Sun N1 Service Provisioning System 5.1 プランとコンポーネントの開発者ガイド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–4223–10
2005 年 9 月

Copyright 2005 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

本製品およびそれに関連する文書は著作権法により保護されており、その使用、複製、頒布および逆コンパイルを制限するライセンスのもとにおいて頒布されます。サン・マイクロシステムズ株式会社の書面による事前の許可なく、本製品および関連する文書のいかなる部分も、いかなる方法によっても複製することが禁じられます。

本製品の一部は、カリフォルニア大学からライセンスされている Berkeley BSD システムに基づいていることがあります。UNIX は、X/Open Company, Ltd. が独占的にライセンスしている米国ならびに他の国における登録商標です。フォント技術を含む第三者のソフトウェアは、著作権により保護されており、提供者からライセンスを受けているものです。

U.S. Government Rights Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

本製品に含まれる HG-MinchoL、HG-MinchoL-Sun、HG-PMinchoL-Sun、HG-GothicB、HG-GothicB-Sun、および HG-PGothicB-Sun は、株式会社リコーがリョービイマジクス株式会社からライセンス供与されたタイプフェースマスタをもとに作成されたものです。 HeiseiMin-W3H は、株式会社リコーが財団法人日本規格協会からライセンス供与されたタイプフェースマスタをもとに作成されたものです。フォントとして無断複製することは禁止されています。

Sun、Sun Microsystems、docs.sun.com、AnswerBook、AnswerBook2 は、米国およびその他の国における米国 Sun Microsystems, Inc. (以下、米国 Sun Microsystems 社とします) の商標、登録商標もしくは、サービスマークです。

サンのロゴマークおよび Solaris は、米国 Sun Microsystems 社の登録商標です。

すべての SPARC 商標は、米国 SPARC International, Inc. のライセンスを受けて使用している同社の米国およびその他の国における商標または登録商標です。SPARC 商標が付いた製品は、米国 Sun Microsystems 社が開発したアーキテクチャーに基づくものです。

OPENLOOK、OpenBoot、JLE は、サン・マイクロシステムズ株式会社の登録商標です。

Wnn は、京都大学、株式会社アステック、オムロン株式会社で共同開発されたソフトウェアです。

Wnn6 は、オムロン株式会社、オムロンソフトウェア株式会社で共同開発されたソフトウェアです。©Copyright OMRON Co., Ltd. 1995-2000. All Rights Reserved. ©Copyright OMRON SOFTWARE Co.,Ltd. 1995-2002 All Rights Reserved.

「ATOK」は、株式会社ジャストシステムの登録商標です。

「ATOK Server/ATOK12」は、株式会社ジャストシステムの著作物であり、「ATOK Server/ATOK12」にかかる著作権その他の権利は、株式会社ジャストシステムおよび各権利者に帰属します。

「ATOK Server/ATOK12」に含まれる郵便番号辞書 (7 桁/5 桁) は日本郵政公社が公開したデータを元に制作された物です (一部データの加工を行なっています)。

「ATOK Server/ATOK12」に含まれるフェイスマーク辞書は、株式会社ビレッジセンターの許諾のもと、同社が発行する『インターネット・パソコン通信フェイスマークガイド』に添付のものを使用しています。

Unicode は、Unicode, Inc. の商標です。

本書で参照されている製品やサービスに関しては、該当する会社または組織に直接お問い合わせください。

OPEN LOOK および Sun Graphical User Interface は、米国 Sun Microsystems 社が自社のユーザーおよびライセンス実施権者向けに開発しました。米国 Sun Microsystems 社は、コンピュータ産業用のビジュアルまたはグラフィカル・ユーザーインタフェースの概念の研究開発における米国 Xerox 社の先駆者としての成果を認めるものです。 米国 Sun Microsystems 社は米国 Xerox 社から Xerox Graphical User Interface の非独占的ライセンスを取得しており、このライセンスは、OPEN LOOK のグラフィカル・ユーザーインタフェースを実装するか、またはその他の方法で米国 Sun Microsystems 社との書面によるライセンス契約を遵守する、米国 Sun Microsystems 社のライセンス実施権者にも適用されます。

本書は、「現状のまま」をベースとして提供され、商品性、特定目的への適合性または第三者の権利の非侵害の黙示の保証を含みそれに限定されない、明示的であるか黙示的であるかを問わない、なんらの保証も行われないものとします。

本製品が、外国為替および外国貿易管理法 (外為法) に定められる戦略物資等 (貨物または役務) に該当する場合、本製品を輸出または日本国外へ持ち出す際には、サン・マイクロシステムズ株式会社の事前の書面による承諾を得ることのほか、外為法および関連法規に基づく輸出手続き、また場合によっては、米国商務省または米国所轄官庁の許可を得ることが必要です。

原典: Sun N1 Service Provisioning System 5.1 Plan and Component Developer's Guide

Part No: 819-1659–10

Revision A





051021@13215

目次

はじめに　**7**

1　プランおよびコンポーネント開発の概念　**11**
コンポーネントの概念　11
　コンポーネントのモデル化　13
　コンポーネントの特性　14
　コンポーネント手続き　15
　コンポーネントの継承　16
　コンポーネント変数　16
　変数設定値　17
　変数の優先指定　17
　ステップ　17
コンポーネントタイプの概念　18
システムサービスの概念　19
プランの概念　19
　プランの種類　19
　ステップの概要　20
リソース記述子ファイルの概念　22
セッション変数の概念　23
　セキュリティー保護されたセッション変数　24

2　コンポーネント　**25**
コンポーネントの管理　25
　▼ コンポーネントを作成する　27
　▼ コンポーネントを削除する　28
コンポーネント CLI コマンドの概要　29

コマンド行インタフェースによるコンポーネントのチェックイン　31

3　組み込みコンポーネントタイプ　**33**
コンポーネントタイプ: `system#file`　33
ブラウズ　34
拡張制御手続き　34
コンポーネントタイプ: `system#directory`　35
ブラウズ　36
拡張制御手続き　37
コンポーネントタイプ: `system#symbolic link`　37
ブラウズ　38
エクスポート/内部ファイル形式　38
コンポーネントタイプ: `system#container`　39
ブラウズ　39
モデルとインストールの差分　39
拡張制御手続き　39
コンポーネントタイプ: `untyped`　40

4　プラン　**41**
プランの管理　41
▼ プランを編集する　42
プランの作成　43
▼ プランを生成する　43
▼ カスタムプランを作成する　44
プランの実行　46
▼ プランを実行する　46
▼ 直接実行コンポーネント手続きを使用してコンポーネントを配備する　49
プラン CLI コマンドの概要　51

5　セッション変数　**53**
セッション変数の管理　53
セッション変数 CLI コマンドの概要　54

6　構成の生成　**55**
構成の生成の概要　55
コンポーネントへの置換変数の追加　56
置換変数の値　56

生成コンテキスト　57
入力ソース　57
置換変数の検証　57
置換に使用できる変数の種類　58
単純置換変数参照　58
外部コンポーネント置換変数参照　61
セッション置換変数の参照　67
ターゲット置換変数の参照　68
置換変数の使用法　71

A　変数置換の文法　**77**

用語集　**83**

索引　**91**

はじめに

『Sun N1 Service Provisioning System 5.1 プランとコンポーネントの開発者ガイド』では、プランおよびコンポーネントを作成する開発者に必要な情報を提供します。プランおよびコンポーネントの開発に関連する概念についても説明します。

対象読者

このマニュアルは、Sun N1™ Service Provisioning System を使用してデータセンターにアプリケーションを導入し、管理する開発者が対象です。

このマニュアルの構成

第 1 章では、コンポーネント、プラン、コンポーネントタイプ、システムサービス、セッション変数に関連する概念について説明します。

第 2 章では、コンポーネントに関連する作業の手順について説明します。

第 3 章では、組み込みコンポーネントタイプについて説明します。

第 4 章では、プランの作成、管理、実行方法について説明します。

第 5 章では、セッション変数の管理方法について説明します。

第 6 章では、構成の生成および置換変数について説明します。

付録 A では、変数置換で使用する文法について説明します。

関連マニュアル

Sun N1 Service Provisioning Systemのマニュアルには、以下の関連マニュアルが含まれます。

- 『Sun N1 Service Provisioning System 5.1 リリースノート』
- 『Sun N1 Service Provisioning System 5.1 インストールガイド』
- 『Sun N1 Service Provisioning System 5.1 システム管理者ガイド』
- 『Sun N1 Service Provisioning System 5.1 Operations and Provisioning Guide』
- 『Sun N1 Service Provisioning System 5.1 コマンド行インタフェース(CLI) リファレンスマニュアル』
- 『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』
- 『Sun N1 Service Provisioning System 5.1 Plug-In Developer's Guide』

マニュアル、サポート、およびトレーニング

Sun のサービス	**URL**	内容
マニュアル	`http://jp.sun.com/documentation/`	PDF 文書および HTML 文書をダウンロードできます。
サポートおよびトレーニング	`http://jp.sun.com/supportraining/`	技術サポート、パッチのダウンロード、および Sun のトレーニングコース情報を提供します。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 **P–1** 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	system% **su** password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャー・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% **grep '^#define ** **XV_VERSION_STRING'**

コード例は次のように表示されます。

- C シェル

  machine_name% **command y|n** [*filename*]

- C シェルのスーパーユーザー

  machine_name# **command y|n** [*filename*]

- Bourne シェルおよび Korn シェル

  $ **command y|n** [*filename*]

- Bourne シェルおよび Korn シェルのスーパーユーザー

  # **command y|n** [*filename*]

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

第 1 章

プランおよびコンポーネント開発の概念

この章では、Sun N1 Service Provisioning System ソフトウェア (これ以降、プロビジョニングシステム) に関連するプランとコンポーネントの概念について説明します。

この章の内容は次のとおりです。

- 11 ページの「コンポーネントの概念」
- 18 ページの「コンポーネントタイプの概念」
- 19 ページの「システムサービスの概念」
- 19 ページの「プランの概念」
- 22 ページの「リソース記述子ファイルの概念」
- 23 ページの「セッション変数の概念」

コンポーネントの概念

*component*は、アプリケーションを定義する *resources*の論理的なグループです。アプリケーションを構成するリソースの扱い方を規定した一連の命令も含まれます。コンポーネントは、以下のように分けることができます。

- ファイルとディレクトリ
- 自律型アーカイブ (J2EE™ Enterprise Archives (EAR)、COM コンポーネントなど)
- 完結したアプリケーション (BEA WebLogic Server など)
- オペレーティングシステムレベルの更新 (パッチ、サービスパックなど)
- ほかのコンポーネントを示すポインタ

プロビジョニングシステムを使用して、データセンター内のアプリケーションを管理できます。この製品でアプリケーションを管理するには、先にアプリケーションをコンポーネントとして*model*しておく必要があります。プロビジョニングシステムを使用すると、次の作業を行えます。

- アプリケーションのインストール、構成、および分析に関する情報とともに、各アプリケーションごとに慎重に定義したソフトウェアリソースが含まれる、アプリケーションモデルを作成できます。
- バージョン管理が採用されているコンポーネントリポジトリにコンポーネントを格納できます。バージョン管理により、任意のバージョンのコンポーネントを取得できます。
- プランでコンポーネントを利用できるようにします。プランでは、手順を追ってデータセンターオペレーションを実行し、各コンポーネントに組み込まれているナレッジを活用します。
- コンポーネント間の比較およびコンポーネントとソフトウェア導入先との比較

プロビジョニングシステムがサポートするコンポーネントの基本タイプは、次の 2 種類です。

- 単純コンポーネント。*simple component*には、単一リソースが含まれ、ほかのコンポーネントへの参照は含まれません。
- 複合コンポーネント。*composite component*には、ほかのコンポーネント (単純と複合の両方) への参照だけが含まれ、リソースは含まれません。

複合コンポーネントには、コンポーネントの参照が含まれます。参照先のコンポーネントを *contained components* (子コンポーネント) といいます。参照する側のコンポーネントを *container component* (親コンポーネント) といいます。

複合コンポーネントは、各包含コンポーネントについて、 *top-level component*としてインストールするのか、それとも *nested component*としてインストールするのかを宣言します。上位レベルとしてインストールされた包含コンポーネントは、プランによって直接インストールされた場合と同様、どのコンポーネントでも使用されます。ただし、包含コンポーネントが入れ子としてインストールされた場合、そのサービスを利用できるのはコンテナコンポーネントに限られます。入れ子の包含コンポーネントは、コンテナコンポーネントが必要とする、よりきめの細かい機能単位を定義します。ただし、ほかのコンポーネントにとっては有用ではありません。一方、上位レベルの包含コンポーネントでは、コンテナコンポーネントが使用するサービスを定義しますが、定義されたサービスはほかのコンポーネントも使用できます。

注 – 複合コンポーネントにはコンポーネントそのものではなく、ほかのコンポーネントの参照が含まれます。参照コンポーネントは、別の既存コンポーネントであり、コンテナコンポーネントとは別に更新され管理されます。1 つのコンポーネントを、任意の数の複合コンポーネントが参照できます。ほかの複合コンポーネントから参照されるかどうかで、コンポーネント名が変わることはありません。名前の衝突はパスを使用して解決されます。

プロビジョニングシステムは、コンポーネントに関連付けられる物理リソースを管理します。これには事前に定義された一連のコンポーネントも含まれます。このようなコンポーネントは直接使用することも、ほかのコンポーネントを作成するためのサンプルとして使用することもできます。

コンポーネントには手続きも組み込まれます。 *procedure*は、コンポーネントに格納され、コンポーネントの配置を制御するプログラムです。コンポーネント手続きは通常、インストール、アンインストール、およびキャプチャの *snapshots*として定義します。その他の手続きは制御ブロックで定義できます。*control*とは、コンポーネントに組み込まれた一連の命令であり、配備されたアプリケーションを管理する目的で使用できます。たとえばアプリケーションの起動や停止などを制御できます。コンポーネントには、ほかのコンポーネントへの依存を検査したり、特定のプロセスが動作しているかどうかを検証したりする命令を組み込むことができます。

コンポーネントでは *substitution variables*も宣言できます。置換変数は、コンポーネント内で使用することも、コンポーネントに含まれる任意のリソースに使用することもできます。置換変数は、プランの実行時に置換する値のプレースホルダとして使用できます。

コンポーネントのモデル化

プロビジョニングシステムでは、非常に柔軟にコンポーネントをモデル化できます。モデル化するアプリケーションによって、次のどの方式を使用するかが決まります。

- 完全自動のモデル化**:** *gold server*から、またはソースコード制御システムから、コンポーネントにチェックインできます。チェックインが完了すると、プロビジョニングシステムが*component*を自動的に生成するので、生成されたコンポーネントからインストール、構成、*comparison*手続きを実行できます。

  この方式は、プラグインがすでに定義されてインポートされているアプリケーションをモデル化する場合に使用します。組み込みコンポーネントタイプまたはインポートされたコンポーネントタイプを使用すると、アプリケーションコンポーネントを構成する一般的なリソースを自動的にモデル化できます。コンポーネントタイプのテンプレートには、インストールなど、基本的な動作に関する組み込み手続きが含まれています。したがって、プランを作成しなくても、一般的なコンポーネントタイプの基本動作を実行できます。

  組み込みコンポーネントタイプについては、第 3 章を参照してください。プラグインによって取得したコンポーネントタイプのリストは、`http://docs.sun.com/db/coll/1329.1` にアクセスして、『Sun N1 Service Provisioning System Plug-In Collection — Current Version』を参照してください。

- **XML** オーサリングによる組み込みコンポーネントタイプの拡張**:** ブラウザインタフェースの「Advanced Edit」ページで XML を直接編集することによって、自動生成されたコンポーネントをカスタマイズできます。XML が指定されているファイルをシステムにダウンロードし、Turbo XML などの XML スキーマ検証エディタで編集することによって、コンポーネントをカスタマイズする方法もあります。

  XML の編集時には、次の作業が可能です。

  - 変数の追加によるコンポーネントのカスタマイズ
  - 拡張制御手続きの呼び出しの追加 (IIS または Microsoft Windows サービスの起動/停止など)

- **XML** でのコンポーネントモデルのオーサリング**:** XML エディタを使用して、『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の第 2 章「コンポーネントと単純プランにより使用される共有スキーマ」、『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の第 3 章「コンポーネントのスキーマ」、および『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の第 4 章「プランのスキーマ」に記載されているコンポーネントスキーマを参照することによって、独自のコンポーネントを作成できます。

  コンポーネントを使用する前に、そのコンポーネントの XML ファイルとリソースをリポジトリにチェックインしておく必要があります。
- ブラウザインタフェースを使用したコンポーネントモデルのオーサリング。ブラウザインタフェースを使用してコンポーネントを作成できます。保存すると、コンポーネントが自動的にビルドされます。その後、次のいずれかによってコンポーネントをインストールできます。
  - 単一*host*または*host set*にコンポーネントをインストールする、インストール手続きの実行
  - 1つ以上のホストまたはホストセットにコンポーネントをインストールするプランの作成

    プランについての詳細は、第 4 章を参照してください。

  コンポーネントを使用するには、そのコンポーネントをリポジトリにチェックインしておく必要があります。

コンポーネントをチェックインすると、コンポーネントがビルドされ、特定のバージョンのリソースが割り当てられます。ビルドによりコンポーネントにもバージョン番号が割り当てられるため、そのコンポーネントの適切なバージョンが、コンポーネントリソースの特定のバージョンと常に関連付けられるようになります。

コンポーネントの特性

コンポーネントはさまざまな特性に基づいて識別されます。次に、代表的なコンポーネント特性について説明します。

- パス**:** コンポーネントを格納するフォルダ内の位置。フォルダを使用してコンポーネントを階層方式で編成します。
- コンポーネント名**:** コンポーネントの名前です。
- コンポーネントタイプ名**:** コンポーネントに関連付けられるコンポーネントタイプの名前です。18 ページの「コンポーネントタイプの概念」を参照してください。

  プロビジョニングシステムには、ファイル、ディレクトリなどの汎用モデルをサポートするコンポーネントタイプがいくつか組み込まれています。Microsoft Windows および WebLogic など、アプリケーションドメインに固有のプラグインをインポートすることによって、ほかのコンポーネントタイプをシステムに追加できます。
- バージョン番号**:** コンポーネントのバージョン番号です。コンポーネントを変更するたびに、バージョン番号が増えます。

チェックイン時に、バージョン番号の増分方式を選択します。メジャー番号 (小数点の左側) を増やすことも、マイナー番号 (小数点の右側) を増やすこともできます。

- プラットフォーム: このコンポーネントをインストールできるオペレーティングシステムです。
- チェックイン日付: コンポーネントがチェックインされた日時です。
- チェックインユーザー: このコンポーネントをチェックインした人物のユーザー ID です。この属性は、プロビジョニングシステムのプロセスを監査する場合に便利です。
- ラベル: コンポーネントの分類に任意で使用する文字列です。
- カテゴリ: コンポーネントリストをフィルタリングするために、任意で使用するオブジェクトです。カテゴリオブジェクトを作成すると、以後はそのオブジェクトを使用してコンポーネントを分類できます。
- 説明: コンポーネントを説明する文字列。使用は任意です。この属性を使用して、コンポーネントの説明情報を提供します。
- ソース: このコンポーネントに組み込まれているリソースです。ソースは、単一ファイルリソースでも、ほかのコンポーネントのリストでもかまいません。
- コンポーネント変数: コンポーネントリソースの配備に必要な変数と値 (名前と値のペア) のリストです。16 ページの「コンポーネント変数」を参照してください。
- 手続き: リソースおよび変数で何を行うかを指定した、一組の命令です。
- 非表示: コンポーネントリストでコンポーネントを表示するかどうかを指定するための特性です。デフォルトでは、コンポーネントは表示されます。

  バージョンの古いコンポーネントや管理対象ではないコンポーネントを隠すと、コンポーネントリストを短縮できます。短縮リストには、隠されていない、つまり現在管理しているコンポーネントが含まれます。隠しコンポーネントは、表示を要求しないかぎり、リストには含まれません。

コンポーネント手続き

component procedure (または *procedure*) は、コンポーネントに組み込まれ、コンポーネントを管理するプログラムです。コンポーネント手続きはコンポーネントのインストール、アンインストール、または制御を行います。手続きはコンポーネントのビルド時に作成されます。

手続きの実行方法は次のとおりです。

- コンポーネントの「Details」ページ (プランを生成して実行する場所) から直接実行する
- 手続きを呼び出すプランを作成し、そのプランを実行する
- ほかのコンポーネントおよびプランから手続きを呼び出す

コンポーネントには複数の手続きを指定できます。たとえば、制御手続きをコンポーネントに組み込むことで、そのコンポーネントがモデル化するアプリケーションの起動と停止を行います。

コンポーネントタイプの拡張により作成されたコンポーネントは、そのコンポーネントタイプによって定義された手続きを継承します。

複合コンポーネントを作成した場合は、デフォルトのインストールおよびアンインストール手続きを継承します。通常、コンポーネントは関連付けられるコンポーネントタイプから手続きを継承します。

コンポーネントの継承

*Component inheritance*とは、コンポーネントが別のコンポーネントから属性や動作を取得する手段です。作成したコンポーネントは、関連付けられているコンポーネントタイプの*variables*、*snapshots*、および*procedures* を継承します。

継承を用いることによって、強力かつ柔軟性の高いコンポーネントモデルになります。たとえば、IIS Aplication コンポーネントタイプに基づくコンポーネントが 1000 個あるとします。これらのコンポーネントに機能を追加するとします。IIS Application コンポーネントタイプに機能を追加することで、そのコンポーネントタイプに属する 1000 個のコンポーネントが新しい機能を継承することになります。

コンポーネント変数

コンポーネントでは *variables*を使用できます。変数は値を格納するユーザー定義のコンテナであり、 *deployment*時に使用されます。

*component variable*は、コンポーネントの外部にあるオブジェクトからコンポーネントの一部にアクセスして設定できるようにするために使用します。たとえば、コンポーネントにホスト単位で変更される `installPath` という変数を設定できます。この変数では、各コンポーネントのインストール先を定義します。

コンポーネント変数の値を別のコンポーネント変数への参照にすることができます。これには、コンポーネントのコンテナによって定義された変数が含まれます。コンテナコンポーネントに入れ子のコンポーネントを追加すると、ブラウザインタフェースは、参照変数がコンテナで定義されているかどうかを検証します。その変数がコンテナコンポーネントで定義されていない場合、ブラウザインタフェースは自動的に参照変数をコンテナの変数リストに追加します。

たとえば、単純コンポーネントは通常、`installPath` 変数がコンテナコンポーネントの `installPath` 変数の値を持つように定義します。この場合、参照変数の構文は `:[container:installPath]` です。変数置換についての詳細は、第 6 章を参照してください。

コンポーネント変数は、コンポーネントの配備時に評価されて値が割り当てられます。コンポーネント変数を使用して、配備の実装に必要な情報 (ホスト名、IP アドレス、パスなど) を指定します。

変数設定値

*Variable settings*は、変数の値の集まりであり、1 つ以上のコンポーネント変数について、デフォルト値を変更する目的で使用できます。使用する変数設定値に基づいて、コンポーネント変数にさまざまな値を指定できます。どの変数設定値を使用するかは、プランの実行時に指定します。

たとえば、コンポーネントを運用環境および開発環境など、異なる環境に配備します。定義したコンポーネント変数が、環境間の相違を認識するように設定されている場合、変数設定値を使用することによって、各環境に合わせてコンポーネントを設定できます。たとえば、あるセットを使用して運用環境を設定し、別のセットで開発環境用にコンポーネントを設定することが可能です。

変数の優先指定

変数の優先指定によって、複合コンポーネントの変数のデフォルト値を上書きできます。単純コンポーネントでこの機能を使用することはできません。

コンポーネントにほかのコンポーネント (*child components*) が含まれている場合、変数設定値が有効なのは上位レベルの *parent component*に限られます。すべての子コンポーネントは、デフォルトの変数値を使用します。子コンポーネントは、次の 2 通りの方法で、親コンポーネントから変数値を取得できます。

- コンテナ **(**親**)** コンポーネントは、包含 **(**子**)** コンポーネントに変数値を「プッシュ」します。子コンポーネントのデフォルト値を上書きするには、子コンポーネントを収容するコンポーネントを作成するときに、変数の優先指定を設定します。参照コンポーネントごとに、使用できる変数の優先指定値のセットを設定します。
- 包含 **(**子**)** コンポーネントは、コンテナ **(**親**)** コンポーネントから変数値を「プル」します。包含コンポーネントの変数値は 1 つ以上がコンテナコンポーネントの変数値に基づいて定義されます。包含コンポーネントは、変数置換構文 `:[container:`*varname* `]` を変数のデフォルト値に使用します。

ステップ

ステップは、プランとコンポーネントの両方に組み込むことのできる単純な命令です。

ステップについては、20 ページの「ステップの概要」を参照してください。

コンポーネントタイプの概念

*component type*とは、動作をカプセル化してほかのコンポーネントで再利用できるように設計されたコンポーネントです。コンポーネントタイプは通常、特定のアプリケーションタイプに固有の動作を意味します。たとえば、Enterprise JavaBeans™ (EJB™) コンポーネントタイプには、アプリケーションサーバーとの間で相互に EJB アーカイブを配備したり配備を解除したりするために使用できる、汎用手続きが組み込まれます。コンポーネントをコンポーネントタイプから *extends* すると、そのコンポーネントタイプで定義されている動作が自動的に継承されます。

コンポーネントは通常、コンポーネントタイプと関連付けられます。ブラウザインタフェースから `untyped` を選択した場合は、コンポーネントがほかのコンポーネントタイプから拡張されることはありません。プロビジョニングシステムには組み込みコンポーネントタイプがいくつかあります。第 3 章を参照してください。

単純コンポーネントが参照するファイル、ディレクトリ、およびその他のツリー構造は、コンポーネントに含まれる別々の単位として管理されます。

たとえば、プロビジョニングシステムが IIS アプリケーションを参照リソースとして管理する場合、IIS アプリケーションには下記が含まれます。

- ディレクトリとその内容
- IIS Web サイトの設定値
- COM+ アプリケーション
- Microsoft Windows のレジストリ設定値

ファイル、ディレクトリなど、コンポーネントが参照するリソースの一部は、ゴールドサーバーまたは別のデータソースからコピーできます。IIS Web サイトの設定値、Microsoft Windows のレジストリ設定値など、その他については、独立した管理可能な要素として扱えるように、特殊な方法でデータソースから抽出する必要があります。

組み込みコンポーネントタイプに関して、プロビジョニングシステムは Java™ 2 プラットフォーム、Enterprise Edition (J2EE プラットフォーム)、および Microsoft Windows アプリケーションに使用する、一般的なソースアイテムを認識できます。これらのコンポーネントタイプでは、コンポーネントリソースとして使用するデータを正確に抽出し、リポジトリにコンポーネントリソースを格納して、指定された場所にリソースを正しくインストールすることができます。

システムサービスの概念

*System services*は、ホストを準備するときに、該当するすべてのホストに自動的に配備されるコンポーネントです。システムサービスは、コンポーネントのインストールと管理で、ほかのコンポーネントによって使用されるユーティリティ制御とリソースを定義します。

システムサービスによって、プランの実行時に利用できるサービスセットが充実します。

プランの概念

*plan*は *execution plan*ともいい、特定のホスト上で 1 つ以上のコンポーネントを管理するために使用する命令のシーケンスです。たとえば、実行プランで 3 つのコンポーネントをインストールして、別のコンポーネントの「起動」制御を開始することができます。

プランは、ほかのプランのシーケンスを含むことができます。これにより、共通の命令シーケンスをプランとして作成し、複数のプラン間で共有できます。たとえば、3 つのコンポーネントをインストールし、別のコンポーネントに対して起動を開始するように、プランでプロビジョニングシステムに指示します。

プロビジョニングシステムは、 *XML schema*で表すオブジェクトのメモリー内表現を提供します。この表現は、これらのオブジェクトの妥当性検査、持続性、バージョン管理のプロセスも定義します。

プロビジョニングシステムがプランを実行するときに、コンポーネントで宣言された置換変数が実際の値に置き換えられます。プロビジョニングシステムは通知機能もサポートするので、プランの実行に関連したイベントに対して電子メールを送信できます。

プランの種類

プロビジョニングシステムがサポートするプランは、次の 2 種類です。

- 単純プラン。*simple plan*には、単純ステップの集まりが含まれます。このプランは、ほかのプランを呼び出すことはできません。
- 複合プラン。*composite plan*には、ほかのプラン (サブプラン) だけが含まれます。

XML スキーマで 2 種類のプランを区別します。したがって、ほかのサブプランの呼び出しを含んでいる最上位のプランを使用することも、単純なステップからなり、サブプランの呼び出しは含まれていない単純プランを使用することもできます。

単純プランのステップの実行が同一ターゲットホストセット上に限定されるのに対して、複合プランのステップは複数の異なるターゲットホストセット上で実行できるので、この区別は重要です。複合プランでは、含まれる単純プランごとに、ターゲットホストセットを 1 つずつ使用できます。

ステップの概要

*Steps*は、プランとコンポーネントの両方に組み込むことのできる単純な命令です。プロビジョニングシステムがサポートするステップの種類は、次のとおりです。

- コンポーネント内部からのみ呼び出すことのできるステップ: このタイプのステップを呼び出すことができるのは、インストールブロックまたはアンインストールブロックからだけです。『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の「コンポーネントのインストール専用のステップ」と『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の「コンポーネントのアンインストール専用のステップ」を参照してください。
- プランからのみ呼び出すことのできるステップ: このタイプのステップを呼び出すことができるのは、複合プランまたは単純プランからだけです。『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の「複合プラン専用のステップ」と『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の「単純プラン専用のステップ」を参照してください。
- プランまたはコンポーネントから呼び出すことができるステップ: 『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の第 2 章「コンポーネントと単純プランにより使用される共有スキーマ」を参照してください。

コンポーネントの参照

各種ステップは、次の状況でコンポーネントを参照できます。

- まだインストールされていないコンポーネントは `<install>` ステップを使用できます。

  まだインストールされていないコンポーネントを参照するステップは、コンポーネント名とオプションのバージョンを指定するだけです。
- すでにインストール済みのコンポーネントは、次のステップを使用できます。
  - `<uninstall>`
  - `<call>`
  - `<checkDependency>`
  - `<createDependency>`
  - `<addSnapshot>`

  インストール済みコンポーネントを参照するステップは、*installPath* 属性を指定する必要があります。同じサーバーに同じコンポーネントを繰り返しインストールできるからです。

例 **1–1** コンポーネント参照の使い方

「Apache」コンポーネントが次の属性値とともに、ホストにインストールされているものとします。

コンポーネントインスタンス	*installPath*	*version*	*installDate*
A	`/opt`	1.3	6/1/01 6:00 p.m.
B	`/usr/local`	1.4	6/1/01 5:00 p.m.
C	`/opt`	1.2	6/2/01 5:00 p.m.
D	`/usr/local/bin`	1.4	6/3/01 5:00 p.m.
E	`/export`	1.1	6/4/01 5:00 p.m.

installPath 属性と *version* 属性の組み合わせに応じて値が指定されたときに、どのインストール済みコンポーネントが参照されるかを以下に示します。

installPath	*version*	*versionOp*	結果	説明
なし	なし	なし	E	*version* および *installPath* の値に関係なく、ターゲットホストに最後にインストールされたコンポーネントが使用されます。
`/opt`	なし	なし	C	*version* の値に関係なく、指定のインストールパスに最後にインストールされたコンポーネントが使用されます。
`/usr/bin`	なし	なし	エラー	指定のパスにコンポーネントはインストールされていません。
なし	1.4	`=`	D	*installPath* の値に関係なく、指定のバージョンが含まれているコンポーネントのうち、最後にインストールされたものが選択されます。
なし	1.5	任意	エラー	指定されたバージョンのコンポーネントはインストールされていません。
`/usr/local`	1.4	`=, >=`	B	指定の *installPath* 属性値および *version* 属性値と一致するコンポーネントが選択されます。
`/usr/local`	1.2	`=`	エラー	指定のインストール パスにこのバージョンはインストールされていません。
`/usr/local`	1.2	`>, >=`	B	*installPath*、*version*、および *versionOp* の値と一致したもの。

installPath	*version*	*versionOp*	結果	説明
/opt	3	任意	エラー	2 つ以上のコンポーネントが同じパスにインストールされ、なおかつ名前が同じ場合は、事実上、最後にインストールされたコンポーネントがインストール済みコンポーネントを上書きします。それまでにインストールされていたコンポーネントは、直接指定してもアクセスできません。

リソース記述子ファイルの概念

*resource descriptor file*は、単純コンポーネントのリソースを構成するファイルとディレクトリに使用する、所有者、グループ、およびアクセス許可の設定を指定します。このリソース記述子は XML ファイルです。このファイルの XML スキーマの詳細については、『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の第 5 章「リソース記述子スキーマ」を参照してください。リソース記述子ファイルを使用することで、コンポーネントのチェックイン時に決定されるアクセス許可を上書きできます。

リソース記述子ファイルを使用しない場合、リソースはチェックイン時の所有者、グループ、およびアクセス許可の設定を使用します。これは、UNIX システムでチェックインを実行する場合のデフォルトです。Windows システムでコンポーネントをチェックインする場合、デフォルト設定は、リソース記述子ファイルの各設定に `:NONE:` 値を使用したかのようになります。

リソース記述子ファイルを使用する場合、リソースは、リソース記述子により指定される所有者、グループ、およびアクセス許可の設定を使用します。リソースに対して `<entry>` 要素が指定されている場合、設定はそのエントリから取得されます。そのエントリが設定の一部を指定していない場合、存在しない設定値は `<defaultEntry>` から取得されます (存在する場合)。これらの設定値が `<defaultEntry>` で指定されていない場合、リソースはチェックイン時の設定値を使用します。

リソースに対して `<entry>` 要素が指定されていない場合、リソースは `<defaultEntry>` で指定されてる設定を使用します (存在する場合)。`<defaultEntry>` が存在しない場合、リソースはチェックイン時の設定を使用します。

コンポーネントが配備されるファイルシステムのデフォルト設定を使用するようプロビジョニングシステムに通知するには、`:NONE:` 値を使用します。リソースの `<defaultEntry>` ブロックまたは `<entry>` で指定されている設定には `:NONE:` 値を指定できます。

リソース記述子ファイルが使用されるのは、UNIX システムにコンポーネントを配備する場合のみです。コンポーネントを Windows システムに配備する場合、リソース記述子ファイルは無視されます。そのため、Windows システムにだけコンポーネントを適用する場合には、リソース記述子ファイルを作成しないでください。

リソース記述子ファイルは、system#file および system#directory コンポーネントタイプを拡張する単純コンポーネントで使用できます。またリソース記述子ファイルは、Linux プラグインの一部である com.sun.linux#rpm コンポーネントタイプを拡張するコンポーネントで使用することもできます。

コンポーネントをチェックインすると同時に、リソース記述ファイルもチェックインします。最後のチェックインにリソース記述子ファイルを使用したコンポーネントに対して checkin-current を試みる場合、プロビジョニングシステムは、元のチェックイン場所でリソース記述子が見つかると想定します。そのため、そのファイルを別の場所に移動し、コンポーネントに対して checkin-current を試みると、チェックイン処理は失敗します。

チェックインされた単純コンポーネントのリソース記述子をダウンロードすると、コンポーネントのリソースの一部であるあらゆるファイルの設定を確認できます。

このダウンロード機能を使用すると、ファイルを更新し、コンポーネントの更新バージョンをチェックインできます。まず、リソース記述子ファイルをダウンロードしてから、関連付けられたコンポーネントのリソースをチェックアウトします。続いて、リソース記述子ファイルを変更し、それを使用してコンポーネントの新しいバージョンをチェックインします。

ダウンロードするリソース記述子は、コンポーネントのチェックインに使用したリソース記述子とは異なる場合があります。違いが生じるのは、コンポーネントのチェックインに使用する記述子には、リソース内のあらゆるファイルに関する情報を持つ必要がなく、またあらゆるエントリに関する完全な情報を持つ必要がないためです。ダウンロードしたリソース記述子ファイルには <defaultEntry> ブロックが出現しないことに注意してください。その代わりに、どのファイルもそのファイルの <entry> 内で記述されています。

セッション変数の概念

*session*が開始されるのは、ブラウザインタフェースにログインしたとき、または CLI を使用したときです。セッションは、ユーザーがログアウトするまで、または非アクティブな状態によりセッションが終了するまで続きます。論理上、セッションは、特定のユーザーの認証済みの資格情報を表します。セッションを使用すると、再認証されなくても、一連の関連要求を通してユーザーが識別されます。

各セッションは、ログイン時にデータベースから初期化される、一式の *session variables* を持つことができます。セッション変数を使用して、ユーザー認証やその他の資格など、セッション関連の情報を維持できます。現在のセッションのセッション変数は、データベースに保存されているセッション変数に影響を及ぼすことなく変更できます。セッション変数とその値を変更した場合、それらをデータベースに保存できます。保存する場合は、すべての変数とその値が保存され、以降のセッションで使用できるようになります。プランや比較の実行時に、セッション変数を参照できます。

複数のセッションを開始していて、セッション変数を同時に保存しようとした場合、最初のセッションの変数のみが保存されます。最初のセッションのセッション変数が保存されたあと、ほかのセッションのセッション変数の変更を保存しようとしても保存できません。このような場合に変更を保存する方法は、次のとおりです。

1. すべてのセッションからログアウトします。
2. もう一度ログインしてセッションを開始します。

   正常に完了した最後の保存に基づいて、データベースからセッション変数と値の新しいセットを取得します。
3. セッション変数を適切に変更します。
4. 変更したセッション変数を保存します。

セッション変数は大域名前空間に存在するので、宣言したセッション変数名をすべてのユーザーセッションで使用できます。たとえば、セッション変数名として `passwd` を定義したとします。`passwd` セッション変数を要求するプランは、いずれも同じ変数を参照します。セッション変数名の有効範囲は現在のプラン、コンポーネント、ブロック、またはホストに限定されません。したがって、セッション変数名が一意になるように配慮する必要があります。たとえば、自分のイニシャルと生年月日、またはその他の識別情報を使用して、セッション変数の一意性を確保してください。

セキュリティー保護されたセッション変数

セッション変数は名前、値、および `secure` フラグで構成されます。セッション変数名に関する制約は、ホストタイプ変数名の場合と同じです。`secure` フラグは、セキュリティーを保護して値を格納するかどうかを指定します。

値のセキュリティーを確保して格納するには、`secure` フラグを true に設定します。この場合、セッション変数の値は `***` で表示され、保存時には変数が暗号化されます。プランの実行時にこの値が使用された場合、この変数の値はプランの履歴に反映されません。この `secure` フラグは、パスワードなど、変数値が機密データの場合に使用します。セッション変数はデフォルトで、`secure` フラグを true に設定して作成されます。

セキュリティー保護された変数をデータベースに保存する場合は、暗号鍵として使用されている自分のパスワードを入力する必要があります。パスワードを間違って指定すると、パスワードの再入力が求められます。

入力したパスワードが認識されないと、エラーが発生します。システム管理者によってパスワードまたはログイン構成が変更されている場合などに、このよう状況が発生します。次の 2 つのオプションがあります。

- セッション変数の再暗号化**:** ユーザー名、セッション変数の暗号化に使用したパスワード、さらに新しいパスワードを入力します。
- セッション変数のフラッシュ**:** 宣言したすべてのセッション変数を削除します。ユーザー名と新しいパスワードを入力します。

  このオプションは、セッション変数の暗号化時に使用したパスワードを忘れた場合に使用します。

第 2 章

コンポーネント

この章では、コンポーネントを管理する方法について説明します。次の内容について説明します。

- 25 ページの「コンポーネントの管理」
- 29 ページの「コンポーネント CLI コマンドの概要」

コンポーネントの管理

コンポーネントの管理には、Sun N1 Service Provisioning System ソフトウェアのブラウザインタフェースを使用します。

コマンド行インタフェース (CLI) を使用してコンポーネントを管理することもできます。29 ページの「コンポーネント CLI コマンドの概要」を参照してください。CLI コマンドについての詳細は、『Sun N1 Service Provisioning System 5.1 コマンド行インタフェース(CLI) リファレンスマニュアル』を参照してください。

ブラウザインタフェースに表示されるページに、コンポーネントの表示方法と管理方法に関する情報があります。

- 「**Components**」ページ**:** コンポーネントの一覧を表示したり、リストに新しいコンポーネントを追加したりします。ほかのコンポーネントページにアクセスして、コンポーネントの詳細を表示することもできます。
  - リポジトリにチェックイン済みのコンポーネントを表示するには、ナビゲーションメニューから「Components」を選択します。
  - コンポーネントの詳細を表示するには、「Components」ページにアクセスして、該当するコンポーネント名をクリックすると、「Details」ページに詳細が表示されます。
  - コンポーネントを作成する場合は、27 ページの「コンポーネントを作成する」を参照してください。

- コンポーネントのインストール先を表示するには、コンポーネントの「Details」ページにアクセスして、「Where Installed」をクリックします。

- 「**Details**」ページ**:** コンポーネントの属性、値などのコンポーネントの詳細情報を表示します。このページは、コンポーネントを管理するための情報やボタンも提供します。

  「Details」ページに表示される情報は、コンポーネントのコンポーネントタイプによって決まります。

  - コンポーネントを編集するには、コンポーネントの「Details」ページにアクセスして、「Edit」をクリックします。「Edit」ページでコンポーネントの構成を変更して、「Check In」をクリックします。使用するバージョン番号を選択できます。その後、「Continue To Check In」をクリックします。
  - コンポーネントを削除する場合は、28 ページの「コンポーネントを削除する」を参照してください。
  - コンポーネントの名前を変更するには、コンポーネントの「Details」ページにアクセスして、「Rename」をクリックします。新しいコンポーネント名を指定して「Rename」をクリックします。コンポーネントの名前を変更しない場合は、「Cancel」をクリックします。
  - コンポーネントを別のフォルダに移動するには、コンポーネントの「Details」ページにアクセスして「Move」をクリックします。コンポーネントの移動先フォルダ名を指定して「Move」をクリックします。コンポーネントを移動しない場合は、「Cancel」をクリックします。

    注 **–** まだ存在しないフォルダにコンポーネントを移動する場合は、「Move Component」ウィンドウで「New Folder」をクリックします。この新しいフォルダは、親フォルダからアクセス許可を継承します。「Create」をクリックすると、新しいフォルダが選択された状態でフォルダツリーが表示されます。「Move」をクリックします。

  - コンポーネントの XML を編集するには、コンポーネントの「Details」ページにアクセスして「Advanced Edit」をクリックします。XML を適切に変更します。現在の XML を上書きする場合は、「Check In」をクリックします。変更した XML に基づいて新しいコンポーネントを作成する場合は、「Check In As」をクリックします。または、コンポーネントの XML を変更しない場合は、「Cancel」をクリックします。
  - システムにコンポーネントの XML をダウンロードするには、コンポーネントの「Details」ページにアクセスして「Download」をクリックします。ダウンロード処理を続ける場合は「OK」を、取り消す場合は「Cancel」をクリックします。コンポーネントの XML のダウンロード先ディレクトリを指定し、「Save」をクリックします。コンポーネントの XML をダウンロードしない場合は、「Cancel」 をクリックします。
  - コンポーネントの変数設定を変更するには、コンポーネントの「Details」ページにアクセスして、「Variable Settings」をクリックします。「Create Set」をクリックして新しい変数設定を作成します。または「Import Set」をクリック

して、システムから変数設定をインポートします。

- コンポーネントのインストール先を表示するには、コンポーネントの「Details」ページにアクセスして、「Where Installed」をクリックします。

- 「**Edit** 」ページ**:** コンポーネントの属性値を変更します。すべてのフィールドを変更できるわけではありません。

  このページに表示される情報は、コンポーネントのコンポーネントタイプによって決まります。

▼ コンポーネントを作成する

次の手順で、単純コンポーネントと複合コンポーネントを作成します。

手順 **1.** ナビゲーションメニューから「**Components**」を選択します。

「Components」ページに、チェックイン済みコンポーネントの一覧が表示されます。

2. (省略可能**)** 「**Change Folder**」をクリックし、新しいコンポーネントを追加するフォルダを指定します。

表示されたウィンドウで、コンポーネントを作成するフォルダの名前を指定します。

- フォルダが存在する場合は、フォルダ名を選択して「**Change to Selected Folder**」をクリックします。

  「Components」ページに、指定したフォルダにあるコンポーネントの一覧が表示されます。

- フォルダが存在しない場合は、フォルダを作成します。

  a. 親フォルダを選択し、「**New Folder**」をクリックします。

  「Create New Folder」ウィンドウが表示されます。

  b. 新しいフォルダの名前を入力し、「**Create**」をクリックします。

  新しいフォルダは、親フォルダからアクセス許可を継承します。「Create」をクリックすると、新しいフォルダが選択された状態でフォルダツリーが表示されます。

注 **–** 一度に複数のフォルダレベルを作成することはできません。複数の入れ子になったフォルダを作成するには、「Change Folder」ウィンドウに戻り、手順 a および b を繰り返す必要があります。

---

c. フォルダを作成したあとで、フォルダ名を選択して「**Change to Selected Folder**」をクリックします。

「Components」ページに、指定したフォルダにあるコンポーネントの一覧が表示されます。

3. 「**Component**」フィールドに新しいコンポーネントの名前を入力して「**Create**」をクリックします。

新しいコンポーネントの「Edit」ページが表示されます。

4. コンポーネントを定義します。

a. **(**省略可能**)** 「**Component**」フィールドでコンポーネント名を変更して「**Rename**」をクリックします。

b. ドロップダウンメニュー「**Component Type**」からコンポーネントタイプを選択します。

選択したコンポーネントタイプが使用するものに応じて、表示されるフィールドの一部が変わります。

c. ドロップダウンメニュー「**Platform Type**」から、このコンポーネントを配備できるプラットフォームの種類を選択します。

d. **(**省略可能**)** 「**Label**」フィールドにラベルを指定します。

e. **(**省略可能**)** 「**Description**」フィールドに説明を入力します。

f. その他の必須情報を入力します。

必須情報は選択したコンポーネントタイプに基づいて決まります。

このコンポーネントがほかのコンポーネントを参照する場合は、このコンポーネントが参照するコンポーネントごとにローカル名が作成されます。

g. 「**Check In**」をクリックします。

プロビジョニングシステムが、コンポーネントをバージョン 1.0 でチェックインすることを示すウィンドウが表示されます。

5. 「**Continue To Check In**」をクリックします。

▼ コンポーネントを削除する

コンポーネントを削除する前に、次の制限を考慮してください。

- 別のコンポーネントが参照しているコンポーネントは削除できません。
- インストールされたコンポーネントは、アンインストールされないかぎり削除できません。
- プラグインによって作成されたコンポーネントは、プラグインを削除しない限り、削除できません。

- ファミリのルートコンポーネントは、そのファミリのルート以外のメンバーが存在している場合、削除できません。
- プランの実行中はコンポーネントを削除できません。削除操作とプランの実行は例外なくシステムがロックされ、これらの操作を同時に実行できないようになるからです。

コンポーネントを削除すると、インストールレコードも削除されます。これが当てはまるのは、インストールとアンインストールが行われたコンポーネントに限られ、ユーザーが気づくような変化はありません。コンポーネントと関連付けられているリソースがある場合には、コンポーネントとともにそのリソースも削除されます。直接実行コンポーネント手続きを実行することで作成されたすべてのプランも削除されます。コンポーネントをインストールしたプランの履歴が編集され、プランでインストールしたコンポーネントが削除されたことが反映されます。プラン履歴そのものは削除されません。

手順

1. ナビゲーションメニューから「**Components**」を選択します。
   「Components」ページに、チェックイン済みコンポーネントの一覧が表示されます。
2. 「**Change Folder**」をクリックします。
   表示されたウィンドウで、コンポーネントを削除するフォルダの名前を指定します。
3. フォルダ名を指定して「**Change to Selected Folder**」をクリックします。
   「Components」ページに、指定したフォルダに含まれているコンポーネントの一覧が表示されます。
4. 該当するコンポーネント名をクリックします。
   そのコンポーネントの「Details」ページが表示されます。
5. 「**Delete**」をクリックします。
   選択されたコンポーネントをプロビジョニングシステムが削除することを示すウィンドウが表示されます。
6. 「**Continue To Delete**」をクリックします。
   コンポーネントを削除しない場合は、「Cancel」をクリックします。

コンポーネント CLI コマンドの概要

`cdb.c` コマンドを使用して、コンポーネントを管理できます。『Sun N1 Service Provisioning System 5.1 コマンド行インタフェース(CLI) リファレンスマニュアル』の第 3 章「cdb: コンポーネントを管理する CLI コマンド」を参照してください。

- cdb.c.ci – 既存コンポーネントの新しいバージョンを作成するか、または XML 形式を使用して非ブラウザ対応コンポーネントの初期バージョンを作成する
- cdb.c.co – XML 形式のコンポーネントをチェックアウトする
- cdb.c.del – コンポーネントを削除する
- cdb.c.la – すべてのコンポーネントのすべてのバージョンを表示する
- cdb.c.lo – コンポーネントの詳細情報を表示する
- cdb.c.lv – 特定のコンポーネントのすべてのバージョンを表示する
- cdb.c.mod – コンポーネントを変更する
- cdb.c.sc – コンポーネントに 1 つ以上のカテゴリを適用する
- cdb.c.sh – コンポーネントを表示する、またはコンポーネントを隠す

次の cdb.ic コマンドを使用して、ホストにインストール済みのコンポーネントに関する情報を検索します。

- cdb.ic.lbc – 特定のコンポーネントがインストールされているホストをすべて表示する
- cdb.ic.lbh – 特定のホストにインストールされているコンポーネントをすべて表示する
- cdb.ic.vs.lo – 生成された特定の変数設定の詳細を表示する

次の cdb.rsrc コマンドを使用してリソースを管理します。

- cdb.rsrc.ci – 特定のコンポーネントとそのソースオブジェクトをチェックインする
- cdb.rsrc.cib – バッチファイルに指定されたすべてのリソースをチェックインする
- cdb.rsrc.co – 指定されたリソースをチェックアウトする
- cdb.rsrc.gd – 指定されたコンポーネントのリソース記述子を生成する
- cdb.rsrc.rci – リソースを再びチェックインする
- cdb.rsrc.showopts – 特定のコンポーネントタイプでサポートされるチェックインオプションを表示する

次の cdb.vs コマンドで、コンポーネントの変数設定を管理します。

- cdb.vs.add – 新しい変数設定を追加する
- cdb.vs.del – 変数設定を削除する
- cdb.vs.imp – あるコンポーネントから別のコンポーネントに変数設定をインポートする
- cdb.vs.la – 特定のコンポーネントに関連付けられているすべての変数設定を表示する
- cdb.vs.lo – 特定の変数設定を詳細を表示する
- cdb.vs.mod – 変数設定を変更する

コマンド行インタフェースによるコンポーネントのチェックイン

コンポーネントをチェックインするときに、ゴールドサーバー上のディレクトリなどのデータソースからコンポーネントリポジトリに、特定のリソースをコピーします。*component repository*は、階層型の名前空間です。この名前空間内でコンポーネントは、名前とバージョン番号で識別されます。

コンポーネントには、通常、コンポーネントの形式と機能を特定するコンポーネントタイプを設定します。プロビジョニングシステムで使用できる組み込みコンポーネントタイプについては、第 3 章を参照してください。

`cdb.rsrc.ci` コマンドを使用してコンポーネントをチェックインするときには、次のオプションを指定します。

- `-src` – リソースのソース位置を指定する
- `-dst` – コンポーネントリポジトリのどこにコンポーネントを格納するのか、その場所を指定する
- `-type` – コンポーネントタイプを指定する

複数のコンポーネントで同じリソースを参照できます。`cdb.rsrc.ci` コマンドを使用してリソースをチェックインすると、そのリソースが指定したコンポーネントに関連付けられます。

第 3 章

組み込みコンポーネントタイプ

組み込みコンポーネントタイプを使用すると、WebLogic、Microsoft Windows、および J2EE アプリケーションの一般的なコンポーネントをすばやくモデル化できます。これらのコンポーネントタイプにより、インストール、アンインストール、エクスポート、スナップショットなどの動作を、特定のコンポーネントと自動的に関連付けることができます。

基本製品の一部として、次のコンポーネントタイプを利用できます。

- 33 ページの「コンポーネントタイプ: system#file」
- 35 ページの「コンポーネントタイプ: system#directory」
- 37 ページの「コンポーネントタイプ: system#symbolic link」
- 39 ページの「コンポーネントタイプ: system#container」
- 40 ページの「コンポーネントタイプ: untyped」

WebLogic や Microsoft Windows のアプリケーションなどのコンポーネントタイプは、プラグインにより利用可能です。詳細については、http://docs.sun.com/app/docs/coll/1329.1を参照してください。

コンポーネントタイプ: system#file

コンポーネントタイプ system#file は、ターゲットサーバーから取得した単一ファイルを意味します。プロビジョニングシステムは、特別な後処理を行わなくても、ファイルを直接配備します。コンポーネントタイプ system#file には、インストール、アンインストール、およびスナップショット手続きが含まれています。

コンポーネントタイプ system#file では、次の変数を定義します。

- installName – インストールしたリソースに使用する名前。デフォルト値はコンポーネント名です。
- installPath – リソースをインストールするパス。デフォルト値は、コンテナコンポーネントの installPath 変数の値です。

- `installPermissions` – インストールしたリソースに割り当てるアクセス許可。`chmod(1M)` のマニュアルページを参照してください。デフォルト値は空です。
- `installUser` – このリソースをインストールしたときの所有者。デフォルト値は空です。
- `installGroup` – このリソースをインストールしたときに割り当てるグループ。デフォルト値は空です。
- `installDiffDeploy` – リソースを差分配備モードで配備するかどうかを指定します。デフォルト値は `TRUE` です。
- `overrideRsrcInstallPath` – リソースをインストールする絶対パス。未定義の場合、`installPath` によって指定されたパスにインストールされます。この値は通常使用しません。`installPath` を使用すると、コンポーネントとリソースの両方について、インストールするパスを指定できるからです。

ブラウズ

	UNIX システム	**Microsoft Windows** システム
ルートパス	/	ホストまたはネットワーク上の物理ドライブリストをドライブ文字に対応づけます。リムーバブルメディアは表示されません。
区切り文字	/	\
順序付け	ディレクトリのアルファベット順に表示	
選択方式	チェックインするファイルをユーザーが選択できます。ディレクトリをダブルクリックすると、ディレクトリの内容が表示されます。	
サンプルパス	`/foo/foo.txt`	`C:\foo\foo.txt`
特別	リンクは、ローカル名と指定された場所を示します。 `foo->/usr/bar`	

拡張制御手続き

コンポーネントタイプ `system#file` には次の手続きが定義されています。

- `default` **install block** – 変数の状態に基づいて包含ファイルを配備します。スナップショットは取得されません。
- `markOnly` **install block** – 新しいバージョンのコンポーネントがインストールされたことを明確にします。リソースの転送もスナップショットの取得も行われません。

- `default` **uninstall block –** 包含ファイルの配備を解除して、最初に配備されていた場所に戻します。
- `markOnly` **uninstall block –** 現在のバージョンのコンポーネントがアンインストールされたことを明確にします。コンポーネントのリソースがターゲットホストから実際に削除されるわけではありません。
- `default` **snapshot block –** 配備されたファイルのスナップショットを取得します。デフォルトインストールブロックは、デフォルトではファイルをキャプチャしないので、スナップショットが必要な場合はコンテナコンポーネントでこのルーチンを明示的に呼び出す必要があります。

コンポーネントタイプ: system#directory

プロビジョニングシステムは任意の数のコンポーネントを収容し、ユーザーが作成する任意のディレクトリ構造にそれらを配置することができます。作成可能なコンポーネントの数は、マスターサーバー上のファイルシステムによってのみ制限されます。各ファイルシステムは作成可能なファイルの数を制限します。単一のディレクトリに作成するファイルの数が多過ぎると、ファイルシステムの制限を超える可能性があります。コンポーネントとファイルの間の関係は、必ずしも同比率ではありません。たとえば、あるコンポーネントに 100,000 個のファイルが含まれる場合、そのコンポーネントはファイルシステムの制限を超えることもありますが、ファイルを含むコンポーネントがない場合、 100,000 個のコンポーネントでもファイルシステムの制限に収まることもあります。

ファイルシステムの制限を超えても、ファイルの作成時には直接エラーが生じない場合があります。ただし、問題は別の機会に、パフォーマンス劣化の深刻化や、予測不可能な障害の発生といった形で生じる可能性があります。

1 つのディレクトリがサポートできるファイルの数に対する制限はファイルシステムにより異なり、またオペレーティングシステムの構成方法の影響を受ける可能性があります。使用するファイルシステムの制限を理解し、単一の大きなディレクトリではなく小さな複数のサブディレクトリにコンポーネントを配置することで、ディレクトリの負荷超過を回避できます。コンポーネントを作成する際に使用する安全な方法としては、1 つのディレクトリに配置するファイル数を 30,000 以下にします。

コンポーネントタイプ `system#directory` は、ターゲットサーバーから取得されたファイルとフォルダの集まりを表します。コンポーネントタイプ `system#directory` には、インストール、アンインストール、およびスナップショット手続きが含まれています。

コンポーネントタイプ `system#directory` では次の変数を定義します。

- `installName` – インストールしたリソースに使用する名前。デフォルト値はコンポーネント名です。

- installPath – リソースをインストールするパス。デフォルト値は、コンテナコンポーネントの installPath 変数の値です。
- installPermissions – インストールしたリソースに割り当てるアクセス許可。chmod(1M) のマニュアルページを参照してください。デフォルト値は空です。
- installUser – このリソースをインストールしたときの所有者。デフォルト値は空です。
- installGroup – このリソースをインストールしたときに割り当てるグループ。デフォルト値は空です。
- installDiffDeploy – リソースを差分配備モードで配備するかどうかを指定します。デフォルト値は TRUE です。
- overrideRsrcInstallPath – リソースをインストールする絶対パス。未定義の場合、installPath によって指定されたパスにインストールされます。この値は通常使用しません。installPath を使用すると、コンポーネントとリソースの両方について、インストールするパスを指定できるからです。
- installDeployMode – ファイルをディレクトリに追加する (ADD_TO) か、ディレクトリのファイルを置き換える (REPLACE) かを指定します。デフォルト値は REPLACE です。

ブラウズ

	UNIX システム	**Microsoft Windows** システム
ルートパス	/	ホストまたはネットワーク上の物理ドライブリストをドライブ文字に対応づけます。リムーバブルメディアは表示されません。
区切り文字	/	\
順序付け	ディレクトリのアルファベット順に表示	
選択方式	チェックインするディレクトリをユーザーが選択できます。ディレクトリをダブルクリックすると、ディレクトリの内容が表示されます。	
サンプルパス	/foo/foo/	C:\foo\foo\
フィルタ	なし	
特別	リンクは、ローカル名と指定された場所を示します。 foo/->/usr/bar/	

拡張制御手続き

コンポーネントタイプ system#directory には次の手続きが定義されています。

- default **install block –** 変数の状態に基づいて包含ファイルを配備します。スナップショットは取得されません。
- markOnly **install block –** 新しいバージョンのコンポーネントがインストールされたことを明確にします。リソースの転送もスナップショットの取得も行われません。
- default **uninstall block –** 包含ファイルの配備を解除して、最初に配備されていた場所に戻します。
- markOnly **uninstall block –** 現在のバージョンのコンポーネントがアンインストールされたことを明確にします。コンポーネントのリソースがターゲットホストから実際に削除されるわけではありません。
- default **snapshot block –** 配備されたファイルのスナップショットを取得します。デフォルトインストールブロックは、デフォルトではファイルをキャプチャしないので、スナップショットが必要な場合はコンテナコンポーネントでこのルーチンを明示的に呼び出す必要があります。

コンポーネントタイプ: system#symbolic link

コンポーネントタイプ system#symbolic link は、ターゲットサーバーから取得した単一のシンボリックリンクを表します。プロビジョニングシステムは、ターゲットサーバー上に新しいリンクを作成することによって、シンボリックリンクを配備します。シンボリックリンクが示すファイルは、symlinkTarget 変数の値に基づきます。リンクのインストール位置にファイルが存在している場合は、新しいシンボリックリンクを作成する前に、それらのファイルが削除されます。

コンポーネントタイプ system#symbolic link には、インストール、アンインストール、およびスナップショット手続きが含まれています。アンインストール手続きは、結合先のファイルではなく、シンボリックリンクを削除します。スナップショット手続きも同様に、ファイルそのものではなく、シンボリックリンクが示すファイルの名前を取得します。

コンポーネントタイプ system#symbolic link では、次の変数を定義します。

- installName – インストールしたリソースに使用する名前。デフォルト値はコンポーネント名です。
- installPath – リソースをインストールするパス。デフォルト値は、コンテナコンポーネントの installPath 変数の値です。
- symlinkTarget – このリンクによって結合されるファイルの絶対パス。デフォルト値は、ファイルが本来リンクされていたファイルです。

コンポーネントタイプ system#symbolic link には次の手続きが定義されています。

- default**install block –** インストールパスにすでに存在していたファイルを削除して、symlinkTarget 変数の値にリンクするシンボリックリンクに置き換えます。スナップショットは取得されません。
- default**uninstall block –** リンク先のファイルを削除しないで、配備されたリンクを削除します。
- default **snapshot block –** 配備されたリンクのスナップショットを取得します。デフォルトインストールブロックは、デフォルトではファイルをキャプチャしないので、スナップショットが必要な場合はコンテナコンポーネントでこのルーチンを明示的に呼び出す必要があります。シンボリックリンクのスナップショットを要求すると、そのシンボリックリンクが示すパスが取得されます。

ブラウズ

	UNIX システム	**Microsoft Windows** システム
ルートパス	/	ホストまたはネットワーク上の物理ドライブリストをドライブ文字に対応づけます。リムーバブルメディアは表示されません。
区切り文字	/	\
順序付け	ディレクトリのアルファベット順に表示	
選択方式	チェックインするファイルをユーザーが選択できます。ディレクトリをダブルクリックすると、ディレクトリの内容が表示されます。チェックインを選択できるのは、シンボリックリンク型のファイルだけです。	
サンプルパス	/foo/foo.txt	C:\foo\foo.txt
特別	リンクは、ローカル名と指定された場所を示します。 foo->/usr/bar	

エクスポート/内部ファイル形式

シンボリックリンクには、シンボリックプレースホルダの役割を果たすリソースが含まれます。シンボリックリンクデータは変数セットとして、名前と場所ごとに 1 つずつコンポーネントに格納されます。

コンポーネントタイプ: system#container

コンポーネントをグループにまとめ、1 つのコンポーネントとしてインストールしなければならない場合は、必ず、コンポーネントタイプ system#container を使用します。コンテナコンポーネントにはインストールブロック、アンインストールブロック、およびスナップショットブロックがあります。これらのブロックは子コンポーネントを自動的に呼び出します。このような呼び出しによって、コンテナコンポーネントの XML を編集しなくても、コンテナは Component Builder ユーザーインタフェースを使用することによって、子コンポーネントを直接追加できます。

ブラウズ

コンテナコンポーネントタイプは複合コンポーネントであり、ブラウズできません。

モデルとインストールの差分

この比較では、任意ですべての子コンポーネントのスナップショットを得ます。

default スナップショットブロックは、各子コンポーネントの default スナップショットブロックを使用し、installMode="NESTED" が設定されているすべての子コンポーネントのスナップショットを取得します。

拡張制御手続き

コンポーネントタイプ system#container には次の手続きが定義されています。

- default **install block –** installMode="NESTED" が設定されている子コンポーネントだけをインストールします。
- markOnly **install block –** すべての入れ子になった子コンポーネントの markOnly インストールブロックを呼び出し、このコンポーネントにインストール済みのマークを付けます。
- default **uninstall block –** installMode="NESTED" が設定されている子コンポーネントだけをアンインストールします。
- markOnly **uninstall block –** すべての入れ子になった子コンポーネントの markOnly アンインストールブロックを呼び出し、このコンポーネントにアンインストール済みのマークを付けます。
- default **snapshot block –** createSnapshot が true に設定されている場合は、すべての子コンポーネントのスナップショットを取得します。installMode="NESTED" が設定されている子コンポーネントだけがインストールされます。createSnapshot はデフォルトで false に設定されます。

スナップショットは常に有効なわけではありません。別のコンポーネントの入れ子コンポーネントとして配備されているコンポーネントは、createSnapshot="false" を設定して配備してください。すべての包含コンポーネントが配備されたあとで、コンテナコンポーネントがこのコンポーネントのスナップショットを取得するからです。最上位コンポーネントとして配備されているコンポーネントは、createSnapshot="true" を設定することによってスナップショットが使用可能になります。

コンポーネントがコンテナコンポーネントに含まれている場合、スナップショットは取得されません。このコンポーネントのあとでほかの包含コンポーネントが配備されると、このコンポーネントのインストール状態に影響を及ぼす可能性があります。したがって、スナップショットは上位レベルのインストールが完了してから取得します。

コンポーネントタイプ: untyped

untyped タイプのコンポーネントの拡張はベースコンポーネントなしに行われるため、デフォルト動作はありません。このタイプのコンポーネントは、カスタムタイプや XML をコンポーネントに追加する場合で組み込みタイプのデフォルト動作のどれもが関連性を持っていないときに、高度な編集を行うためのものです。

第 4 章

プラン

プロビジョニングシステムによって管理されるホスト上で処理を実行するには、プランを使用します。

この章の内容は次のとおりです。

- 41 ページの「プランの管理」
- 43 ページの「プランの作成」
- 46 ページの「プランの実行」
- 51 ページの「プラン CLI コマンドの概要」

プランの管理

プランの管理には、Sun N1 Service Provisioning System ソフトウェアのブラウザインタフェースを使用します。ブラウザインタフェースに表示されるページに、プランの表示方法と管理方法に関する情報があります。

- 「**Plans**」ページ**:** リストにプランを表示したり、リストに新しいプランを追加したりします。ほかのプランページにアクセスして、プランの詳細を表示することもできます。

  チェックイン済みのプランを表示するには、ナビゲーションメニューから、「Plans」を選択します。
- 「**Details**」ページ**:** プランの属性、値などの詳細情報を表示します。このページは、プランを管理するための情報やボタンも提供します。
  - プランの詳細を表示するには、「Plans」ページにアクセスし、該当するプラン名をクリックします。「Details」ページに詳細が表示されます。
  - プランを削除するには、そのプランの「Details」ページにアクセスして「Delete」をクリックします。「Continue To Delete」をクリックします。プランを削除しない場合は、「Cancel」をクリックします。

注 – 生成されたプランや直接実行コンポーネント手続きを削除する必要はありません。これらのプランは、生成元のコンポーネントが削除された時点で、システムから自動的に削除されます。

- プランを別のフォルダに移動するには、そのプランの「Details」ページにアクセスし、「Move」をクリックします。プランの移動先のフォルダ名を指定し、「Move」をクリックします。プランを移動しない場合は「Cancel」をクリックします。

注 – まだ存在しないフォルダにプランを移動する場合は、「Move Component」ウィンドウで「New Folder」をクリックします。この新しいフォルダは、親フォルダからアクセス許可を継承します。「Create」をクリックした後、新しいフォルダが選択された状態でフォルダツリーが表示されます。「Move」をクリックします。

- 「**Advanced Edit**」ページ**:** プランの XML を変更することによって、プランで実行する処理を変更します。

  プランを編集するには、42 ページの「プランを編集する」を参照してください。

プランを管理する以外に、プランの作成および実行を行えます。それについては、この章の後で説明します。

CLI でプランを処理することもできます。51 ページの「プラン CLI コマンドの概要」を参照してください。CLI コマンドについての詳細は、『Sun N1 Service Provisioning System 5.1 コマンド行インタフェース(CLI) リファレンスマニュアル』を参照してください。

▼ プランを編集する

手順 1. 該当するプランの「**Details**」ページにアクセスして、「**Advanced Edit**」をクリックします。

2. **XML** を変更するか、またはそのプランの新しい **XML** をインポートします。

   別のシステムからプランをインポートするには、次のどちらか一方を実行します。

   - ファイルのフルパスを入力して「**Replace**」をクリックします。
   - 「**Browse**」をクリックしてファイルを検索し、「**Replace**」をクリックします。

3. プランをチェックインします。

- 「**Check In**」をクリックして、同じ名前でプランを保存します。
  「Plans」ページが表示されます。
- 「**Check In As**」をクリックして、別の名前でプランを保存します。
  プランをチェックインすることを示す新しいページが表示されます。バージョン番号が増します。

4. 「**Continue To Check In**」をクリックします。

プランの作成

プランの生成、カスタムプランの作成、生成されたプランのカスタマイズが行えます。

プロビジョニングシステムでは、複数の *direct-run component procedures*からなるプランを自動生成できます。このプランは、そのまま使用することも、XML を使ってより複雑なプランのテンプレートとして保存することもできます。

複数のコンポーネント、複数のホストセット、またはその両方の調整を必要とする配備では、プランを作成し、プロビジョニングシステム XML スキーマを使用して処理を定義します。このような処理として、依存性チェック、およびアプリケーションのコンソール上でコマンドを実行するスクリプトの作成などがあります。作成したプランは、プランリポジトリにチェックインする必要があります。

▼ プランを生成する

プランの生成は、コンポーネントの「Details」ページから行います。

始める前に　プランを生成するためには、プランを生成するコンポーネントが含まれるフォルダに対して、Create、Edit 、Delete のアクセス許可を持つユーザーグループに属する必要があります。

手順

1. ナビゲーションメニューから「**Components**」を選択します。
   「Components」ページに、チェックイン済みコンポーネントの一覧が表示されます。
2. 必要に応じて、配備するコンポーネントを表示できるように、フォルダを変更します。
   a. 「**Change Folder**」をクリックします。

b. コンポーネントが含まれるフォルダ名を指定し、「**Change to Selected Folder**」をクリックします。

「Components」ページに、指定したフォルダに含まれているコンポーネントの一覧が表示されます。

3. コンポーネントの一覧表から、配備するコンポーネントの記述が含まれる行を選択して、「**Details**」をクリックします。

コンポーネントの「Details」ページが表示されます。

4. 「**Component Procedures**」表から、プランに含める手続きを選択します。

5. 「**Component Procedures**」表の下部にある「**Generate Plan With Checked Procedures**」をクリックします。

生成されたプランの「Advanced Edit」ページが表示されます。

6. 「**Plan**」フィールドに新しいプランの名前を入力します。

7. 「**Check In**」をクリックします。

注 – 次のようなメッセージが表示される場合があります。「`Warning - plan names and/or paths differ`」。これは、指定したプラン名が XML モデルで指定されたプラン名と異なることを意味します。フィールドに入力したプラン名に基づいて、XML モデルが更新されます。

---

8. コンポーネントのチェックインを確認します。

a. 指定したプラン名が正しいかどうかを確認します。

b. 必要に応じて、コンポーネントの新しいバージョン番号を選択します。

c. 「**Continue To Check In**」をクリックします。

新しいプランの「Details」ページが表示されます。

▼ カスタムプランを作成する

プランは、TurboXML などのスキーマ検証エディタを使用して作成するか、ブラウザの「Advanced Edit」ページ上で作成できます。

ここでは、XML エディタで作成したプランをインポートする手順について説明します。

手順 **1.** プランを作成します。

XML スキーマ要素については、『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の第 2 章「コンポーネントと単純プランにより使用さ

れる共有スキーマ」と『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の第 4 章「プランのスキーマ」を参照してください。

2. ブラウザインタフェースを起動し、「**Plans**」ページにアクセスします。

3. **(省略可能)** 「**Change Folder**」をクリックし、新しいプランを追加するフォルダを指定します。

   表示されたウィンドウで、コンポーネントを作成するフォルダの名前を指定します。

   - フォルダが存在する場合は、フォルダ名を選択して「**Change to Selected Folder**」をクリックします。

     「Plans」ページには、指定したフォルダにあるプランのリストが表示されます。

   - フォルダが存在しない場合は、フォルダを作成します。

     a. 親フォルダを選択し、「**New Folder**」をクリックします。

        「Create New Folder」ウィンドウが表示されます。

     b. 新しいフォルダの名前を入力し、「**Create**」をクリックします。

        新しいフォルダは、親フォルダからアクセス許可を継承します。「Create」をクリックしたあと、新しいフォルダが選択された状態でフォルダツリーが表示されます。

        注 – 一度に複数のフォルダレベルを作成することはできません。複数の入れ子になったフォルダを作成するには、「Change Folder」ウィンドウに戻り、手順 a および b を繰り返す必要があります。

        ---

     c. フォルダを作成したあとで、フォルダ名を選択して「**Change to Selected Folder**」をクリックします。

        「Plans」ページには、指定したフォルダにあるプランのリストが表示されます。

4. 作成するプランの名前と簡単な説明を入力して「**Create**」をクリックします。

   そのプランの「Advanced Edit」ページが表示されます。

   プロビジョニングシステムは、そのプランの XML スケルトンを作成します。

5. 「**Plan Definition**」フィールドで次の作業を行います。

   - プランの **XML** を入力します。

   - ローカルシステムに格納されているプランをインポートする場合は、ファイルのフルパス名を入力するか、または「**Browse**」ボタンをクリックします。次に、「**Replace**」をクリックします。

6. 「**Check In**」をクリックします。
   プランをチェックインすることを示すウィンドウが表示されます。プランにはバージョン 1.0 が割り当てられます。

7. 「**Continue To Check In**」をクリックします。

プランの実行

プランを実行するには、そのプランを事前にプランリポジトリにチェックインしておく必要があります。プランを実行すると、指定したホスト上でそのプランが処理を実行します。

preflight は、シミュレーションされた UNIX® 環境に対するプランのシミュレートを実行し、配備に影響を与える可能性のあるエラーまたは潜在的なエラーの有無を調べて報告します。予備テストは必ず、配備に先行しますが、単独の処理として実行できます。

詳細なプリフライトを実行することもできます。この場合は次の作業を行います。

- 可用性およびターゲットホストへの接続を調べる
- アクセス許可を確認する
- 制約の妥当性を検査する

▼ プランを実行する

次の手順に従い、ブラウザインタフェースでプランの「Details」ページからプランを実行します。

始める前に　プランの変数を設定するために必要なインストール情報が揃っているかどうかを確認してください。

プランを実行するには、実行するプランが含まれるフォルダに対して、`Allow on Host Set` 許可を持つユーザーグループに属する必要があります。

手順

1. プランの「**Details**」ページにアクセスして、「**Run**」をクリックします。
   プランの「Run」ページが表示されます。

2. 「**Plan Parameters**」領域で、配備するコンポーネントに適した変数設定を選択します。

   - このコンポーネント用の変数設定がすでにある場合は、メニューから該当する設定値を選択します。

- メニューに適切な設定値がない場合は、「**Select From List**」をクリックします。
  「Select Variable Settings From List」ウィンドウが表示されます。
  - 別のコンポーネントの変数設定を使用する場合は、「**Import Set**」をクリックします。
    「Import Variable Settings」ウィンドウが表示されます。
    a. 必要に応じて、インポートする変数設定を持つコンポーネントが含まれているフォルダにアクセスします。
    b. コンポーネントのバージョンを選択します。

    注 **–** 変数設定はコンポーネントのバージョンごとに異なる場合があります。
    ---

    c. 「**Import Variable Settings**」をクリックします。
    インポートされた変数設定が表に表示されます。
    d. 「**Select**」をクリックします。
  - 新しい変数設定を作成する場合は、「**Create Set**」をクリックします。
    「Select Variable Settings From List」ウィンドウが表示されます。
    a. 表上部のテキストフィールドに、新しい変数設定に使用する名前を入力します。
    b. 変更するコンポーネント変数を選択します。
    表のセルが強調表示され、テキストフィールドが表示されます。
    c. コンポーネント変数の新しい値を入力します。
    d. (省略可能**)** 変更する変数値ごとに、手順 **b** と **c** を繰り返します。
    e. 変数設定値の更新後、「**Save**」をクリックします。
    テーブルに新しい変数設定が表示されます。
    f. 「**Select**」をクリックします。

3. コンポーネントの配備先を指定します。

- ホストを指定するには、「**Target Host**」フィールドにホスト名を入力するか、「**Select From List**」をクリックしてホストを選択します。
- ターゲットホストセットを選択する場合は、「**Target Host Set**」メニューからホストセットを **1** つ選択します。

4. 「**Plan Variables**」領域で、設定する変数を構成します。
   構成する変数があるかどうかは、プランの内容によって決まります。

5. 「**Options**」領域で、詳細プリフライトを実行するかどうかを指定します。
   このプリフライトでは、リモートエージェント上で実行されるプランの各手順をシミュレートすることにより、詳細プリフライトの内容は次のとおりです。

   - リソースのインストール
   - `<execNative>` 呼び出し
   - ファイル変換
   - ファイルの削除 (リソースのアンインストール時)

   詳細プリフライトを実行すると、プリフライトの所要時間が長くなります。

6. **(**省略可能**)** 同時に稼働するホスト数を制限するには、用意されたフィールドにプランを実行するホスト数を入力します。
   複数のホストで同時に同じプランを実行する場合、すべてのホストで 1 つのステップを完了しなければ、どのホストも次のステップを開始できません。ホスト数が多すぎると、次のステップが開始されるまでにネットワーク接続がタイムアウトする可能性があります。

   たとえば、ホスト A が最初のステップを完了してから、次のステップを開始するまでの間隔が長すぎる場合があります。この場合、間隔が長すぎることが原因でネットワーク接続がタイムアウトします。

7. **(**省略可能**)** プランまたはネイティブ呼び出しの全体的な実行時間を制限するには、用意されたフィールドに値を入力し、メニューから **minute (**分**)** などの時間単位を選択します。
   プランまたはネイティブ呼び出しの実行時間を制限すると、応答しないホストがあるためにプランの実行が途中で停止するのを防ぐことができます。

   プランの失敗を知らせる通知規則を設定しておくと、プランがタイムアウトした場合に、エラーが通知されます。

   通知規則の設定方法については、『Sun N1 Service Provisioning System 5.1 システム管理者ガイド』の「電子メール通知の構成」を参照してください。

8. コンポーネントを配備するかどうかをを指定します。

   - 手続きのプリフライトを実行するだけの場合は、「**Run Preflight Only**」をクリックします。

   - プリフライトに続いて手続きそのものを実行する場合は、「**Run Plan (Includes Preflight)**」をクリックします。
     プリフライトが失敗した場合、手続きは実行されません。

▼ 直接実行コンポーネント手続きを使用してコンポーネントを配備する

コンポーネントのほとんどに、インストール、アンインストール、およびスナップショット手続きが含まれています。インストール済みのコンポーネントを管理するための制御手続きがコンポーネントに含まれる場合もあります。たとえば、制御手続きによってアプリケーションの起動や停止を行えます。

コンポーネントのチェックイン時にプロビジョニングシステムによって自動的に生成される、直接実行コンポーネント手続きの 1 つを使用できる場合もあります。この方式は、複数のコンポーネントやホスト間の同期が不要な単純な配備に使用します。

手順

1. ナビゲーションメニューから「**Components**」を選択します。
   「Components」ページに、チェックイン済みコンポーネントの一覧が表示されます。
2. 必要に応じて、配備するコンポーネントを表示できるように、フォルダを変更します。
   a. 「**Change Folder**」をクリックします。
   b. コンポーネントが含まれるフォルダ名を指定し、「**Change to Selected Folder**」をクリックします。
      「Components」ページに、指定したフォルダに含まれているコンポーネントの一覧が表示されます。
3. 該当するコンポーネント名をクリックします。
   そのコンポーネントの「Details」ページが表示されます。
4. **Component Procedures** 表で、実行する手続きを決定して「**Run**」を実行します。
   プロビジョニングシステムによってプランが生成され、`/system/autogen` フォルダに格納されます。
   生成されたプランの「Run」ページが表示されます。
5. 「**Plan Parameters**」領域で、配備するコンポーネントに適した変数設定を選択します。
   - このコンポーネント用の変数設定がすでにある場合は、メニューから該当する設定値を選択します。
   - メニューに適切な設定値がない場合は、「**Select From List**」をクリックします。
     「Select Variable Settings From List」ウィンドウが表示されます。
     - 別のコンポーネントの変数設定を使用する場合は、「**Import Set**」をクリックします。
       「Import Variable Settings」ウィンドウが表示されます。

a. 必要に応じて、インポートする変数設定を持つコンポーネントが含まれているフォルダにアクセスします。

b. コンポーネントのバージョンを選択します。

注 **–** 変数設定はコンポーネントのバージョンごとに異なる場合があります。

---

c. 「**Import Variable Settings**」をクリックします。
インポートされた変数設定が表に表示されます。

d. 「**Select**」をクリックします。

- 新しい変数設定を作成する場合は、「**Create Set**」をクリックします。
「Select Variable Settings From List」ウィンドウが表示されます。

a. 表上部のテキストフィールドに、新しい変数設定に使用する名前を入力します。

b. 変更するコンポーネント変数を選択します。
表のセルが強調表示され、テキストフィールドが表示されます。

c. コンポーネント変数の新しい値を入力します。

d. **(**省略可能**)** 変更する変数値ごとに、手順 **b** と **c** を繰り返します。

e. 変数設定値の更新後、「**Save**」をクリックします。
テーブルに新しい変数設定が表示されます。

f. 「**Select**」をクリックします。

6. コンポーネントの配備先を指定します。

- ホストを指定するには、「**Target Host**」フィールドにホスト名を入力するか、「**Select From List**」をクリックしてホストを選択します。
- ターゲットホストセットを選択する場合は、「**Target Host Set**」メニューからホストセットを **1** つ選択します。

7. 「**Plan Variables**」領域で、設定する変数を構成します。
構成する変数の有無は、選択したコンポーネント手続きの内容によって決まります。

8. 「**Options**」領域で、詳細プリフライトを実行するかどうかを選択します。
このプリフライトでは、リモートエージェント上で実行されるプランの各手順をシミュレートすることにより、詳細プリフライトの内容は次のとおりです。

- リソースのインストール

- <execNative> 呼び出し
- ファイル変換
- ファイルの削除 (リソースのアンインストール時)

詳細プリフライトを実行すると、プリフライトの所要時間が長くなります。

9. **(**省略可能**)** 同時に稼働するホスト数を制限するには、用意されたフィールドにプランを実行するホスト数を入力します。

   複数のホストで同時に同じプランを実行する場合、すべてのホストで 1 つのステップを完了しなければ、どのホストも次のステップを開始できません。ホスト数が多すぎると、次のステップが開始されるまでにネットワーク接続がタイムアウトする可能性があります。

   たとえば、ホスト A が最初のステップを完了してから、次のステップを開始するまでの間隔が長すぎる場合があります。この場合、間隔が長すぎることが原因でネットワーク接続がタイムアウトします。

10. **(**省略可能**)** プランまたはネイティブ呼び出しの全体的な実行時間を制限するには、用意されたフィールドに値を入力し、メニューから **minute (**分**)** などの時間単位を選択します。

    プランまたはネイティブ呼び出しの実行時間を制限すると、応答しないホストがあるためにプランの実行が途中で停止するのを防ぐことができます。

    プランの失敗を知らせる通知規則を設定しておくと、プランがタイムアウトした場合に、エラーが通知されます。

    通知規則の設定方法については、『Sun N1 Service Provisioning System 5.1 システム管理者ガイド』の「電子メール通知の構成」を参照してください。

11. コンポーネントを配備するかどうかをを指定します。

    - 手続きのプリフライトを実行するだけの場合は、「**Run Preflight Only**」をクリックします。
    - プリフライトに続いて手続きそのものを実行する場合は、「**Run Plan (Includes Preflight)**」をクリックします。

      プリフライトが失敗した場合、手続きは実行されません。

プラン CLI コマンドの概要

次の pdb コマンドを使用して、プランを管理できます。

- pdb.p.ci – 新しいバージョンのプランをチェックインする
- pdb.p.co – XML 形式のプランをチェックアウトする
- pdb.p.del – プランを削除する

- pdb.p.genplan – XML 形式のプランを生成して出力を表示する
- pdb.p.la – すべてのプランのすべてのバージョンを表示する
- pdb.p.lo – プランの詳細情報を表示する
- pdb.p.lv – 特定のプランのすべてのバージョンを表示する
- pdb.p.sc – プランを一連のカテゴリに関連付ける
- pdb.p.sh – プランを表示する、または隠す

次の pe コマンドは、実行中のプランと関連付けられます。

- pe.h.prep – 1 組のホストを準備する
- pe.p.del – 実行が完了したプランの履歴を削除する
- pe.p.en – <execNative> ステップの出力を表示する
- pe.p.la – 実行中のプランと完了したプランの一覧を表示する
- pe.p.lo – 実行中のプランと完了したプランの詳細情報を表示する
- pe.p.lp – プランに関連付けられているサブプランとターゲットを表示する
- pe.p.run – プランを実行する
- pe.p.stop – 実行中のプランを中止する
- pe.pi.lo – プランの実行に使用されるパラメータを表示する

プロビジョニングシステムの CLI コマンドについての詳細は、『Sun N1 Service Provisioning System 5.1 コマンド行インタフェース(CLI) リファレンスマニュアル』を参照してください。

第 5 章

セッション変数

この章では、セッション変数とその管理方法について説明します。

この章の内容は次のとおりです。

- 53 ページの「セッション変数の管理」
- 54 ページの「セッション変数 CLI コマンドの概要」

セッション変数の管理

*Session variables*は、配備に関連付けられる大域値であり、プロビジョニングシステムの柔軟性を拡大します。作成したセッション変数は、任意の配備で将来使用できるように保存することも、保存しないで 1 回限りの配備で使用することもできます。

セッション変数の管理には、Sun N1 Service Provisioning System ソフトウェアのブラウザインタフェースを使用します。セッション変数情報は、ブラウザインタフェースの次のページに含まれます。

- 「**Session Variables**」ページ**:** セッション変数を表示し、新しいセッション変数をリストに追加します。ほかのセッション変数ページにアクセスして、詳細を表示することもできます。

  チェックイン済みのセッション変数を表示するには、ナビゲーションメニューから「Session Variables」を選択します。
- 「**Details**」ページ**:** セッション変数の属性および値などの詳細情報を表示します。このページは、セッション変数を管理するための情報やボタンも提供します。

  セッション変数の詳細を表示するには、「Session Variables」ページにアクセスし、該当するセッション変数名をクリックします。「Session Variables Details」ページに詳細が表示されます。
- 「**Edit**」ページ**:** セッション変数情報を変更します。

セッション変数を編集するには、「Session Variables Details」ページにアクセスして「Edit」をクリックします。「Session Variables Edit」ページでセッション変数を変更します。「Save」をクリックして変更を保存するか、または「Cancel」をクリックして変更を破棄します。

セッション変数 CLI コマンドの概要

次の `udb` コマンドでセッション変数を管理します。

- `udb.sv.add` – 新しいセッション変数の追加
- `udb.sv.del` – セッション変数の削除
- `udb.sv.fl` – ユーザーのすべてのセッション変数のフラッシュ
- `udb.sv.la` – 全セッション変数の表示
- `udb.sv.lo` – セッション変数の詳細情報の取得
- `udb.sv.mod` – セッション変数の変更
- `udb.sv.re` – ユーザーの全セッション変数の再暗号化

プロビジョニングシステムの CLI コマンドについての詳細は、『Sun N1 Service Provisioning System 5.1 コマンド行インタフェース(CLI) リファレンスマニュアル』を参照してください。

第 6 章

構成の生成

Sun N1 Service Provisioning System ソフトウェアでは、*substitution variables*および*variable settings*を使用することによって、あらゆるコンポーネントの配備をカスタマイズできます。

この章の内容は次のとおりです。

- 55 ページの「構成の生成の概要」
- 58 ページの「置換に使用できる変数の種類」
- 71 ページの「置換変数の使用法」

構成の生成の概要

アプリケーションをコンポーネントとしてモデル化し、さまざまな環境で動作するようにカスタマイズできます。

たとえば、データベースアプリケーションを使用してオンラインカタログ販売を管理するとします。運用環境にアプリケーションを配備する前に、テスト環境でアプリケーションをテストします。この 2 つの環境で同じアプリケーションを実行しますが、構成は異なります。テスト環境にアプリケーションを配備する場合は、テスト用のデータベースを使用します。運用環境にアプリケーションを配備する場合は、運用版のデータベースを使用します。

使用するデータベースなど、アプリケーションに関する構成情報は、構成ファイルに格納できます。各環境をサポートするには、これらの構成ファイルをカスタマイズする必要があります。構成ファイルには、コンポーネントの配備時に変数設定値に置き換えられる置換変数を含めることができます。

置換変数は構成ファイルで使用できるだけではなく、プランとコンポーネントでも使用できます。たとえば、アプリケーションをインストールするディレクトリを置換変数で指定できます。プロビジョニングシステムでは、配備環境ごとに個別の変数設定を定義して管理できます。

構成ファイル、プラン、またはコンポーネント中の置換変数参照により、次の種類の値を取得できます。

- ユーザー定義の値
- 名前、ラベルなど、コンポーネント固有の値
- IP アドレスなど、ターゲットホスト固有の値
- ユーザー定義の、プロビジョニングシステムのホスト固有の値
- セッション変数の値
- インストール済みコンポーネントに関連付けられた値

プロビジョニングシステムでは、*configuration generation engine*を使用して、置換変数参照を適切な変数設定値に置き換えます。このエンジンは、ホストリポジトリおよびコンポーネントリポジトリと対話し、プランを実行するたびに値を解決してコンポーネントを配備します。

プロビジョニングシステムには変数設定を格納しておくためのリポジトリがあるので、あとで値を再利用できます。コンポーネントのインストールとプランの実行に使用された変数設定を表示すると、変数の状態を調べられます。インストール済みコンポーネントとの比較実行時に置換変数の状態を調べる場合にも、このリポジトリを使用します。

コンポーネントへの置換変数の追加

複数のホストに同じアプリケーションを配備するプランを実行する場合は、構成の生成を使用することによって、置換変数を各ホストに適した値に自動的に置き換えることができます。

その場合、コンポーネントに置換変数定義を追加します。置換変数定義はたとえば、アプリケーションのインストール先ディレクトリを設定する 1 つの手段として使用できます。プロビジョニングシステムを使用することにより、次のように、各ターゲットホスト上で、アプリケーション配備に関してそれぞれ異なる変数設定を定義して管理できます。

- コンポーネントのバージョンごとに、独自の変数定義を宣言できます。
- コンポーネントのバージョンごとに、独自の変数設定 (前のバージョンからインポートされることもある) があります。
- 任意の変数設定を使用して各コンポーネントをインストールできます。

置換変数の値

アプリケーションのインストール時にプロビジョニングシステムが置換する値は、次のいずれかになります。

- ユーザー指定の値
- 名前、ラベルなど、コンポーネント固有の値
- インストール済みのコンポーネントに関連付けられた値

- セッション変数の値
- IP アドレス、ユーザー定義のホスト変数など、ターゲットホスト固有の値

生成コンテキスト

変数が置換されるのは、ターゲットホスト上でターゲットステップが実行されたときです。ステップはターゲットホストにインストールする、またはインストール済みのプランまたはホストに組み込めます。ターゲットホストとコンポーネントに関連付けられた状態がある場合は、その状態に基づいて個々の置換変数値が決定されます。状態に含まれる可能性がある情報は、次のとおりです。

- ターゲットコンポーネント – 処理の実行対象となるコンポーネント
- ターゲットホスト – ターゲットステップが実行されている現在のホスト
- ターゲット変数設定値のソース – コンポーネントのインストール時に定義されていたデフォルト値を書き換える、名前と値の組み合わせのコレクション
- ローカル変数 – 囲い込みブロックまたはプランで宣言した変数など、ターゲットステップそのものに含まれるあらゆる変数

入力ソース

変数置換エンジンは、`String` 形式または `Reader` 形式であれば、あらゆるテキストベースの入力ソースを処理します。ただし、実際には入力ソースとして使用されるのは、次のエンティティーだけです。

- 構成タイプのリソースファイル
- CLI コマンドに対する入力 (オンデマンド方式で変数置換を実行する)
- コンポーネント属性とプラン属性の一部 (`<installSteps>` 要素の *defaultInstallPath* 属性、`<execNative>` ステップの *command* 属性など)

  『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の第 2 章「コンポーネントと単純プランにより使用される共有スキーマ」、『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の第 3 章「コンポーネントのスキーマ」、および『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の第 4 章「プランのスキーマ」を参照してください。

置換変数の検証

コンポーネント、プラン、または構成テンプレートのチェックイン時に、置換変数は構文エラーがないか解析および調査されます。

プロビジョニングシステムは、状況に依存した構文検証を行います。たとえば、式 `:[target:sys.hostName]` はコンポーネント変数のデフォルト値では使用できますが、ステップでは使用できません。

プロビジョニングシステムは、置換変数により参照されるエンティティーの意味は検証しません。たとえば、変数がシステムサービスを参照する場合、プロビジョニングシステムはそのシステムサービスが実際に存在するかは確認しません。

置換に使用できる変数の種類

コンポーネント、プラン、変数設定値、および構成可能なリソース内で、次の種類の変数を参照できます。

- 単純置換変数
- 外部コンポーネント置換変数
- セッション置換変数
- ターゲット置換変数

単純置換変数参照

入力ソースには、任意の数の単純置換変数参照を含めることができます。単純置換変数の構文は、次のとおりです。

:[*variable*]

variable は次のいずれかです。

- ローカル変数名**:** ローカル変数名は、プランのステップ、およびコンポーネントのインストール、アンインストール、スナップショット、および制御サービスブロックに組み込まれたステップで参照できます。
- パラメータ名**:** パラメータはプランのステップ、およびコンポーネントのインストール、アンインストール、スナップショット、および制御サービスブロックに組み込まれたステップで参照できます。
- 事前定義されたコンポーネント変数名**:** 事前定義されたコンポーネント変数名を使用できるのは、コンポーネントに関連付けられていて、プランには関連付けられていない入力ソース内に限られます。

  事前定義されたコンポーネント変数名は、変数参照がベースコンポーネントで行われる場合でも、常に、インストールされている実際のコンポーネントに関連付けられて解決されます。アクセシビリティの観点から、事前定義されたコンポーネント変数は、 `PUBLIC` 変数として扱われます。

  名前の一覧については、59 ページの「事前定義されたコンポーネント変数名」を参照してください。
- 動的コンポーネント変数名**:** コンポーネントはローカルで宣言された変数と継承された変数の両方を参照できます。参照のために置換される値は、関連する変数設定ソースの優先指定値または変数のデフォルト値のどちらかです。使用される値は、置換の前に展開されます。

動的コンポーネント変数名は、ターゲットコンポーネントに関連付けられるコンポーネントが宣言または継承します。この場合、変数参照で置換する値は、関連する変数設定の優先指定値またはコンポーネントによって定義されたデフォルト値のどちらかです。この変数の値は、置換の前に展開されます。

事前定義されたコンポーネント変数名

事前定義されたコンポーネント変数名は、次のように、生成コンテキストの関連するコンポーネント属性を参照します。

- `sys.name` – コンポーネント名
- `sys.version` – 標準形式 *major*. *minor* のコンポーネントのバージョン番号
- `sys.description` – コンポーネントの説明
- `sys.label` – コンポーネントのラベル
- `sys.softwareVendor` – コンポーネントでモデル化したアプリケーションの作成元であるソフトウェアベンダー
- `sys.author` – コンポーネントの作成者
- `sys.path` – コンポーネントのパス

  `sys.path` 変数は、コンポーネントの *path* 属性に類似していますが、後続スラッシュ (/) が含まれます。`sys.path` を `sys.name` 変数と直接結合することによって、コンポーネントの完全な名称を作成できます。

  ```
  :[sys.path]:[sys.name]
  ```

- `sys.rsrcInstallPath` – コンポーネントリソースのインストール先

  `sys.rsrcInstallPath` 変数は、コンポーネントをインストールする、またはコンポーネントがインストールされている *remote agent* のファイル形式になります。この変数を使用できるのは、単純コンポーネントのみです。この変数の値は、ほかの動的コンポーネント変数の計算を必要とする場合があるため、コンポーネント変数のデフォルト値または変数設定の優先指定値に `sys.rsrcInstallPath` を使用することはできません。また、この変数は、`<component>` *installPath* 属性の値として、または `<resourceRef>/<installSpec>` 要素の属性内に使用することもできません。これらの制約によって、参照が循環する可能性が排除されます。

- `sys.targetRefName` – 関連付けられたコンポーネントのターゲットホスト名

  `sys.targetRefName` 変数は、*targetable components*にのみ定義されます。コンポーネントのターゲットホストは、ターゲット作成可能コンポーネントの `<targetRef>` 要素の *hostName* 属性の解決された値によって定義されます。コンポーネント変数が `sys.targetRefName` 値の計算に使用される可能性があるので、コンポーネントのターゲットホストをコンポーネント変数値で単純置換変数として使用することはできません。この変数は通常、`<retarget>` ステップと組み合わせて使用し、ターゲット作成可能コンポーネントにコンポーネントをインストールしたり、またはターゲット作成可能コンポーネントに含まれているコンポーネントを操作したりします。

事前定義されたコンポーネント変数は、変数参照がベースコンポーネントで行われる場合でも、常に、インストールされている実際の コンポーネントに関連付けられて解決されます。事前定義されたコンポーネント変数は、PUBLIC 変数として扱われます。

置換変数のエスケープシーケンス

入力ソースに :[を含める場合は、:[[を使用してエスケープする必要があります。:[[を検出するたびに、構成生成機能が :[に置き換えます。

たとえば、プロビジョニングシステムに *box* の値を置換させるのではなく、:[box] を字句どおりに含める場合は、:[[box] を使用します。

入力ソースが XML で作成されたコンポーネントの属性である場合、値は XML ベースのエスケープシーケンスに関する次の規則が適用されます。

- 引用符 ("") を得るには、" シーケンスを使用します。
- アポストロフィー (') を得るには、' シーケンスを使用します。
- 小記号 (<) を得るには、< シーケンスを使用します。
- 大記号 (>) を得るには、> シーケンスを使用します。
- アンパサンド (&) を得るには、& シーケンスを使用します。

これらのエスケープ規則は、プロビジョニングシステムのブラウザインタフェース、構成リソースファイル、または構成生成 CLI コマンドの入力による優先指定値には適用されません。

置換変数の展開

置換変数参照が変数設定値によって置き換えられるときには、値がそのつど展開されてから置換されます。この展開が必要なのは、値そのものに単純置換変数の参照が含まれている場合があるからです。値の置換を防ぐには、入力ソースを :[[でエスケープします。

デフォルト置換値または優先置換値、パラメータのデフォルト値、およびローカル変数のデフォルト値で使用できるのは、ホストおよび外部コンポーネント置換参照だけです。このような参照は、その他の入力ソースでは認められません。

単純置換変数参照の展開

変数設定値における単純置換変数参照の構文は、入力ソースの場合と同じです。

:[*varname*]

この場合、参照する変数名が、参照が含まれる変数より先に宣言されていなければなりません。この制約によって、参照の循環が防止されます。

派生コンポーネントの場合、ベースコンポーネントから継承した値が、親での宣言順に従って先に展開されます。その後、ローカルの非優先変数が宣言された順に展開されます。継承した値を無効にするローカル変数は、継承値の展開順序と同じ順で継承

値の代わりに展開されます。したがって、優先変数が参照できるのは、すでに宣言されているほかの継承変数または優先変数だけです。たとえば、ベースコンポーネント A が *x* と *y* という変数を宣言するとします。さらに、派生コンポーネント B が変数 *z* と *y* を宣言します。コンポーネント B で変数が評価される順番は、コンポーネント A から継承した *x*、続いてコンポーネント B によって上書きされた *y*、最後にコンポーネント B のローカル変数である *z* です。

例 **6–1** 単純置換変数の使用方法

次の表に、置換変数、対応する展開後の値、未展開の値の例を示します。

変数名	未展開の値	展開後の値
`foo`	`silly`	`silly`
`bar`	`:[foo]`	`silly`
`baz`	`a :[foo] :[bar] example`	`a silly silly example`
`badFrob`	`:[frob]`	エラー – 順参照
`frob`	`:[[foo]`	`:[foo]`
`compName`	`:[sys.name]`	ターゲットコンポーネント名
`badFoz`	`:[foz]`	エラー – `foz` が未宣言

外部コンポーネント置換変数参照

変数設定値には、1 つ以上の外部コンポーネント置換参照を含めることができます。外部コンポーネント置換参照は、次のいずれか 1 つです。

- `:[`*primary-component* `:`*varname*`]`
- `:[`*primary-component* `:`*secondary-component-list*`:` *varname*`]`

外部コンポーネント置換参照には空白が含まれないことに注意してください。

primary-component は次のいずれか 1 つです。

- *explicit-external-component*
- *system-service-component*
- *system-type-component*
- *targetable-component*
- *secondary-component*

secondary-component は次のいずれか 1 つです。

- *nested-component*
- *toplevel-component*
- *dependee-component*
- *container-component*

secondary-component-list は、コロンで区切った 1 つ以上の *secondary-components* の列です。

varname は、事前定義の置換変数名または参照先コンポーネントによって宣言される動的置換変数名のいずれかです。

明示的な外部コンポーネントの展開

explicit-external-component は、特定の場所の特定のホストにインストールされる予定のコンポーネントを明示的に指定します。インストール型コンポーネントターゲッタ `<installedComponent>` のコンポーネント解決セマンティックスが含まれます。『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の「インストール済みコンポーネントターゲッター」を参照してください。

explicit-external-component の構文は次のとおりです。

component-target : *explicit-component-reference*

component-target では、特定のコンポーネントをインストールするホストを指定します。次のいずれか 1 つです。

- `component` – 生成コンテキストに含まれている現在のターゲットホストを指定します。
- `component` (*host-redirect*) – *host-redirect* によって名前が指定されたホストを参照します (69 ページの「ホストリダイレクト」を参照)。

explicit-component-reference は、*component-target* で指定したターゲットホストにすでにインストールされているコンポーネントに対する参照です。*explicit-component-reference* では最小限、インストールするコンポーネントの名前を含めますが、次のように、バージョンとインストールパスも指定できます。

full-component-name
full-component-name#*version*
full-component-name@{*path*}
full-component-name#*version*@{*path*}

full-component-name は、指定したコンポーネントに対する絶対参照または相対参照です。相対参照は、*explicit-component-reference* が含まれているプランまたはコンポーネントとの関係に基づいて展開されます。

path は、指定したコンポーネントの絶対インストールパス名、または拡張時に指定したコンポーネントの絶対インストールパスを表す置換変数参照です。*path* に } 記号が含まれる場合は、}} シーケンスを使用してエスケープする必要があります。

システムサービスコンポーネントの展開

system-service-component は、現在のターゲットのルート*physical host*にインストールされる予定のシステムサービスコンポーネントを指定します。インストール型コンポーネントターゲッタ `<systemService>` のコンポーネント解決セマンティックスが含まれます。『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の「インストール済みコンポーネントターゲッター」を参照してください。

system-service-component の構文は、次のとおりです。

`systemService:`*system-service-name*

システムサービスがプラグインによって定義される場合、定義するプラグインの名前を *system-service-name* の接頭辞にする必要があります。

システムタイプコンポーネントの展開

system-type-component は、特定の場所の特定のホストに最後にインストールした、特定のコンポーネントタイプから派生したコンポーネントを指定します。インストール型コンポーネントターゲッタ `<systemType>` のコンポーネント解決セマンティックスが含まれます。『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の「インストール済みコンポーネントターゲッター」を参照してください。

system-type-component の構文は、次のとおりです。

`systemType:`*component-type-name*
`systemType:`*component-type-name*`@{`*path*`}`
`systemType(`*host-redirect*`):`*component-type-name*
`systemType(`*host-redirect*`):`*component-type-name*`@{`*path*`}`

path は省略可能です。*path* を指定する場合は、指定したコンポーネントの絶対インストールパス名または置換変数参照です。展開された参照は、指定したコンポーネントの絶対インストールパスを表します。*path* に } 記号が含まれる場合は、}} シーケンスを使用してエスケープする必要があります。

host-redirect も省略可能です。コンポーネントのインストール先ホストを指定します。69 ページの「ホストリダイレクト」を参照してください。

コンポーネントタイプがプラグインによって定義される場合、定義するプラグインの名前を *component-type-name* の接頭辞にする必要があります。

`targetableComponent` コンポーネントの展開

targetable-component は、ターゲット作成可能コンポーネントに関連付けられる変数を参照し、`<targetableComponent>` ターゲッタと類似しています。『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の「インストール済みコンポーネントターゲッター」を参照してください。

targetable-component の構文は、次のとおりです。

- `targetableComponent`
- `targetableComponent(`*host-redirect* `)`

host-redirect はコンポーネントのターゲットホストの名前です。指定しなかった場合、現在プランが実行されているホストが使用されます。これらの構文は、特定のコンポーネントのターゲットホストに関連付けられているターゲット作成可能コンポーネントに解決されます。69 ページの「ホストリダイレクト」を参照してください。

二次コンポーネントの展開

入れ子になったコンポーネントの展開

nested-component は、現在のコンポーネントで宣言された入れ子のコンポーネント参照によって参照されるコンポーネントを指定します。インストール型コンポーネントターゲッタ `<nestedRef>` のコンポーネント解決セマンティックスが含まれます。『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の「インストール済みコンポーネントターゲッター」を参照してください。

nested-component の構文は、次のとおりです。

`nestedRef:`*component-reference-name*

component-reference-name は、現在のコンポーネントで `installMode=NESTED` の場合、`<componentRef>` 要素の名前です。

nested-component を使用できるのは、現在のコンポーネントが複合コンポーネントの場合だけです。また、外部コンポーネント置換参照が複合コンポーネントに含まれる場合、*primary-component* としての使用に限定されます。*primary-component* として使用する場合は、コンポーネント変数のデフォルト値または変数設定の優先指定値として使用することはできません。これは、参照対象のコンポーネントが変数の評価時点ではまだインストールされていないからです。

上位レベルコンポーネントの展開

toplevel-component は、現在のコンポーネントで宣言された上位レベルコンポーネント参照によって参照されるコンポーネントを指定します。インストール型コンポーネントターゲッタ `<toplevelRef>` のコンポーネント解決セマンティックスが含まれます。『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の「インストール済みコンポーネントターゲッター」を参照してください。

toplevel-component の構文は、次のとおりです。

`toplevelRef:`*component-reference-name*
`toplevelRef:`*component-reference-name*`@{`*path*`}`
`toplevelRef(`*host-redirect*`):`*component-reference-name*
`toplevelRef(`*host-redirect*`):`*component-reference-name*`@{`*path*`}`

component-reference-name は、現在のコンポーネントで `installMode=TOPLEVEL` の場合、`<componentRef>` 要素の名前です。

path は、指定したコンポーネントの絶対インストールパス名または置換変数参照です。展開された参照は、指定したコンポーネントの絶対インストールパスを表します。*path* に } 記号が含まれる場合は、}} シーケンスを使用してエスケープする必要があります。

host-redirect は、参照対象のコンポーネントがインストールされる予定のホストです。*host-redirect* の構文については、69 ページの「ホストリダイレクト」を参照してください。

toplevel-component を使用できるのは、現在のコンポーネントが複合コンポーネントの場合だけです。また、外部コンポーネント置換参照が複合コンポーネントに含まれる場合、*primary-component* としての使用に限定されます。*primary-component* として使用する場合は、コンポーネント変数のデフォルト値または変数設定の優先指定値として使用することはできません。これは、参照対象のコンポーネントがすでにインストールされているからです。

依存先コンポーネントの展開

dependee-component は、<createDependency> ステップで生まれた依存関係の結果として、現在のコンポーネントが依存するコンポーネントを指定します。*dependee-component* には、インストール型コンポーネントターゲッタ <dependee> のコンポーネント解決セマンティックスが含まれます。『Sun N1 Service Provisioning System 5.1 XML スキーマリファレンスガイド』の「インストール済みコンポーネントターゲッター」を参照してください。

dependee-component の構文は、次のとおりです。

```
dependee:dependency-name
```

dependency-name は、現在のコンポーネントによって作成された依存関係の名前です。

コンポーネントに外部コンポーネント置換参照が含まれる場合、*dependee-component* は *primary-component* としての使用に限定されます。*primary-component* として使用する場合は、コンポーネント変数のデフォルト値または変数設定の優先指定値として使用することはできません。これは、変数の評価時点では、依存関係がまだ埋まれていないからです。

コンテナコンポーネントの展開

container-component は、現在のコンポーネントが入れ子の参照として含まれるコンポーネントを指定します。

container-component の構文は、次のとおりです。

```
container
```

別の「コンテナ」コンポーネントにより、入れ子コンポーネントとしてすでにインストールされているコンポーネントに、外部コンポーネント置換参照が含まれる場合、*container-component* は、*primary-component* としての使用に限定されます。

外部コンポーネント置換参照の解決

外部コンポーネント置換参照の値は、最初に、*primary-component* が参照するコンポーネントを解決することで計算されます。*secondary-component* を *primary-component* として使用している場合、外部コンポーネント置換参照を含むコンポーネントは、*primary-component* を解決するときに使用される最初の現在のコンポーネントとして機

能します。さらに、*primary-component* によって解決されたコンポーネントが、新しい現在のコンポーネントになります。次に、*secondary-component-list* に含まれている各 *secondary-component* が、最後に解決されたコンポーネントを現在のコンポーネントとして使用することで解決されます。最後に、プロビジョニングシステムは最後に解決されたコンポーネントで変数 *varname* を検索して返します。

外部コンポーネント置換参照の値は、コンポーネントのインストール時に参照先コンポーネントに指定されていた変数の値に基づいて計算されます。インストール済みのコンポーネントで宣言されていない変数、またはターゲットホストにインストールされていないコンポーネントは、参照することはできません。また、外部コンポーネント置換参照を宣言したコンポーネントまたはプランがアクセスできない変数を参照することもできません。変数にアクセスできるのは、その変数がアクセス可能なアクセスモードで宣言され、宣言するコンポーネントがアクセス可能で、なおかつそのコンポーネントに到達するまでにたどる主コンポーネントと二次コンポーネントのそれぞれがアクセス可能な場合に限られます。

外部コンポーネント置換参照は、プランの実行開始時ではなく、実行中に検出された時点で計算されます。したがって、プランでそれまでに実行されたステップに応じて、参照対象である、実際にインストールされたコンポーネントが変化する可能性があります。さらに、現在のターゲットホスト以外のホストにインストールされたコンポーネントを参照する場合は、他方のホストで同時に実行されている他のプランによって影響を受ける可能性があります。予測可能な結果を保証するには、現在のプラン実行のターゲットセットに含まれているホストに参照を限定してください。これらのホストは確実にロックされているからです。さらに、再ターゲットまたはシリーズの実行モードを使用することによって、プラン内の外部参照を受けるホストでの `<install>` または `<uninstall>` 動作の同期を取ります。

例 **6–2** 外部コンポーネント置換変数参照の使用

- 次に、明示的な外部コンポーネントを使用する、外部コンポーネント置換変数参照の例を示します。

```
:[component:installApache:sys.label]
:[component:jdk#1.3:classpath]
:[component:webApp#2.4@{/usr/local}:bannerColor]
```

- 次の例で、プロビジョニングシステムは、現在のターゲットホストのルート物理ホストにインストールされたコンポーネントを解決します。

```
:[component(/):IIS global settings:install_path]
```

- 次の例では、`webAppPath` は、生成コンテキストの関連コンポーネントに設定された、単純置換変数の名前に解決されます。外部参照対象のコンポーネントで定義されている変数ではありません。

```
:[component:webApp@{:[webAppPath]}:bannerColor]
```

- 次に、*targetableComponent* を使用する外部コンポーネント置換変数参照の例を示します。

```
:[targetableComponent:hostName]
:[targetableComponent(host1):port]
```

- 次に、*nested-component* と *toplevel-component* を使用する外部コンポーネント置換変数参照の例を示します。

例 **6–2** 外部コンポーネント置換変数参照の使用　　*(続き)*

```
:[nestedRef:ref1:sys.label]
:[toplevelRef:ref2:sys.name]
:[toplevelRef(/):ref3@{/usr/local}:var1]
```

- 次に、*system-type-component* と *system-service-component* を使用する外部コンポーネント置換変数参照の例を示します。

```
:[systemService:com.sun.windows#Windows SS:classPath]
:[systemType:com.sun.weblogic#WL Target Type:serverName]
:[systemType(..):MyType@{/tmp}:var1]
```

- 次に、*dependee-component* を使用する外部コンポーネント置換変数参照の例を示します。

```
:[dependee:app2domain:domainName]
```

- 次に、*container-component* を使用する外部コンポーネント置換変数参照の例を示します。

```
:[container:sys.installPath]
```

- 次に、連鎖型コンポーネント参照の例を示します。この参照では、現在のターゲットコンポーネントが別のコンテナコンポーネントの入れ子コンポーネントとしてインストールされているものとします。`container:` 部分は現在のコンポーネントが含まれるコンポーネントに解決されます。この例では、X になります。`nestedRef:ref1` 部分は、`ref1` というコンポーネント参照を使用し、コンポーネント X によって参照されるコンポーネントに解決されます。それがコンポーネント Y です。`toplevelRef:ref1` 部分は、`ref1` というコンポーネント参照を使用し、コンポーネント Y が参照するコンポーネントに解決されます。これはコンポーネント Z です。最後に、プロビジョニングシステム がコンポーネント Z によって定義された *label* 属性の値を解決して使用します。

```
:[container:nestedRef:ref1:toplevelRef:ref1:sys.label]
```

セッション置換変数の参照

*Session variables*を使用すると、ユーザーセッションごとに 1 回ずつ、WebLogic 資格などのデータを入力できます。 これらの変数の内容は、セキュリティーを保護して保存することもできるので、ログインのたびにデータを再入力しなくてすみます。
セッション変数は、置換変数を使用することによって、モデル化レベルで組み込まれます。

セッション置換変数参照の構文は、次のとおりです。

`:[session:`*varname*`]`

varname はセッション変数名です。

セッション置換変数参照は、現在のユーザーセッションのセッション変数値に解決されます。現在のユーザーセッションで定義されていないセッション変数を解決しようとすると、エラーが発生します。セッション変数生成エラーが検出されると、ユーザーの現在のセッション変数セットに、値が空のセッション変数を追加することが試行されます。

セッション変数は、単純置換変数参照を使用できるところであれば、どこでも使用できます。しかし、別のセッション変数の値にセッション変数を使用することはできません。

事前定義のセッション変数

`sys:sessionID` は、現在のユーザーセッションの ID に解決される、事前定義セッション変数です。システム `sessionID` 変数を使用すると、プランを実行したユーザーと同じ資格で、CLI を介して *master server* にコールバックするステップを作成できます。 システム `sessionID` 変数は、`config.allowSessionIDOnHosts` 構成変数に基づき、特定のホストのコンテキストに限って解決できます。したがって、システム `sessionID` 変数を解決できるのは、現在の生成コンテキストのターゲットホストが `config.allowSessionIDOnHosts` 構成変数によって指定されたホストに含まれている場合だけです。

プランの実行時にコールバックで使用できるように CLI を設定する場合は、同じサーバーにリモートエージェントと CLI Client をインストールする必要があります。最大限のセキュリティー保護を実現するには、マスターサーバーと同じサーバー上に CLI Client とリモートエージェントをインストールしてください。この場合、CLI を介してコールバックを行うプランおよびコンポーネントは、`<retarget>` ステップを使用して CLI が含まれている 1 つのホストに実行をリダイレクトします。さらに、`<retarget>` ステップのあとでシステム `sessionID` 変数が解決され、構成変数 `config.allowSessionIDOnHosts` は、マスターサーバーホストだけを指定するようになります。

ターゲット置換変数の参照

ターゲット置換変数を使用すると、特定のホストから値を直接取得できます。

ターゲット置換変数参照の構文は、次のとおりです。

`:[target:`*varname*`]`
`:[target(`*host-redirect*`):`*varname*`]`

`target` から始まる変数名は、プランが現在実行されている論理ホストを表します。

varname は、属性値を表します。この属性値は、インストールするアプリケーション用のターゲットホストの定義で指定されています。参照できるホスト変数は、事前定義ホスト変数と動的ホスト変数の 2 種類です。70 ページの「事前定義ホスト変数名」と71 ページの「動的ホスト変数名」を参照してください。

host-redirect も指定した場合、*varname* の値は特定のホストから取得されます。そのホストは、現在プランを実行しているホストでなくてもかまいません。*hostname* に別の置換変数を指定することもできます。この場合は展開時に、ホスト名に解決されます。*host-redirect* の構文については、69 ページの「ホストリダイレクト」を参照してください。

事前定義ホスト置換の短縮形

ホスト置換参照を使用できるところであれば、どこでも事前定義ホスト置換の短縮形を使用できます。

- `:[/]` – 現在のターゲットホストのルート物理ホストに基づいて、オペレーティングシステム固有のファイル区切り文字に展開される
  - UNIX システム – `:[/]` は / に展開される
  - Microsoft Windows システム – `:[/]` は \ に展開される
- `:[:]` – 現在のターゲットホストのルート物理ホストに基づいて、オペレーティングシステム固有のパス区切り文字に展開される
  - UNIX システム – `:[:]` は : に展開される
  - Microsoft Windows システム – `:[:]` は ; に展開される

ホストリダイレクト

ターゲットホスト置換変数の参照と外部コンポーネント置換変数の参照には、ホストリダイレクトを任意で組み込むことができます。この場合、変数またはコンポーネント検索が行われるホストは、現在のターゲットではなく、指定されたホストになります。*host-redirect* の形式は、次のいずれかです。

- *parent-ref*
- *hostname*
- *hostname*/ *parent-ref*

hostname は、ホスト名、または展開時にホスト名に解決される置換変数参照のどちらかです。

parent-ref は次のいずれか 1 つです。

- / – 指定されたホストのルート親ホストです。*hostname* を省略した場合は、現在のターゲットホストになります。
- スラッシュで区切られた **1** つ以上の `..` の連続 – 指定されたホストの *n* 番目の親、または*hostname* を省略した場合は現在のターゲットホストの *n* 番目の親です。*n* は `..` の回数です。

parent-ref 演算子は主に、*virtual hosts*に使用しますが、物理ホストにも使用できます。物理ホストに両方の演算子を適用すると、無操作になります。

例 **6–3** ターゲット置換変数参照の使用

- 次に、ホスト `myHost` 上で変数 `var1` を検索する例を示します。

```
:[target(myHost):var1]
```

例 **6–3** ターゲット置換変数参照の使用　*(続き)*

- 次に、置換変数 hostName で指定されたホスト上で変数 var1 を検索する例を示します。

```
:[target(:[hostName]):var1]
```

- 次に、プランを現在実行している論理ホスト上で変数 var1 を検索する例を示します。

```
:[target:var1]
```

事前定義ホスト変数名

事前定義ホスト変数名は、参照されるホストに関連付けられる属性を示します。

- sys.hostName – ターゲットホスト名
- sys.description – ターゲットホストの説明
- sys.hostType – ターゲットホストのホストタイプ
- sys.ipAddress – ターゲットホスト上のリモートエージェントの IP アドレス
- sys.portNumber – ターゲットホスト上のリモートエージェントのポート番号
- sys.raHomeDir – ターゲットホスト上のリモートエージェントのホームディレクトリの絶対パス
- sys.raDataDir – ターゲットホスト上のリモートエージェントのデータディレクトリの絶対パス
- sys.raTmpDir – ターゲットホスト上のリモートエージェントの一時ディレクトリの絶対パス
- sys.raConfigDir – ターゲットホスト上のリモートエージェントの構成ディレクトリの絶対パス
- sys.OSArch – sparc など、ターゲットホストのアーキテクチャー
- sys.OSName – ターゲットホスト上で動作するオペレーティングシステムの名前
- sys.OSVersion – ターゲットホスト上で動作するオペレーティングシステムのバージョン

例 **6–4** ホスト置換変数参照の使用法

次に、ホスト置換変数参照と対応する値の例を示します。

- :[target:sys.ipAddress] – 現在のホストの IP アドレス
- :[target:var1] – 現在のホストの動的変数 var1
- :[target(host1):var1] – ホスト host1 の動的変数 var1
- :[target(:[hostNameVar]):var1] – 名前が置換参照 hostNameVar の値であるホストの動的変数 var1
- :[target(/):var1] – 現在のホストのルート親ホストの動的変数 var1

例 **6–4** ホスト置換変数参照の使用法　*(続き)*

- `:[target(host1//):var1]` – `host1` というホストのルート親ホストの動的変数 `var1`
- `:[target(../..):var1]` – 現在のホストの第 2 親ホストの動的変数 `var1`
- `:[target(host1/../..):var1]` – `host1` というホストの第 2 親ホストの動的変数 `var1`

動的ホスト変数名

動的ホスト変数名は、参照されたホストの*host type*によって宣言された、任意の属性名です。この場合、変数参照で置換される値は、参照されたホストで定義されている属性の対応する値です。該当する場合、値を展開してから置換する必要があります。

置換変数の使用法

単純置換変数は、構成ファイルを含めてあらゆる入力ソースで使用できます。外部コンポーネント置換変数とターゲット置換変数は `<installSteps>`、`<uninstallSteps>`、`<controlService>`、および `<executionPlan>` 要素の `<varList>/<var>` のデフォルト値をはじめ、さまざまな場所で使用できます。

`:[` *varname*`]` 形式の単純置換変数参照は、構成ファイルや構成可能コンポーネントの属性をはじめ、あらゆる入力ソースで使用できます。ただし、ホスト置換参照 (`:[target:`*varname*`]`) と外部コンポーネント置換参照 (`:[component:`*compRef*`:`*varname*`]`) を使用できるのは、変数設定値に限定されます。

変数設定値には、`<var>` 要素のデフォルト属性と、変数設定の優先指定値が含まれますが、構成ファイルまたはそのほかの構成可能コンポーネント属性は含まれません。この制限によって、コンポーネントとコンポーネントが参照する構成ファイル内のすべての変数が、コンポーネントの `<varlist>` 要素によって明示的に宣言されて検証されることが保証されます。

次に、特定の入力ソースで使用できる置換変数をすべて示します。記載する要素と属性は入力ソースであり、サブリストには使用できる変数の種類を示します。

- `<component>/<varList>/ <var>/` *default* 属性:

  変数設定の優先指定値:

  - `sys.rsrcInstallPath` および `sys.targetRefName` 以外のコンポーネント置換参照
  - ホスト置換参照
  - 主コンポーネントとしての *nested-component* および *dependee-component* を除く外部コンポーネント置換参照

- セッション変数参照:

- `<component>/<installList>/ <installSteps>/<paramList>/<param>` / *default* 属性:

  `<component> /<uninstallList>/<uninstallSteps>/ <paramList>/<param>`/*default* 属性:

  `<component>/<controlList>/ <control>/<paramList>/<param>`/ *default* 属性:

  `<component>/ <snapshotList>/<snapshot>/<paramList> /<param>`/ *default* 属性:

  - コンポーネント置換参照
  - ホスト置換参照
  - 外部コンポーネント置換参照
  - セッション変数参照:

- `<component>/<installList>/ <installSteps>/<varList>/<var>`/ *default* 属性:

  `<component>/ <uninstallList>/<uninstallSteps>/<varList> /<var>`/ *default* 属性:

  `<component>/<controlList>/ <control>/<varList>/<var>`/ *default* 属性:

  `<component>/ <snapshotList>/<snapshot>/<varlList> /<var>`/*default* 属性:

  - 宣言済みのローカル変数置換参照
  - パラメータ置換参照
  - コンポーネント置換参照
  - ホスト置換参照
  - 外部コンポーネント置換参照
  - セッション変数参照:

- `<component>/<installList>/ <installSteps>`/子ステップ:

  `<component>/ <uninstallList>/<uninstallSteps>`/子ステップ:

  `<component>/<controlList>/ <control>`/子ステップ:

  `<component>/ <snapshotList>/<snapshot>/<prepare>` /子ステップ:

  `<component>/<snapshotList> /<snapshot>/<capture>`/子ステップ:

  `<component>/<snapshotList>/ <snapshot>/<cleanup>`/子ステップ:

  - ローカル変数置換参照
  - パラメータ置換参照
  - コンポーネント置換参照

- `<component>/<resourceRef>/ <installSpec>`:

  `<component>`/ *installPath* 属性:

  - `sys.rsrcInstallPath` 以外のコンポーネント置換参照

- `<component>/<diff>/ <ignore>`:

<component>/<componentRef> /<argList>:

構成ファイル:

- コンポーネント置換参照

- <component>/<targetRef>:

  <component>/<targetRef>/ <agent>:

  - sys.targetRefName および sys.rsrcInstallPath を除くコンポーネント置換参照

- <executionPlan>/<paramList>/ <param>/ *default* 属性:

  - セッション変数参照:

- <executionPlan>/<varList>/ <var>/ *default* 属性:

  <executionPlan>/*/<inlineSubplan>/<varList> /<var>/ *default* 属性:

  - セッション変数参照:
  - 宣言済みのローカル変数置換参照
  - 包含するプランの表示状態のローカル変数置換参照
  - 表示状態のパラメータ置換参照
  - 単純プランおよびサブプラン限定のホスト置換参照
  - 単純プランおよびサブプラン限定の外部コンポーネント置換参照 (主コンポーネントとしてのコンテナ、入れ子、上位レベル、および依存先は除く)

- <executionPlan>/*/子ステップ:

  - ローカル変数置換参照
  - 包含するプランの表示状態のローカル変数置換参照
  - 表示状態のパラメータ置換参照

- hostAttributes

  - セッション変数参照:

次に示すコンポーネントの *configuration* 属性には、置換変数参照を含めることができます。

親要素	置換可能な属性
<capture>/<addFile>	*displayName*、*path*
<component>	*installPath*
<diff>/<ignore>	*path*
<installSpec>	*name*、*path*、*permissions*、*user*、*group*、*deployMode*、*diffDeploy*
<paramList>/<param>	*default*

親要素	置換可能な属性
<targetRef>	*hostName*
<targetRef>/<agent>	*connection*、*ipAddr*、*port*、*params*
<varList>/<var>	*default*

次に示す、コンポーネントまたはプランに指定された *step* 属性には、置換変数参照を含めることができます。

親要素	置換可能な属性
<argList>	<argList> 要素の属性、形式は自由
<execJava>	*className*、 *classPath*
<execNative>	*dir*、*userToRunAs*
<execNative>/<ouputFile>	*name*
<execNative>/<errorFile>	*name*
<execNative>/<env>	*name*、*value*
<execNative>/<exec>	*cmd*
<execNative>/<exec>/<arg>	*value*
<execNative>/<shell>	*cmd*、<body>
<execNative>/<successCriteria>	*outputMatches*、 *errorMatches*
<execNative>/<inputFile>	*name*
<execNative>/<inputText>	<body>
<if>/<condition>/<equals>	*value1*、*value2*
<if>/<condition>/<istrue>	*value*
<if>/<condition>/<matches>	*value*、*pattern*
<processTest>	*processNamePattern*、 *user*
<raise>	*message*
<retarget>	*host*
<retarget>/<varList>/<var>	*default*
<sendCustomEvent>	*message*
<transform>	*input*、*output*
<transform>/<source>	*name*

親要素	置換可能な属性
<transform>/<stylesheet>	<body>
<transform>/<subst>	*match*、*replace*
<urlTest>	*url*、*pattern*

次のプラン属性には、置換変数参照を含めることができます。

親要素	置換可能な属性
<paramList>/<param>	*default*
<varList>/<var>	*default*

インストール型コンポーネントターゲッタの次の属性には、置換変数参照を含めることができます。

親要素	置換可能な属性
<installedComponent>	*installPath*、*host*
<systemType>	*installPath*、*host*
<targetableComponent>	*host*
<toplevelRef>	*installPath*、*host*

リポジトリコンポーネントターゲッタの次の属性には、置換変数参照を含めることができます。

親要素	置換可能な属性
<component>	*host*
<toplevelRef>	*host*

例 **6–5** 置換変数の使用法

次に、Apache Web サーバー用のコンポーネントで定義されている変数の例を示します。

```
<varList>
    <var name="domainname" default=":[target:domainname]"/>
    <var name="name" default="apache"/>
    <var name="installPath" default="/opt/apache"/>
    <var name="execNativeShutdown"
       default=":[installPath]/bin/apachectlstop"/>
    <var name="execNativeStartUp"
```

例 **6–5** 置換変数の使用法　(続き)

```
        default=":[installPath]/bin/apachectlstart"/>
</varList>
```

このコンポーネントの指定内容は、次のとおりです。

- ドメイン名はターゲットホストから動的に取得する必要があります。
- *name* のデフォルト値は apache です。
- installPath という名前の変数には /opt/apache という値が設定されています。
- installPath の値は、2 つの <execNative> 変数の定義で使用されています。

構成ファイルでターゲット置換変数を使用することはできませんが、構成ファイルからホスト固有の値を参照できます。これが可能なのは、参照する変数の値がホスト固有値として計算可能だからです。たとえば、コンポーネントの <varList> セクションに、次の変数を定義しておくことができます。

```
<var name="box" value=":[target:room]">
```

この場合、構成ファイルから :[box] を参照できます。:[box] を置換するときには、ターゲットホストによって定義された room 変数の値に置き換えられます。

付録 A

変数置換の文法

ホスト属性値に指定できるのは、*session-variable-substitution-reference* だけです。次の文法では、構成リソースファイルと特定のコンポーネント要素属性で *simple-substitution-variable-reference* を使用できます。*complex-substitution-variable-reference* は、置換変数のデフォルト値と優先指定値で使用できます。

この文法では、`\p{N}` はすべての Unicode 数字を、`\p{L}` はすべての Unicode 文字を表します。

simple-substitution-variable-reference:
    local-component-substitution-reference

complex-substitution-variable-reference:
    simple-substitution-variable-reference
    host-substitution-reference
    external-component-substitution-reference
    session-variable-substitution-reference

local-component-substitution-reference:
    : [*component-variable-name*]

component-variable-name:
    predefined-component-variable-name
    dynamic-component-variable-name

predefined-component-variable-name:
    `sys.name`
    `sys.version`
    `sys.description`
    `sys.label`
    `sys.softwareVendor`
    `sys.author`
    `sys.path`
    `sys.rsrcInstallPath`
    `sys.targetRefName`

dynamic-component-variable-name:
    identifier

host-substitution-reference:
:[target:*host-variable-name*]
:[target(*host-reference*):*host-variable-name*]
:[:]
:[/]

host-reference:
root-host-selector
parent-host-selector
host-selector / *root-host-selector*
host-selector / *parent-host-selector*

host-selector:
host-name
complex-substitution-variable-reference

root-host-selector:
/

parent-host-selector:
..(/..)*

host-variable-name:
predefined-host-variable-name
dynamic-host-variable-name

predefined-host-variable-name:
sys.hostName
sys.description
sys.hostType
sys.ipAddress
sys.portNumber
sys.raHomeDir
sys.raDataDir
sys.raConfigDir
sys.raTmpDir
sys.OSName
sys.OSVersion
sys.OSArch

dynamic-host-variable-name:
identifier

session-variable-substitution-reference:
:[session:*session-variable-name*]

session-variable-name:
predefined-session-variable-name
dynamic-session-variable-name

predefined-session-variable-name:
sys:sessionID

external-component-substitution-reference:

: [*component-reference-list*:*component-variable-name*]

component-reference-list :
primary-component-reference
primary-component-reference : *secondary-component-reference-list*

primary-component-reference :
explicit-external-component-reference
system-service-component-reference
system-type-component-reference
secondary-component-reference
targetable-component-reference

secondary-component-reference :
nested-component-reference
toplevel-component-reference
dependee-component-reference
container-component-reference

secondary-component-reference-list :
secondary-component
secondary-component : *secondary-component-reference-list*

explicit-external-component-reference :
`component` : *explicit-component-reference*
`component` (*host-reference*) : *explicit-component-reference*

explicit-component-reference :
full-component-name
full-component-name#*component-version*
full-component-name@{*install-path*}
full-component-name#*component-version*@{*install-path*}

full-component-name :
path-reference component-name
component-name

path-reference :
/
/ *relative-path-reference* /
relative-path-reference /

relative-path-reference :
.
..
path-part
. / *relative-path-reference*
.. / *relative-path-reference*
path-part / *relative-path-reference*

system-service-component-reference :
`systemService` : *system-name*

system-type-component-reference :
`systemType` : *system-name*

```
        systemType:system-name@{install-path}
        systemType(host-reference):system-name
        systemType(host-reference):system-name@{install-path}

targetable-component-reference:
        targetableComponent
        targetableComponent(host-reference)

nested-component-reference:
        nestedRef:component-reference-name

toplevel-component-reference:
        toplevelRef:component-reference-name
        toplevelRef:component-reference-name@{install-path}
        toplevelRef(host-reference):component-reference-name
        toplevelRef(host-reference):component-reference-name@{install-path}

dependee-component-reference:
        dependee:dependency-name

container-component-reference:
        container

dependency-name:
component-reference-name:
        identifier

dynamic-session-variable-name:
host-name:
        [\p{L}_][\p{N}\p{L}-_. ]*

path-part:
component-name:
        [\p{N}\p{L}-_. ]+     (except "." and "..")

system-name:
        system-name-identifier
        plugin-name#system-name-identifer

plugin-name:
        identifier
        identifier . plugin-name

system-name-identifier:
        [\p{L}_][\p{N}\p{L}-_. +]*

component-version:
        [1-9][0-9]*.[0-9]+

install-path:
        complex-substitution-variable-reference
        path-literal

path-liternal:
        (([^}])|(}}))+
```

identifier :
```
[\p{L}_][\p{N}\p{L}_]*
```

用語集

抽象コンポーネント
ほかのコンポーネントを拡張するためのベースコンポーネントとしてのみ機能するコンポーネント。抽象コンポーネントはインストールすることはできません。また、抽象的な子要素を宣言できるのは抽象コンポーネントのみです。

呼び出しの互換性
システムサービスコンポーネントの互換性タイプの 1 つ。この互換性は、API 互換性やインタフェース互換性とも呼ばれます。

カテゴリ
複数のフォルダに保存されたグループオブジェクトをグループ化できる一般的なクラス。

子コンポーネント
コンテナコンポーネントが参照するコンポーネント。包含コンポーネント ともいいます。

*container component*も参照してください。

比較
ホストとコンポーネントモデル間の相違点を検索し、特定する機能。プロビジョニングシステムは、次に示す 3 種類の比較をサポートします。

- モデルとモデル – マスターサーバー上に保存された 2 つのホストの配備リポジトリと履歴が検査され、差分が報告されます。
- モデルとインストール – ホストにインタフェース済みであるとマスターサーバーから報告された内容とホストに実際にインストールされている内容が比較され、差分が報告されます。
- インストールとインストール – 2 つのホストのファイルシステムの内容が検査され、差分が報告されます。

コンポーネント
アプリケーションを定義するソース情報の論理グループ。コンポーネントは、ソース情報の管理方法を指定する一連の命令も含みます。

コンポーネントの XML 表現には、以下のものが含まれます。

- アプリケーションが使用するリソースの一覧
- インストール手順

- アンインストール手順
- 依存関係

コンポーネントの互換性	コンポーネントを別のコンポーネントで安全に置換できること。プロビジョニングシステムは、コンポーネントの互換性として呼び出しの互換性とインストールの互換性の 2 種類をサポートします。
コンポーネントの継承	コンポーネントが、別のコンポーネントの属性と動作を取得するための手法。作成したコンポーネントは、対応づけられたコンポーネントタイプから変数、スナップショット、および手続きを継承します。
コンポーネント手続き	コンポーネントの配備を制御するコンポーネント内のプログラム。インストール、アンインストール、管理、スナップショットのキャプチャなどの手続きがあります。管理手続きは、制御ブロックで定義されます。
コンポーネントリポジトリ	コンポーネントとそのリソースが登録される、マスターサーバー上の場所。
コンポーネントタイプ	ほかのコンポーネントが再利用できる動作をカプセル化する、特殊なコンポーネント。コンポーネントは、拡張によって特定のコンポーネントタイプの動作を継承できます。
コンポーネント変数	ユーザー定義が可能な、名前と値の組み合わせ。コンポーネント変数は、外部のオブジェクトからコンポーネントの一部にアクセスして設定できるようにするために使用されます。
複合コンポーネント	ほかのコンポーネント (単純コンポーネントおよび複合コンポーネント) の参照だけを含むコンポーネント。複合コンポーネントはリソースを含むことはできません。
複合プラン	サブプラン (単純サブプランまたは複合サブプラン) だけから成るプラン。　異なるターゲットセットで各サブプランが実行される可能性があるため、複合プランを直接のターゲットとすることはできません。
構成生成エンジン	置換変数参照を適切な変数設定値に置換する、マスターサーバー上のソフトウェアエンジン。このエンジンは、ユーザーがプランを実行してコンポーネントを配備する際に、ホストリポジトリおよびコンポーネントリポジトリとやりとりしながら値を解決します。
包含コンポーネント	ほかのコンポーネントが参照するコンポーネント。
コンテナコンポーネント	ほかのコンポーネントへの参照が含まれるコンポーネント。
制御	配備されたアプリケーションの制御に使用できるコンポーネントによって定義される手続き。たとえばアプリケーションの起動や停止などを制御できます。制御サービスとも呼ばれます。
配備	コンポーネントに対して行われる、プランまたはコンポーネント手続きを実行します。コンポーネントのライフサイクルとしては、インストール、アンインストール、アプリケーション管理などがあります。
直接実行手続き	ブラウザインタフェースを使用してコンポーネントから直接実行できるコンポーネント手続きです。

ダウンストリーム	Sun N1 Service Provisioning System ソフトウェアのネットワーク階層では、マスターサーバーまでの階層的距離がより遠いサーバー。たとえば、マスターサーバーはダウンストリームをローカルディストリビュータに接続します。このローカルディストリビュータに接続されるリモートエージェントはすべて、ローカルディストリビュータから見てダウンストリームになります。
`execNative` 呼び出し	プランまたはコンポーネントの XML からカスタムスクリプトに対して行われるオプション呼び出しです。
実行プラン	*plan*を参照してください。
拡張	コンポーネントタイプによって定義された変数と手続きをコンポーネントが継承するように、コンポーネントタイプをベースにしてコンポーネントを作成すること。コンポーネントは、そのコンポーネントタイプで定義された変数値と手続き定義を上書きできます。
最終コンポーネント	ほかのコンポーネントによって拡張できないコンポーネント。
フォルダ	コンポーネント、プラン、サブフォルダをまとめることができる、ディレクトリに似たコンテナ。コンテナにはアクセス許可を設定できます。
ゴールドサーバー	アプリケーションを構成するファイル、ディレクトリ、およびその他のリソースを格納する参照サーバー。これらのリソースは、ゴールドサーバーによってマスターサーバーにチェックインされます。
ホスト	プロビジョニングシステムが管理するサーバー。
ホストセット	1 つ以上の属性 (物理的な場所や機能グループなど) を共有するホストを論理的にグループ化したもの。ホストセットはユーザー定義が可能です。ホストセットを使用すると、セット内の全ホストのアプリケーションを簡単にすばやく更新できます。また、2 つのホスト間で「モデルとインストールの比較」を実施するのにも利用できます。
ホストタイプ	ユーザー定義が可能な一連の共通属性によってバインドされる、サーバーの基本クラス。ホストタイプを使用することで、ホストを論理グループとして分類し、ホスト検索の効率を高めることができます。
ホスト検索	ホストリポジトリに対して行われるクエリー。この結果、指定されたものに一致する属性を持つホストの一覧が表示されます。ホスト検索を行うことで、ホストタイプが同じであるホスト、同じアプリケーションを実行しているホスト、同じサブネットマスクが設定されているホストなどの一覧を作成できます。
インストールの互換性	コンポーネントタイプの互換性タイプの 1 つです。この互換性は、構造上の互換性とも呼ばれます。
Java Runtime Environment (JRE)	実行環境を再配布するユーザーおよび開発者用の Java Development Kit (JDK®) のサブセット。Java Runtime Environment は、Java 仮想マシン (JVM)、Java コアクラス、サポートファイルから構成されます。

Java 仮想マシン (JVM)	Java Runtime Environment (JRE) の一部であり、バイトコードを解釈します。
Jython	オブジェクト指向の高度な動的言語 Python の実装の 1 つ。Java プラットフォームとシームレスに統合されます。Jython の前身である JPython は、100% ピュア Java として認定されています。
ラベル	プロビジョニングシステムのバージョン番号という域を越えてコンポーネントバージョンをマークする手段。たとえば、コンポーネントバージョン番号はコンポーネントのバージョンを示すために使用できます。一方ラベルは、コンポーネントが表現するアプリケーションのバージョンを示すために使用できます。
ローカルディストリビュータ **(Local Distributor)**	サーバーにインストールされるアプリケーション。ローカルディストリビュータアプリケーションは次の方法で、プロビジョニングシステム内のそのほかのサーバー間のリンクとして動作します。 ■ マスターサーバーとリモートエージェントの間のリンク ■ マスターサーバーとほかのローカルディストリビュータの間のリンク ■ ローカルディストリビュータとリモートエージェントの間のリンク ローカルディストリビュータには帯域幅の効率と速度を最大化する効果があるほか、制限された環境をナビゲートするための安全なネットワーク接続も提供できます。
マスターサーバー	プロビジョニングシステムを管理するサーバー上にインストールされているアプリケーション。マスターサーバーアプリケーションは、プロビジョニングシステムによって管理される任意のデータセンター環境に接続できます。マスターサーバーは、一元的なデータストレージ、データ処理、ユーザーインタフェースなどを実現します。
モデル化	プロビジョニングシステムで配備するアプリケーションを表すコンポーネントおよびプランを作成すること。
入れ子のコンポーネント	インストールされたとき、そのコンテナコンポーネントにのみサービスを提供する包含コンポーネント。入れ子の包含コンポーネントは、コンテナコンポーネントが必要とする、よりきめの細かい機能単位を定義します。ただし、ほかのコンポーネントにとっては有用ではありません。
ネットワークプロトコル	ネットワーク上のデバイス間でデータを転送する方法。Sun N1 Service Provisioning System ソフトウェアは TCP/IP、SSH、および SSL を使用します。
通知電子メール	システム、管理、またはカスタムイベントが発生したことを通知するために、プロビジョニングシステムにより送信される電子メール。システム管理者は、電子メールを送信するタイミングと電子メールの送信先アドレスを決定するための規則を指定します。

通知規則	電子メール通知を送信するかどうかを決定するためにプロビジョニングシステムにより使用される基準。システム管理者は、電子メール通知が送信される場合の決定に使用される基準を定義します。
親コンポーネント	ほかのコンポーネントへの参照が含まれるコンポーネント。コンテナコンポーネントとも呼ばれます。 *contained componen*も参照してください。
物理ホスト	ネットワークに接続される物理サーバー。プロビジョニングシステムでは、物理ホストはリモートエージェントまたはローカルディストリビュータとして機能できます。
プラン	1 つ以上のコンポーネントを操作するために使用される一連の命令。プランは、ほかの一連のプランの一部になることもあります。これにより、共通の命令シーケンスを 1 つ以上のプランで共有できるようになります。
Plan executor	プリフライトと配備を行う、マスターサーバー上のソフトウェアエンジンです。
プリフライト	シミュレートされた UNIX 環境でのプランのシミュレーション。配備に影響を及ぼす可能性のあるエラーや潜在的なエラーを検出し報告します。予備テストは必ず、配備に先行しますが、単独の処理として実行できます。
手続き	*component procedure*を参照してください。
プロビジョニングシステム	サーバーにインストールされると、Sun N1 Service Provisioning System ソフトウェアを形成するソフトウェアアプリケーション。
リモートエージェント	コンポーネントの配備先であるプロビジョニングシステムの任意のサーバーにインストールするアプリケーション。リモートエージェントアプリケーションは、ソフトウェアのインストール、サービスの制御、マスターサーバーへ配布する情報の収集などの作業を管理します。
リソース	プランが実行されると、ホストに配備されるファイル。ディレクトリ、シンボリックリンク、または別の種類のファイルを使用できます。
リソース記述子ファイル	単純コンポーネントのリソースを構成するファイルとディレクトリに使用される、所有者、グループ、およびアクセス許可の設定を指定する XML ファイル。リソース記述子ファイルを使用することで、コンポーネントのチェックイン時に決定されるアクセス許可を上書きできます。
サーバー	リソースを管理し、クライアントにサービスを提供するコンポーネント。Sun N1 Service Provisioning System ソフトウェアでは、サーバーはプロビジョニングシステムアプリケーションの 1 つがインストールされているコンピュータです。

セッション	ユーザーがログインしたときに開始される一定期間。セッションは、ユーザーがログアウトするまで、または非アクティブな状態によりセッションが終了するまで続きます。論理上、セッションは、特定のユーザーの認証済みの資格情報を表します。セッションにより、再認証されなくても、一連の関連要求を通してユーザーが識別されます。
セッション変数	ユーザーセッションに関連する変数。ユーザーはログインセッションごとに、セッション変数の値を変更できます。セッション変数値をセキュリティー保護して保管し、以後のセッションで再利用することもできます。
単純コンポーネント	単一リソースからなるコンポーネント。単純コンポーネントには、ほかのコンポーネントへの参照を含めることはできません。
単純プラン	特定のターゲットサーバー (複数) で実行される手順の順次リスト。単純プランにはほかのプランは含まれず、ほかのプランを呼び出すこともありません。
スナップショット	配備時にホストに格納されるリソースのキャプチャ。スナップショットは、ホストと、マスターサーバー上のそのモデルを比較する場合に使用されます (モデルとインストールの比較)。
ステップ	プランまたはコンポーネントの一部をなす命令。
置換変数	プラン、コンポーネント、または構成ファイルに現れる変数で、配備中に構成生成エンジンに置換されます。
システムサービス	ホストを準備するときに、該当するすべてのホストに自動配備されるコンポーネント。システムサービスは、ほかのコンポーネントも使用するユーティリティコントロールやリソースを定義します。
ターゲット作成可能コンポーネント	インストールされたときに、ほかのコンポーネントの配備ターゲットとなるホストを作成するコンポーネント。ターゲット作成可能コンポーネントをアンインストールすると、そのコンポーネントによって作成されたホストが自動的に削除されます。
上位レベルコンポーネント	インストールされると、プランによって直接インストールされたかのように、どのコンポーネントでも使用される包含コンポーネント。上位レベルの包含コンポーネントは、コンテナコンポーネントやほかのコンポーネントによって使用されるサービスを定義します。
アップストリーム	プロビジョニングシステムネットワーク階層では、マスターサーバーに近いサーバー。たとえば、マスターサーバーはローカルディストリビュータから見てアップストリームになります。ローカルディストリビュータは、そのローカルディストリビュータに接続されたどのリモートエージェントから見てもアップストリームになります。
変数	*component variable*を参照してください。
変数設定値	1 つ以上のコンポーネント変数のデフォルト値を無効にする、変数値の集まり。使用する変数設定値に基づいて、コンポーネント変数にさまざまな値を指定できます。プランを実行するときの変数設定を指定できます。

仮想ホスト	ほかのサービスのホストとして機能するサービス。たとえば、仮想ホストは Web アプリケーションのホストとして機能するアプリケーションサーバーになることができます。
XML スキーマ	プロビジョニングシステムがプランやコンポーネントの作成において使用する言語。

索引

A
「Advanced Edit」ページ, プラン，42

C
cdb コンポーネントコマンド，29-31
「Components」ページ，25
　「Details」ページ，26
　「Edit 」ページ，27

D
「Details」ページ
　コンポーネント，26
　セッション変数，53
　プラン，41

E
「Edit 」ページ, コンポーネント，27
「Edit」ページ, セッション変数，53

P
pdb プラン管理コマンド，51-52
pe プラン実行コマンド，52
「Plans」ページ，41
　「Advanced Edit」ページ，42
　「Details」ページ，41

S
「Session Variables」ページ，53
　「Details」ページ，53
　「Edit」ページ，53
system#container コンポーネントタイプ，39
system#directory コンポーネントタイプ，35-37
system#file コンポーネントタイプ，33
system#symbolic link コンポーネントタイプ，37

T
targetableComponent コンポーネントの展開，63

U
udb セッション変数コマンド，54
untyped コンポーネントタイプ，40

V
viewing, plan details，52

X

XML
  オーサリング，14
  スキーマ，19
    コンポーネントのダウンロード，26
    コンポーネントの編集，26
    プラン用の CLI コマンド，29-31
XML でのオーサリング, ブラウザインタフェース，14

あ

アプリケーション, 完全自動のモデル化，13
アプリケーションのモデル化，11
  XML でのオーサリング，14
  拡張
    組み込みコンポーネントタイプ，13
  完全自動，13
  ブラウザインタフェースの使用，13, 14
  方式，13

い

依存先コンポーネントの展開，65
入れ子のコンポーネント，12
  展開，64
インストール, コンポーネントの場所，26

お

親コンポーネント，12, 17

か

解決, 外部コンポーネント置換変数参照，65-67
外部コンポーネント置換変数参照, 解決，65-67
隠しコンポーネント，15
  表示，15, 30
拡張, 組み込みコンポーネントタイプ，13
カスタムプラン, 作成，44-46
完全自動のアプリケーションのモデル化，13
管理
  コンポーネント，25, 29-31
  セッション変数，53, 54
管理 (続き)
  プラン，41, 51-52
  変数設定，29-31

く

組み込みコンポーネントタイプ，33-40
  `system#container`，39
  `system#directory`，35-37
  `system#file`，33
  `system#symbolic link`，37
  `untyped`，40

け

継承, コンポーネント，16
検証, 置換変数，57

こ

構成の生成
  エンジン，56
  概要，55
  生成コンテキスト，57
  置換変数，56
  入力ソース，57
ゴールドサーバー，13
子コンポーネント，12, 17
コマンド行インタフェース
  「ブラウザインタフェース」も参照
  `cdb` コンポーネントコマンド，29-31
  `pdb` プラン管理コマンド，51-52
  `pe` プラン実行コマンド，52
  `udb` セッション変数コマンド，54
  コンポーネントのチェックイン，31
コンテナコンポーネント，12, 17
  展開，65
コンポーネント
  CLI コマンド，29-31
  XML でのオーサリング，14
  アプリケーションのモデル化，13
  インストール先，26
  概念，11
  カテゴリ，15
  完全自動のモデル化，13

コンポーネント (続き)
管理，25, 29-31
組み込みコンポーネントタイプの拡張，13
継承，16
削除，28-29, 30
作成，27-28, 30
参照，20
事前定義された変数名，59
ステップ，20
説明，15
タイプ，12, 14
単純，12
チェックイン，30, 31
直接実行コンポーネント手続きによる配備，49-51
定義，11
手続き，15
特性，14
名前，14
バージョン，14
パス，14
非表示，15
表示，25
複合，12
プラットフォーム，15
編集，26, 30
変数，15, 16
変数設定，17
管理，26, 30
変数の優先指定，17
ラベル，15
リソース，15
コンポーネントタイプ，14
system#container，39
system#directory，35-37
system#file，33
system#symbolic link，37
untyped，40
概念，18
組み込み，33-40
プラグインから，33-40
コンポーネント置換変数参照，61
コンポーネントのチェックイン，28, 31

さ

サーバー, ゴールド，13
削除
コンポーネント，28-29, 30
制限，28
セッション変数，54
プラン，41, 51
作成
カスタムプラン，44-46
コンポーネント，27-28, 30
生成プラン，43-44
プラン，43
参照, コンポーネント，20

し

システムサービスコンポーネントの展開，62
システムサービスの概念，19
システムタイプコンポーネントの展開，63
事前定義されたコンポーネント変数名，59
事前定義のセッション変数，68
事前定義ホスト置換の短縮形，69
事前定義ホスト変数名，70
実行
コンポーネント手続き，15
直接実行コンポーネント手続き，15, 49-51
手続きをプランから，15
手続きをほかのコンポーネントまたはプランから，15
プラン，46-48
実行プラン，「プラン」を参照
自動生成プラン，「生成プラン」を参照
上位レベルコンポーネント，12
展開，64

す

ステップ，20

せ

制御，13
生成コンテキスト, 構成の生成，57
生成プラン, 作成，43-44
セッション置換変数の参照，67
セッション変数
CLI コマンド，54

セッション変数 (続き)
概念，23
削除，54
事前定義，68
詳細の表示，53
セキュリティー保護された，24
追加，56
表示，53
編集，53, 54

た
ターゲット置換変数の参照，68
短縮形, 事前定義ホスト置換，69
単純コンポーネント，12
単純置換変数参照，58
単純置換変数の展開，60
単純プラン，19

ち
置換変数，55
値，56
エスケープシーケンス，60
検証，57
種類，58
使用，71
追加，56
リポジトリ，56
置換変数参照
外部コンポーネント，61
単純，58
置換変数のエスケープシーケンス，60
置換変数の参照
セッション，67
ターゲット，68
置換変数の展開，60
直接実行コンポーネント手続き
コンポーネントの配備，49-51
実行，15

つ
追加
セッション変数，53, 54
追加 (続き)
置換変数，56
プラン，41, 51

て
手続き
定義，15
プランから実行，15
ほかのコンポーネントまたはプランから実行，15
展開
targetableComponent コンポーネント，63
依存先コンポーネント，65
入れ子のコンポーネント，64
コンテナコンポーネント，65
システムサービスコンポーネント，62
システムタイプコンポーネント，63
上位レベルコンポーネント，64
単純置換変数，60
置換変数，60
明示的外部コンポーネント，62

と
動的ホスト変数名，71

に
入力ソース, 構成の生成，57

ひ
表示
コンポーネント，25, 30
非表示，15, 30
コンポーネントの詳細，25, 30
セッション変数，53, 54
セッション変数の詳細，53, 54
プラン，41, 52
プランの詳細，41

ふ
複合コンポーネント，12
複合プラン，19
ブラウザインタフェース
「コマンド行インタフェース」も参照
「Components」ページ，25
「Details」ページ，26
「Edit 」ページ，27
「Plans」ページ，41
「Advanced Edit」ページ，42
「Details」ページ，41
「Session Variables」ページ，53
「Details」ページ，53
「Edit」ページ，53
XML オーサリング，14
XML でのオーサリング，13
アプリケーションのモデル化，13, 14
拡張
XML による，13
組み込みコンポーネントタイプ，13
管理
コンポーネント，25
セッション変数，53
プラン，41
コンポーネントのチェックイン，28
削除
コンポーネント，28-29
プラン，41
作成
カスタムプラン，44-46
コンポーネント，27-28
プラン，43
実行
プラン，46
表示
コンポーネント，25
コンポーネントの詳細，25
セッション変数，53
セッション変数の詳細，53
プラン，41
プランの詳細，41
編集
コンポーネント，26
セッション変数，53
プラン，42-43
プラン
CLI コマンド，51-52
概念，19
プラン (続き)
管理，41, 51-52
削除，41
作成，43
カスタム，44-46
生成，43-44
実行，46, 52
種類，19
ステップ，20
単純，19
定義，19
表示，41
複合，19
プリフライト，46
編集，42-43
プロビジョニングシステム
機能，11
目的，11
文法, 変数置換，77

へ
編集
コンポーネント，26, 30
セッション変数，53, 54
プラン，42-43
変数設定，30
変数
コンポーネント，16
事前定義されたコンポーネント，59
事前定義セッション，68
事前定義ホスト変数，70
セッション，53
置換
タイプ，55
変数参照
コンポーネント置換，61
置換，55
変数設定，17
コンポーネントの管理，26, 30
変数置換の文法，77
変数の参照, ターゲット置換，68
変数の優先指定，17
変数名
事前定義ホスト，70
動的ホスト，71

ほ
包含コンポーネント，12, 17
ホスト
　事前定義置換の短縮形，69
　事前定義の変数名，70
　リダイレクト，69

り
リソース，11
　コンポーネントの管理，29-31
リソース記述子ファイル，22
リポジトリ
　コンポーネント，31
　置換変数，56
　プラン，43