-1
- Built for h2m144 SCL.
```

### 4.4.3. h2m144 Software Collection の構築

h2m144 Software Collection を構築するには、以下を実行します。

1. perl524 Software Collection に含まれる rh-perl524-scldevel パッケージおよび rh-perl524-perl-macros パッケージをインストールします。

2. h2m144.spec を構築して、h2m144-runtime パッケージおよび h2m144-build パッケージをインストールします。

3. help2man のすべてのビルド要件である rh-perl524-perl パッケージ、rh-perl524-perl-Text-ParseWords パッケージ、および rh-perl524-perl-Getopt-Long パッケージをインストールします。

4. **help2man.spec** をビルドします。

### 4.4.4. h2m144 Software Collection のテスト

h2m144 Software Collection をテストするには、以下を実行します。

1. h2m144-help2man パッケージをインストールします。
2. 次のコマンドを実行します。

```
$ scl enable h2m144 'help2man bash'
```

3. 出力が以下の行と類似することが確認されます。

```
.\" DO NOT MODIFY THIS FILE!  It was generated by help2man 1.44.1.
.TH BASH, "1" "April 2014" "bash, version 4.1.2(1)-release (x86_64-redhat-linux-gnu)" "User
Commands"
.SH NAME
bash, \- manual page for bash, version 4.1.2(1)-release (x86_64-redhat-linux-gnu)
.SH SYNOPSIS
.B bash
[\fIGNU long option\fR] [\fIoption\fR] ...
.SH DESCRIPTION
GNU bash, version 4.1.2(1)\-release\-(x86_64\-redhat\-linux\-gnu)
.IP
bash [GNU long option] [option] script\-file ...
.SS "GNU long options:"
.HP
\fB\-\-debug\fR
```

# 第5章 SOFTWARE COLLECTIONS のトラブルシューティング

本章では、Software Collection の構築時に発生する可能性のある典型的な問題のトラブルシューティングを行う方法を説明します。

## 5.1. ERROR: LINE *XX*: UNKNOWN TAG: %SCL_PACKAGE *SOFTWARE_COLLECTION_NAME*

Software Collection パッケージを構築すると、このエラーメッセージが出力されます。通常、これはscl-utils-build パッケージがないために生じます。scl-utils-build パッケージをインストールするには、次のコマンドを使用します。

```
# yum install scl-utils-build
```

詳細は、「Software Collections のサポートの有効化」 を参照してください。

## 5.2. SCL COMMAND DOES NOT EXIST

このエラーメッセージは、通常、scl-utils パッケージがないために生じます。scl-utils パッケージをインストールするには、次のコマンドを使用します。

```
# yum install scl-utils
```

詳細は、「Software Collections のサポートの有効化」 を参照してください。

## 5.3. UNABLE TO OPEN /ETC/SCL/PREFIXES/*SOFTWARE_COLLECTION_NAME*

このエラーメッセージは、呼び出す **scl** コマンドで誤った引数を使用すると発生する可能性があります。**scl** コマンドが正しいことと、引数を誤って入力していないことを確認します。

Software Collection が欠落しているのと同じエラーメッセージも考えられます。*software_collection_name* Software Collection がシステムに適切にインストールされていることを確認します。詳細は、「インストールされた Software Collections の一覧表示」 を参照してください。

## 5.4. SCL_SOURCE: COMMAND NOT FOUND

このエラーメッセージは、通常、scl-utils パッケージの古いバージョンがインストールされているために発生します。scl-utils パッケージを更新するには、以下のコマンドを実行します。

```
# yum update scl-utils
```

# 付録A 詳細情報の入手

Software Collection のパッケージ化、Red Hat Developers、Red Hat Software Collections、および Red Hat Developer Toolset 製品、および Red Hat Enterprise Linux の詳細は、以下の資料を参照してください。

## A.1. RED HAT 開発者

- Overview of Red Hat Software Collections on Red Hat Developers 『Red Hat Developers』 ポータルでは、さまざまな開発技術を使用してコードを開発するためのチュートリアルがいくつか紹介されています。これには、Node.js、Perl、PHP、Python、Ruby Software Collections が含まれます。

- Red Hat Developer Blog - 『Red Hat Developer Blog』 には、最新の情報、ベストプラクティス、意見、製品およびプログラムアナウンス、ならびに Red Hat の技術に基づくアプリケーションを設計および開発するユーザー向けのサンプルコードやその他のリソースへのポインターが含まれます。

## A.2. インストールされているドキュメント

- **scl**(1) - Software Collections を有効にし、Software Collection の環境でプログラムを実行する **scl** ツールの man ページです。

- **scl --help** - Software Collections を有効にし、Software Collection の環境でプログラムを実行する **scl** ツールの一般的な使用情報です。

- **rpmbuild**(8) - バイナリーパッケージとソースパッケージの両方を構築する **rpmbuild** ユーティリティーの man ページです。

## A.3. RED HAT ドキュメントへのアクセス

https://access.redhat.com/documentation/ にある **Red Hat 製品ドキュメント** は、一元的な情報源となります。リリースおよび技術ノートから、HTML、PDF、および EPUB 形式のインストール、ユーザー、リファレンスガイドなど、さまざまな種類のガイドを提供します。

以下は、本書に直接的または間接的に関連するドキュメントの簡潔な一覧です。

- Red Hat Software Collections 3.8 リリースノート - Red Hat Software Collections 3.8 の 『 リリースノート』 では、主な機能と、動的なプログラミング言語、データベースサーバー、さまざまな関連パッケージを提供する Red Hat 製品である Red Hat Software Collections に関するその他の情報が含まれています。

- Red Hat Developer Toolset 12.1 User Guide - Red Hat Developer Toolset 12.1 の 『 ユーザーガイド』 には、Red Hat Enterprise Linux プラットフォーム上の開発者向けの Red Hat 製品である Red Hat Developer Toolset に関する情報が記載されています。Software Collections を使用すると、Red Hat Developer Toolset は、**GCC** コンパイラー、**GDB** デバッガー、およびその他のバイナリーユーティリティーの現在のバージョンを提供します。

- Red Hat Software Collections 3.8 Container Images の使用 - 本書は、Red Hat Software Collections をベースとしたコンテナーイメージを使用する方法を説明します。利用可能なコンテナーイメージには、アプリケーション、デーモン、およびデータベースが含まれます。イメージは、Red Hat Enterprise Linux 7 Server および Red Hat Enterprise Linux Atomic Host で実行できます。

- Red Hat Enterprise Linux 7 開発者ガイド : Red Hat Enterprise Linux 7 の 『 開発者ガイド』 で

は、Red Hat Developer Toolset 機能の詳細と、Red Hat Software Collections の概要と、ライブラリーおよびランタイムサポートに関する情報、コンパイルおよび構築、デバッグ、およびプロファイリングについて説明します。

- Red Hat Enterprise Linux 7 システム管理者のガイド - Red Hat Enterprise Linux 7 の『システム管理者のガイド』では、Red Hat Enterprise Linux 7 のデプロイメント、設定、および管理に関する情報を説明しています。

- Red Hat Enterprise Linux 6 開発者ガイド : Red Hat Enterprise Linux 6 の『開発者ガイド』では、Red Hat Developer Toolset 機能の詳細と、Red Hat Software Collections の概要と、ライブラリーおよびランタイムサポートに関する情報、コンパイルおよび構築、デバッグ、およびプロファイリングについて説明します。

- Red Hat Enterprise Linux 6 デプロイメントガイド - Red Hat Enterprise Linux 6 の『デプロイメントガイド』では、Red Hat Enterprise Linux 6 のデプロイメント、設定、および管理に関する関連情報を提供します。

# 付録B 更新履歴

**改訂 4.2-0** **Fri Jun 09 2023** **Lenka Špačková**
「変換スペックファイルの例」 および 「タグおよびマクロ定義の変換」 を改善

**改訂 4.1-9** **Tue May 23 2023** **Lenka Špačková**
Red Hat Developer Toolset 12.1 のリリースで関連資料を更新

**改訂 4.1-8** **Mon Nov 15 2021** **Lenka Špačková**
パッケージガイドの Red Hat Software Collections 3.8 リリース

**改訂 4.1-7** **Mon Oct 11 2021** **Lenka Špačková**
パッケージガイドの Red Hat Software Collections 3.8 Beta リリース

**改訂 4.1-6** **Thu Jun 03 2021** **Lenka Špačková**
パッケージガイドの Red Hat Software Collections 3.7 リリース

**改訂 4.1-5** **Mon May 03 2021** **Lenka Špačková**
パッケージガイドの Red Hat Software Collections 3.7 Beta リリース

**改訂 4.1-4** **Tue Dec 01 2020** **Lenka Špačková**
パッケージガイドの Red Hat Software Collections 3.6 リリース

**改訂 4.1-3** **Tue Oct 29 2020** **Lenka Špačková**
パッケージガイドの Red Hat Software Collections 3.6 Beta リリース

**改訂 4.1-2** **Tue May 26 2020** **Lenka Špačková**
パッケージングガイドの Red Hat Software Collections 3.5 リリース

**改訂 4.1-1** **Tue Apr 21 2020** **Lenka Špačková**
パッケージングガイドの Red Hat Software Collections 3.5 Beta リリース

**改訂 4.1-0** **Tue Dec 10 2019** **Lenka Špačková**
パッケージングガイドの Red Hat Software Collections 3.4 リリース

**改訂 4.0-9** **Mon Oct 28 2019** **Lenka Špačková**
パッケージングガイドの Red Hat Software Collections 3.4 Beta リリース

**改訂 4.0-8** **Tue Jun 04 2019** **Petr Kovář**
パッケージングガイドの Red Hat Software Collections 3.3 リリース

**改訂 4.0-7** **Wed Apr 10 2019** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 3.3 Beta リリース

**改訂 4.0-6** **Thu Nov 01 2018** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 3.2 リリース

**改訂 4.0-5** **Wed Oct 17 2018** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 3.2 Beta リリース

**改訂 4.0-4** **Thu Apr 12 2018** **Petr Kovář**
パッケージングガイドの Red Hat Software Collections 3.1 リリース

**改訂 4.0-3** **Fri Mar 16 2018** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 3.1 Beta リリース

**改訂 4.0-2** **Tue Oct 17 2017** **Petr Kovář**

パッケージングガイドの Red Hat Software Collections 3.0 リリース

**改訂 4.0-1** **Thu Aug 31 2017** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 3.0 Beta リリース

**改訂 3.10-0** **Mon Jun 5 2017** **Petr Kovář**
BZ#1458821 を修正するために再公開。

**改訂 3.9-0** **Thu Apr 20 2017** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 2.4 リリース

**改訂 3.8-0** **Wed Apr 05 2017** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 2.4 Beta リリース

**改訂 3.7-0** **Wed Jan 25 2017** **Petr Kovář**
BZ#1263733 を修正するために再公開。

**改訂 3.6-0** **Wed Nov 02 2016** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 2.3 リリース

**改訂 3.5-0** **Wed Oct 12 2016** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 2.3 Beta リリース

**改訂 3.4-0** **Mon May 23 2016** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 2.2 リリース

**改訂 3.3-0** **Tue Apr 26 2016** **Petr Kovář**
パッケージングガイドの Red Hat Software Collections 2.2 Beta リリース

**改訂 3.2-0** **Wed Nov 04 2015** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 2.1 リリース

**改訂 3.1-0** **Tue Oct 06 2015** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 2.1 Beta リリース

**改訂 3.0-2** **Tue May 19 2015** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 2.0 リリース

**改訂 3.0-1** **Wed Apr 22 2015** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 2.0 Beta リリース

**改訂 2.2-4** **Fri Nov 21 2014** **Petr Kovář**
BZ#1150573、BZ#1022023、および BZ#1149650 を修正するために再公開。

**改訂 2.2-2** **Thu Oct 30 2014** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 1.2 リリース

**改訂 2.2-1** **Tue Oct 07 2014** **Petr Kovář**
パッケージガイドの Red Hat Software Collections 1.2 Beta リフレッシュリリース

**改訂 2.2-0** **Tue Sep 09 2014** **Petr Kovář**
Software Collections ガイドの名前がパッケージガイドに変更
パッケージガイドの Red Hat Software Collections 1.2 Beta リリース

**改訂 2.1-29** **Wed Jun 04 2014** **Petr Kovář**
Software Collections ガイドの Red Hat Software Collections 1.1 リリース

**改訂 2.1-21** **Thu Mar 20 2014** **Petr Kovář**
Software Collections ガイドの Red Hat Software Collections 1.1 Beta リリース

**改訂 2.1-18** **Tue Mar 11 2014** **Petr Kovář**
Software Collections ガイドの Red Hat Developer Toolset 2.1 リリース

**改訂 2.1-8** **Tue Feb 11 2014** **Petr Kovář**
Software Collections ガイドの Red Hat Developer Toolset 2.1 Beta リリース

**改訂 2.0-12** **Tue Sep 10 2013** **Petr Kovář**
Software Collections ガイドの Red Hat Developer Toolset 2.0 リリース

**改訂 2.0-8** **Tue Aug 06 2013** **Petr Kovář**
Software Collections ガイドの Red Hat Developer Toolset 2.0 Beta-2 リリース

**改訂 2.0-3** **Tue May 28 2013** **Petr Kovář**
Software Collections ガイドの Red Hat Developer Toolset 2.0 Beta-1 リリース

**改訂 1.0-2** **Tue Apr 23 2013** **Petr Kovář**
BZ#949000 を修正するために再公開。

**改訂 1.0-1** **Tue Jan 22 2013** **Petr Kovář**
Software Collections ガイドの Red Hat Developer Toolset 1.1 リリース

**改訂 1.0-2** **Thu Nov 08 2012** **Petr Kovář**
Software Collections ガイドの Red Hat Developer Toolset 1.1 Beta-2 リリース

**改訂 1.0-1** **Wed Oct 10 2012** **Petr Kovář**
Software Collections ガイドの Red Hat Developer Toolset 1.1 Beta-1 リリース

**改訂 1.0-0** **Tue Jun 26 2012** **Petr Kovář**
Software Collections ガイドの Red Hat Developer Toolset 1.0 リリース

**改訂 0.0-2** **Tue Apr 10 2012** **Petr Kovář**
Software Collections ガイドの Red Hat Developer Toolset 1.0 Alpha-2 リリース

**改訂 0.0-1** **Tue Mar 06 2012** **Petr Kovář**
Software Collections ガイドの Red Hat Developer Toolset 1.0 Alpha-1 リリース

## B.1. 承認

本書の著者は、以下の方々の貴重な貢献に感謝したいと思います。Jindřich Nový、Marcela Mašláňová、Bohuslav Kabrda、Honza Horák、Jan Zelený、Martin Čermák、Jitka Plesníková、Langdon White、Florian Nadge、Stephen Wadeley、Douglas Silas、Tomáš Čapek、Vít Ondruch、その他貢献いただいた各位。