Solaris WBEM Services の管理

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650–960–1300

Part Number 806-3011–10
2000 年 3 月

原典: *Solaris WBEM Services Administrator's Guide*

Part No: 806-1791-10

Revision A

Adobe PostScript

目次

表

図

例

はじめに

このマニュアルでは、Common Information Model (CIM) の概念と、Solaris™ オペレーティング環境における Web-Based Enterprise Management (WBEM) の管理方法を説明しています。

Solaris WBEM Services ソフトウェアを使用すれば、ソフトウェア開発者は Solaris オペレーティング環境の管理を容易にする Solaris の管理アプリケーションを簡単に作成できます。

対象読者

このマニュアルは、WEBM 対応の環境を管理するシステム管理者を対象としています。

お読みになる前に

このマニュアルは、読者に次の知識があることを前提としています。

- オブジェクト指向プログラミングの概念
- Java™ プログラミング
- Common Information Model (CIM) の概念
- ネットワーク管理の概念

知識が不十分な場合には、次の書籍を参考にすることをお勧めします。

- 『*Java™ How to Program*』

  H. M. Deitel、P. J. Deitel 著、Prentice Hall 発行、ISBN 0–13–263401–5

- 『*The Java Class Libraries*』第 2 版、第 1 巻、Patrick Chan、Rosanna Lee、Douglas Kramer、Addison-Wesley 著、ISBN 0–201–31002–3

- 『*CIM Tutorial*』、『*CIM Tutorial*』、Distributed Management Task Force 提供

次に、WBEM 技術に携わる場合に有用な Web サイトを示します。

- Distributed Management Task Force (DMTF)

  このサイト (`http://www.dmtf.org`) には、CIM の最新の開発情報、各種の作業グループについての情報、CIM スキーマの拡張についての問い合わせ方法などが掲載されています。

- Rational Software

  このサイト (`http://www.rational.com/uml`) では、Unified Modeling Language (UML) と Rose CASE ツールの関連文書を入手できます。

このマニュアルの構成

第 1 章では、Solaris WBEM Services と Web-Based Enterprise Management (WBEM) の概要を説明しています。

第 2 章では、CIM Object Manager とは何かを説明し、さらにその起動と停止方法を説明しています。

第 3 章では、セキュリティ機能とネームスペースやユーザーのアクセス権の設定方法を説明しています。

第 4 章では、`mofcomp` コマンドのコマンド構文と `.mof` ファイルのコンパイル方法を説明しています。

第 5 章では、ロギング機能について説明しています。

第 6 章では、Solaris WBEM Services 製品のコンポーネントが生成するエラーメッセージについて説明しています。

付録 Aでは、Common Information Model (CIM) の一般的な概念を説明しています。

付録 Bでは、Solaris オペレーティング環境の管理オブジェクトを記述する Managed Object Format (MOF) ファイルである Solaris Schema ファイルについて説明しています。

用語集では、このマニュアルで使用されている用語について説明しています。

Sun のマニュアルの注文方法

専門書を扱うインターネットの書店 Fatbrain.com から、米国 Sun Microsystems™, Inc. (以降、Sun™ とします) のマニュアルをご注文いただけます。

マニュアルのリストと注文方法については、`http://www1.fatbrain.com/documentation/sun` の Sun Documentation Center をご覧ください。

Sun のオンラインマニュアル

`http://docs.sun.com` では、Sun が提供しているオンラインマニュアルを参照することができます。マニュアルのタイトルや特定の主題などをキーワードとして、検索をおこなうこともできます。

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・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'**

ただし AnswerBook2™ では、ユーザーが入力する文字と画面上のコンピュータ出力は区別して表示されません。

コード例は次のように表示されます。

- C シェルプロンプト

```
system% command y|n [filename]
```

- Bourne シェルおよび Korn シェルのプロンプト

```
system$ command y|n [filename]
```

- スーパーユーザーのプロンプト

```
system# 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。

一般規則

- このマニュアルでは、英語環境での画面イメージを使っています。このため、実際に日本語環境で表示される画面イメージとこのマニュアルで使っている画面イメージが異なる場合があります。本文中で画面イメージを説明する場合には、日本語のメニュー、ボタン名などの項目名と英語の項目名が、適宜併記されています。
- このマニュアルでは、「IA」という用語は、Intel 32 ビットのプロセッサアーキテクチャ を意味します。これには、Pentium、Pentium Pro、Pentium II、Pentium II Xeon、Celeron、Pentium III、Pentium III Xeon の各プロセッサ、および AMD、Cyrix が提供する互換マイクロ プロセッサチップが含まれます。

第 1 章

概要

この章では、Web-Based Enterprise Management (WBEM) と Solaris WBEM Services の概要を説明します。これらのソフトウェアを使用すれば、ソフトウェア開発者は、Solaris オペレーティング環境の管理を容易にする Solaris の管理アプリケーションを簡単に作成できます。

内容は次のとおりです。

- WBEM について
- Common Information Model について
- Solaris WBEM Services ソフトウェア
- Sun WBEM Software Development Kit

WBEM について

業界全体の取り組みである Web-Based Enterprise Management (WBEM) には、異機種プラットフォーム上のシステム、ネットワーク、デバイスを Web 上で管理するための標準規格が含まれています。この標準化により、システム管理者はデスクトップ、デバイス、およびネットワークを管理することができます。

WBEM は、既存の主要な管理プロトコルに対応するように設計されています。対応するプロトコルは、Simple Network Management Protocol (SNMP)、Distributed Management Interface (DMI)、Common Management Information Protocol (CMIP) などです。

WBEM には次の標準規格が含まれています。

- Common Information Model (CIM) – 管理資源を記述するための情報モデル
- Managed Object Format (MOF) – CIM クラスとインスタンスを定義するための言語
- eXtensible Markup Language (XML) – Web 上の管理リソースを記述するためのマークアップ言語

コンピュータ業界とテレコミュニケーション業界の企業を代表するグループの 1 つである Distributed Management Task Force (DMTF) は、管理の標準規格の開発では主導的な立場にあります。DMTF の目的は、さまざまなプラットフォームおよびプロトコルに渡ってネットワークを管理する統合的な手法を開発し、費用効率の高い相互運用性に優れた製品を提供することにあります。DMTF の提唱とその現況については、このグループの Web サイト `http://www.dmtf.org` を参照してください。

Common Information Model について

この節では、Solaris WBEM Services 製品で使用されている CIM の基本的な用語と概念を簡単に説明します。CIM の詳細は、付録 A を参照してください。

CIM とは、ディスク、CPU、オペレーティングシステムなどの管理資源を記述するためのオブジェクト指向情報モデルです。CIM オブジェクトは、プリンタ、ディスクドライブ、CPU などの管理リソースを表すモデルです。CIM オブジェクトは、WBEM 対応のシステムやデバイス、アプリケーションの間で共有できます。

基本的な CIM 要素

類似したプロパティや目的をもつ CIM オブジェクトは、CIM クラスで表されます。プロパティは、クラスのある単位データを記述する属性です。インスタンスは、特定のクラスの実際の管理オブジェクトを表したものです。インスタンスには実際のデータが含まれています。たとえば、`solaris_computersystem` が Solaris オペレーティングシステムを表す CIM クラスのときは、実際にシステムで動作している Solaris オペレーティング環境は `solaris_computersystem` クラスのインスタンスとして表されます。`ResetCapability` や `InstallDate` は `solaris_computersystem` クラスのプロパティです。

CIM クラスは、スキーマと呼ばれる意味のある集合にグループ分けされます。スキーマは複数のクラスからなるグループですが、クラスは 1 つのスキーマにしか所属できません。スキーマの所有者は 1 人だけです。スキーマは、管理やクラスの名前付けに使用されます。同じスキーマ内ではすべてのクラス名が異なっていなければなりません。スキーマが異なればクラスやプロパティの名前が同じになることがありますが、クラスとプロパティの区別にはスキーマ名が使用されます。スキーマ、クラス、プロパティの名前は、次の構文に従って付けられます。

Schemaname_classname.propertyname

CIMモデル

Common Information Model は、情報を一般的なものから特定のものへと分類します。Solaris の環境などの特定の情報は、このモデルを拡張して記述されています。CIM は、次に示す 3 つの情報層から構成されます。

- コアモデル – プラットフォームに依存しない、CIM のサブセット
- 共通モデル – ネットワーク管理の特定の領域に関連するエンティティ (システム、デバイス、アプリケーションなど) の概念、機能性、および表示方法を視覚的に表す情報モデル
- エクステンション (拡張)– CIM スキーマをサポートし、限定されたプラットフォーム、プロトコル、または企業独自のものを表す情報モデル

コアモデルと 共通モデルを、総称して CIM スキーマと呼びます。

コアモデル

コアモデルは、管理環境の基本となる一般的な前提事項を提供します (たとえば、要求された特定のデータはある場所に格納され、要求元のアプリケーションまたはユーザーに配付されなければならないなど)。これらの前提事項は、管理環境の基盤を概念的に形成する、クラスと関連のセットとして示されます。コアモデルは、管理環境の特定の側面を表現するスキーマに一貫性を持たせます。

コアモデルは、クラス、関連、およびプロパティのセットをアプリケーション開発者に提供します。開発者は、このセットを使用して管理対象システムを表現し、共通モデルを拡張する方法を決定することができます。コアモデルは、その他の管理環境をモデル化する概念的な枠組みを確立します。

コアモデルは、 共通モデルとエクステンション (拡張) により、システム、アプリケーション、ネットワーク、デバイスなどのネットワーク機能に関する特定の情報を拡張するためのクラスと関連を提供します。

共通モデル

共通モデルで示されるネットワーク管理の領域は、特定の技術や実装には依存しない管理アプリケーションの開発基盤を提供します。このモデルは、指定された 5 つの技術別スキーマ、Systems、Devices、Applications、Networks、および Physical に、拡張用の基底クラスセットを提供します。

CIM エクステンション (拡張)

拡張スキーマは、このモデルに特定の技術を関連づけるために CIM に組み込まれます。CIM を拡張すると、多数のユーザーと管理者が Solaris などの特定のオペレーティング環境を使用できるようになります。拡張スキーマのクラスを使用して、ソフトウェア開発者は拡張された技術を管理するアプリケーションを開発することができます。Solaris スキーマは CIM スキーマを拡張したものです。

Solaris WBEM Services

Solaris WBEM Services は、Solaris 8 オペレーティング環境で Web-Based Enterprise Management (WBEM) サービスを提供するソフトウェアです。このようなサービスを使用すれば、ソフトウェア開発者は、Solaris オペレーティング環境の管理を容易にする Solaris の管理アプリケーションを簡単に作成できます。

Solaris WBEM Services ソフトウェアでは、セキュリティが侵害されることなく管理データにアクセスし操作することができます。製品には Solaris Provider が組み込まれているため、管理アプリケーションから Solaris オペレーティング環境の管理リソース (デバイスやソフトウェア) の情報にアクセスできます。

管理アプリケーションは、RMI や XML/HTTP プロトコルを使って CIM Object Manager に接続します。CIM Object Manager は、接続されたクライアントに次のサービスを提供します。

- 管理サービス

  CIM Object Manager は、CIM データの意味と構文を検査し、アプリケーション、CIM Repository、管理リソースとの間でデータの送受信を行います。

- セキュリティサービス

  管理者はユーザーの CIM 情報へのアクセスを制御できます。

- ロギングサービス

  このサービスを構成するクラスを使えば、開発者は、イベントデータを動的にログレコードに記録し、ログレコードからデータを取り出すことができるアプリケーションを作成できます。管理者はこのデータを使ってイベントの原因を追跡したり、判定したりすることができます。

- XML データを CIM クラスに変換する XML サービス

  XML/HTTP ベースの WBEM クライアントで CIM Object Manager と通信できます。

WBEM 対応システムに接続されると WBEM クライアントは、次のような WBEM 操作を要求できます。CIM クラスとインスタンスの作成、表示、削除や、指定する値をもつプロパティの検索、指定するクラス階層にあるインスタンスやクラスの列挙 (リストの取得) などです。

ソフトウェアのコンポーネント

Solaris WBEM Services ソフトウェアは、アプリケーション、管理、プロバイダという 3 つの層で機能するソフトウェアコンポーネントで構成されます。これらのコンポーネントはオペレーティングシステム層やハードウェア層とデータをやりとりします。図 1–1 は、各層におけるソフトウェアコンポーネントとその関係を示しています。

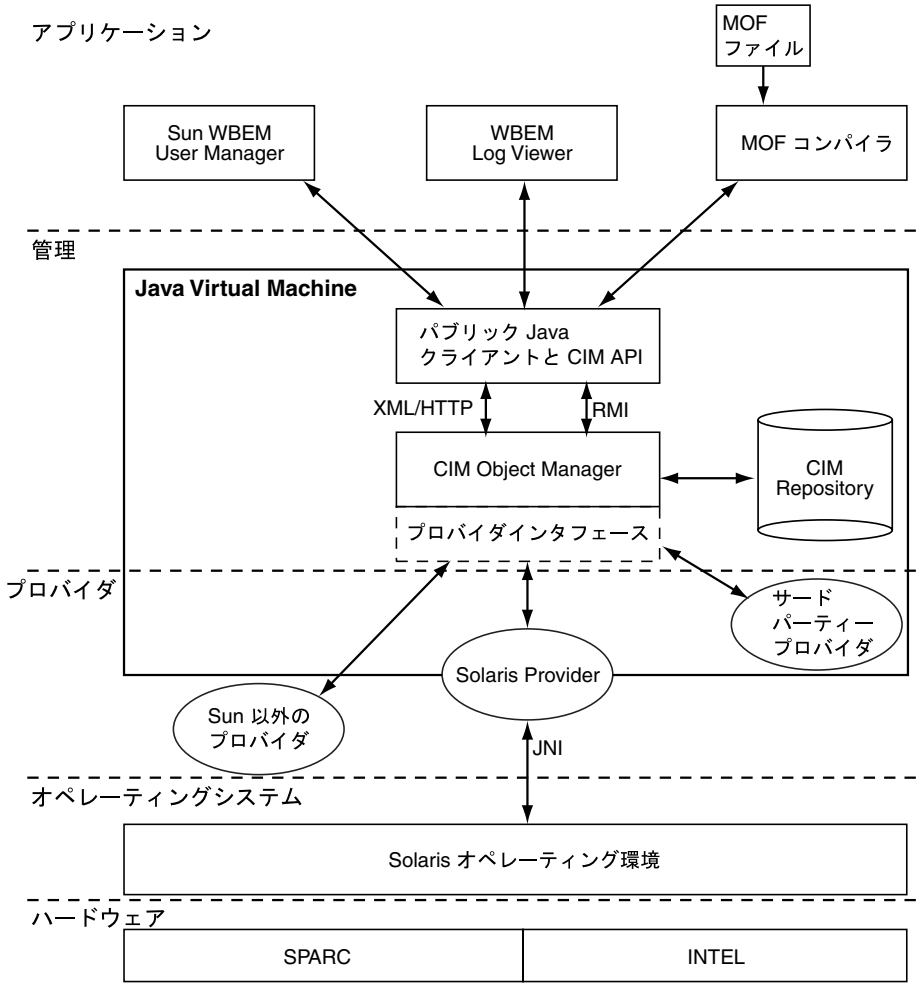

図 1–1　Solaris WBEM Services のアーキテクチャ

- アプリケーション層 – WBEM クライアントが管理リソースからのデータを処理したり、表示したりします。Solaris WBEM Services には、次のアプリケーションがあります。

  - WBEM Log Viewer – ログファイルを表示するアプリケーションです。Log Viewer では、ログファイルに記録されているコマンドを実行したユーザーの名前や、記録されているイベントが起きたクライアントコンピュータなど、ログレコードの詳細を表示できます。
  - Sun WBEM User Manager – このアプリケーションでは、管理者が、承認された WBEM ユーザーの追加や削除を行なったり、管理リソースに対するそれらのユーザーのアクセス権を設定したりすることができます。

- Managed Object Format (MOF) コンパイラ – このプログラムでは、MOF を宣言しているファイルを解析し、ファイルに定義されているクラスやインスタンスを Java クラスに変換し、その Java クラスを CIM Object Manager Repository に追加します。Repository は、管理データを一元的に格納する場所です。

  MOF は CIM クラスやインスタンスを定義する言語です。MOF ファイルは、MOF 言語を使って CIM オブジェクトを記述する ASCII テキストファイルです。CIM オブジェクトは、プリンタ、ディスクドライブ、CPU などの管理リソースを表したモデルです。

  管理リソースの情報は MOF ファイルに格納されることがあります。MOF は Java に変換できるため、Java Virtual Machine をもつシステムで動作するアプリケーションならこの情報の解釈や交換を行うことができます。さらに、インストールの後で、`mofcomp` コマンドを使って MOF ファイルをいつでもコンパイルできます。MOF の詳細は、DMTF の Web ページ `http://www.dmtf.org` を参照してください。

- 管理層 – この層のコンポーネントは、接続された WBEM クライアントに次のサービスを提供します。

  - クライアントおよび CIM アプリケーションプログラミングインタフェース (API) – WBEM クライアントアプリケーションは、これらの Java インタフェースを使って、管理リソースのクラスやインスタンスの作成や表示などの操作を CIM Object Manager に要求します。
  - Common Information Model (CIM) Object Manager – WBEM システム上の CIM オブジェクトを管理するソフトウェアです。CIM オブジェクトは、プリンタ、ディスクドライブ、CPU などの管理リソースを表したモデルです。CIM オブジェクトは内部的には Java クラスとして格納されます。CIM Object Manager は WBEM クライアント、CIM Object Manager Repository、管理リソースとの間で情報を送受信します。
  - CIM Object Manager Repository – CIM クラスやインスタンスの定義を一元的に格納する場所です。
  - プロバイダインタフェース – プロバイダは、これらのインタフェースを使って管理リソースの情報を CIM Object Manager に転送します。CIM Object Manager は、これらのインタフェースを使って、ローカルにインストールされたプロバイダに情報を転送します。

- プロバイダ層 – プロバイダは、CIM Object Manager と 1 つまたは複数の管理リソースとの間の仲介を行います。CIM Object Manager は、WBEM クライアント

から CIM Object Manager Repository にないデータを要求されると、要求を適切なプロバイダに転送します。

- Solaris Provider – Solaris オペレーティング環境にある管理リソースの CIM Object Manager インスタンスを提供します。プロバイダは、管理デバイスに関する情報の取得や設定を行います。ネイティブプロバイダとは、管理デバイスで動作するように作成されたマシン固有のプログラムです。たとえば、Solaris システムのデータにアクセスするプロバイダには、通常、Solaris システムの照会を行う C 関数が含まれています。JNI (Java Native Interface) は、JDK に含まれている Java 用ネイティブプログラミングインタフェースです。JNI を使ってプログラムを作成すれば、コードはどのプラットフォームでも完全に動作します。JNI を使うと、Java Virtual Machine (VM) で動作する Java コードは、C、C++、アセンブリなど他の言語で作成されたアプリケーションやライブラリとともに動作します。
- Solaris スキーマ – Solaris オペレーティング環境にある管理オブジェクトを記述するクラスの集合です。CIM スキーマや Solaris スキーマのクラスは CIM Object Manager Repository に格納されます。CIM スキーマは、すべての管理環境で発生する管理オブジェクトを表すためのクラス定義の集合です。

  Solaris スキーマは CIM スキーマを拡張したクラス定義の集合であり、一般的な Solaris オペレーティング環境にある管理オブジェクトを表します。さらに、ユーザーは、インストールの後で、`mofcomp` コマンドを使って CIM スキーマ、Solaris スキーマ、あるいはその他のクラスを CIM Object Manager Repository に追加できます。

- オペレーティングシステム層 – Solaris プロバイダを使えば、管理アプリケーションから Solaris オペレーティング環境にある管理リソース (デバイスやソフトウェアなど) の情報にアクセスできます。
- ハードウェア層 – 管理クライアントは、サポートされる Solaris プラットフォームの管理データにアクセスできます。

ネームスペース

ネームスペースと呼ぶディレクトリのような構造には、1 つまたは複数のスキーマを格納できます。CIM ネームスペースには、他のネームスペース、クラス、インスタンス、修飾子型を格納できます。ネームスペース内のオブジェクト名は固有のものにします。

Solaris WBEM Services では、WBEM クライアントアプリケーションが特定のネームスペースに接続されると、それ以後のすべての操作はそのネームスペース内で行われます。ネームスペースに接続されているクライアントは、そのネームスペースとそのネームスペースに含まれるすべてのネームスペースのクラスやインスタンスに (存在すれば) アクセスできます。たとえば、child というネームスペースを root\cimv2 ネームスペースに作成すれば、root\cimv2 に接続することにより、root\cimv2 と root\cimv2\child ネームスペースのクラスやインスタンスにアクセスできます。

アプリケーションを、あるネームスペース内の下にあるネームスペースに接続することができます。これはディレクトリ内のサブディレクトリを使用するのと同じことです。アプリケーションを新しいネームスペースに接続すると、それ以後のすべての操作はそのネームスペース内で行われます。たとえば、アプリケーションを root\cimv2\child ネームスペースに接続すると、そのネームスペースのすべてのクラスやインスタンスにはアクセスできますが、親ネームスペース root\cimv2 のクラスやインスタンスにはアクセスできません。

インストール時に 3 つのネームスペースがデフォルトで作成されます。

- root – 他のネームスペースを含む一番上のネームスペースです。
- root\cimv2 – LogicalDisk、Netcard など、システムのオブジェクトを表すデフォルトの CIM クラスとインスタンスが入っています。これがデフォルトのネームスペースです。
- root\security – ユーザーやネームスペースのアクセス権を表すセキュリティクラスが入ります。これらのクラスは CIM Object Manager が使用します。

プロバイダ

WBEM クライアントアプリケーションが CIM データにアクセスすると、WBEM システムは現在のホストにあるユーザーのログイン情報を検証します。ユーザーには、デフォルトで CIM スキーマと Solaris スキーマに対する読み取り権が与えられます。CIM スキーマには、システムにあるすべての管理オブジェクトが標準形式で記述されています。WBEM 対応のすべてのシステムやアプリケーションは、この標準形式を解釈できます。

プロバイダとは、管理オブジェクトと通信してデータにアクセスするクラスです。プロバイダは、統合と解釈のためにこの情報を CIM Object Manager に転送しま

す。CIM Object Manager は、CIM Object Manager Repository にないデータを管理アプリケーションから要求されると、要求をプロバイダに転送します。

CIM Object Manager は、オブジェクトプロバイダのアプリケーションプログラミングインタフェース (API) を使ってプロバイダと通信します。

アプリケーションが動的データを CIM Object Manager に要求すると、CIM Object Manager はプロバイダインタフェースを使って要求をプロバイダに渡します。

プロバイダは、CIM Object Manager の要求に対し次のことを行います。

- 固有の情報形式を CIM クラスに変換します。
  - デバイスから情報を取得します。
  - この情報を CIM クラスの形で CIM Object Manager に渡します。
- CIM クラスからの情報を固有のデバイス形式に変換します。
  - 必要な情報を CIM クラスから取得します。
  - この情報を固有のデバイス形式でデバイスに渡します。

他の WBEM システムとの相互運用性

WBEM クライアントと WBEM システムは同じシステムで動作することも、別々のシステムで動作することもできます。WBEM システムには複数の WBEM クライアントを接続できます。一般的な WBEM システムでは、4 つから 5 つの WBEM クライアントがサポートされます。

Solaris WBEM Services は、Version 1.0 Specification for CIM Operations over HTTP をサポートします。この仕様では、XML を使って CIM のオブジェクトやメッセージを記述します。XML は、Web 上のデータを記述するための標準マークアップ言語です。この仕様では、XML マークアップを拡張して CIM のオブジェクトや操作を定義します。XML は Web 上に送信可能なデータを記述する標準的な方法であるため、WBEM クライアントは、XML データを解析できる任意の WBEM システム上の CIM データにアクセスできます。

Sun WBEM Software Development Kit

Sun WBEM Software Development Kit (SDK) には、WBEM 対応の管理デバイスと通信する管理アプリケーションの作成に必要なコンポーネントが含まれています。さらに、開発者はこのツールキットを使ってプロバイダを作成できます。プロバイダとは管理オブジェクトと通信してデータにアクセスするプログラムです。Sun WBEM SDK を使って開発したすべての管理アプリケーションは Java プラットフォームで実行できます。

WBEM クライアントアプリケーションは、Sun WBEM SDK API を使って CIM オブジェクトを操作するプログラムです。クライアントアプリケーションは、通常、CIM API を使ってオブジェクト (ネームスペース、クラス、インスタンスなど) を構築し、初期化します。次に、クライアントアプリケーションは Client API を使ってオブジェクトを CIM Object Manager に渡し、CIM のネームスペース、クラス、インスタンスの作成などの WBEM 操作を要求します。

Sun WBEM SDK は、どの Java 環境にでもインストールし、実行できます。Sun WBEM SDK は、スタンドアロンアプリケーションとして使用することも、Solaris WBEM Services といっしょに使用することもできます。Sun WBEM SDK は、`http://www.sun.com/developers/tools/sw_overview.html#foundation` の Sun Developer Connection から入手できます。

第 2 章

CIM Object Manager

Common Information Model (CIM) Object Manager は、WBEM クライアントアプリケーションや管理資源との間で CIM データを送受信するソフトウェアです。

この章の内容は次のとおりです。

- CIM Object Manager について
- `init.wbem` コマンド
- CIM Object Manager の停止
- CIM Object Manager の再起動
- 例外メッセージ

CIM Object Manager について

CIM Object Manager は、WBEM 対応のシステム上で CIM オブジェクトを管理します。CIM オブジェクトは、プリンタ、ディスクドライブ、CPU などの管理資源を表したモデルです。CIM オブジェクトは内部的には Java クラスとして格納されます。

WBEM クライアントアプリケーションが CIM オブジェクトの情報にアクセスすると、CIM Object Manager はそのオブジェクトに適したプロバイダか、CIM Object Manager Repository に接続します。プロバイダとは管理オブジェクトと通信してデータにアクセスするクラスです。WBEM クライアントアプリケーションが CIM Object Manager Repository にない管理リソースのデータを要求すると、CIM Object Managerは管理リソースのプロバイダにその要求を転送します。プロバイダは情報を動的に取得します。

CIM Object Manager は起動時に次のことを行います。

- RMI ポート 5987 の RMI 接続と HTTP ポート 80 の XML/HTTP 接続を待ちます。
- CIM Object Manager Repository との接続を設定します。
- 要求が来るのを待ちます。

通常の操作では、CIM Object Manager は次のことを行います。

- セキュリティ検査を行なって、ユーザーログインと、ネームスペースへのアクセス権を認証します。
- CIM データ操作の構文検査と意味的な検査を行なって、最新の CIM 仕様に準拠しているか確認します。
- 適切なプロバイダか CIM Object Manager Repository に要求を転送します。
- プロバイダや CIM Object Manager Repository から受け取ったデータを WBEM クライアントアプリケーションに転送します。

WBEM クライアントアプリケーションは、CIM クラスの作成や CIM インスタンスの更新などの WBEM 操作を行う場合、CIM Object Manager にアクセスして接続を確立します。WBEM クライアントアプリケーションは、CIM Object Manager に接続されると、CIM Object Manager への参照を取得し、この参照を使ってサービスや操作を要求します。

init.wbem コマンド

/etc/init.d/init.wbem start | stop

`init.wbem` ユーティリティは、インストール中や、システムのリブートのたびに自動的に起動されます。このユーティリティは CIM Boot Manager (`cimomboot`) を起動します。これは WBEM クライアントからの接続要求を待つプロセスです。クライアントが接続を要求すると、`cimomboot` プログラムが Common Information Model (CIM) Object Manager を起動します。一般には CIM Object Manager を停止する必要はありませんが、既存のプロバイダを変更する場合は、そのプロバイダを使用する前に CIM Object Manager を停止してから再起動する必要があります。

`init.wbem` コマンドには 2 つの引数があります。

- `start` – ローカルホストの CIM Boot Manager を起動します。
- `stop` – ローカルホストの CIM Boot Manager を停止します。

`init.wbem` スクリプトは `/etc/init.d` ディレクトリにインストールされています。`init.wbem` スクリプトはシステムのリブート時に実行されますが、これ以外にも init state 2 に入ったときに `/etc/rc2.d/S90wbem start` によって実行され、init state 0 に入ったときに `/etc/rc0.d/K36wbem stop` によって実行されます。

CIM Object Manager の停止

プロバイダを変更する場合は、変更したプロバイダを使用する前に CIM Object Manager を停止し、再起動する必要があります。

▼ CIM Object Manager を停止する方法

1. システムプロンプトで次のコマンドを入力して、スーパーユーザーになります。

```
% su
```

2. メッセージに従い、スーパーユーザーのパスワードを入力します。

3. 次のコマンドを入力して、**CIM Object Manager** を停止します。

```
# /etc/init.d/init.wbem
```

CIM Object Manager が停止します。

CIM Object Manager の再起動

CIM Object Manager を再起動するには、`init.wbem start` コマンドを使用します。`init.wbem` コマンドは、WBEM クライアントからの接続要求を待つプロセス CIM Boot Manager (`cimomboot`) を起動します。

▼ CIM Object Manager を再起動する方法

1. システムプロンプトで次のコマンドを入力して、スーパーユーザーになります。

```
% su
```

2. メッセージに従い、スーパーユーザーのパスワードを入力します。

3. 次のコマンドを入力して、**CIM Boot Manager** を再起動します。

```
# /etc/init.d/init.wbem
```

CIM Boot Manager が起動され、クライアントからの接続要求を待ちます。クライアントが接続を要求すると、CIM Boot Manager は CIM Object Manager を起動します。

例外メッセージ

CIM Object Manager は、MOF の構文や意味が正しくないと例外メッセージを生成します。例外メッセージの説明については、第 6 章を参照してください。

第 3 章

セキュリティの管理

この章では、CIM Object Manager が提供するセキュリティ機能について説明します。取り上げる内容は次のとおりです。

- 概要
- Sun WBEM User Manager によるアクセス制御の設定
- API によるアクセス制御の設定
- 例外メッセージ

概要

CIM Object Manager は、CIM Object Manager が動作しているマシンでのユーザーのログイン情報を検証します。検証されたユーザーには、Common Information Model (CIM) スキーマ全体に対する制御アクセス権が与えられます。CIM Object Manager は、個々のクラスやインスタンスのようなシステムリソースに対するセキュリティは提供しません。しかし、CIM Object Manager を使用すると、ネームスペースへのアクセス権を全体的に制御でき、個々のユーザーベースでもアクセス権を制御できます。

セキュリティ関連の情報はすべて、`root\Security` ネームスペースに入った、セキュリティクラスのインスタンスによって表されます。これらの情報は、永続的にこのネームスペースに置く必要があります。

次のセキュリティ機能は、WBEM 対応のシステム上の CIM オブジェクトへのアクセス権を制限します。

- 認証 – コンピュータシステム内のエンティティ (ユーザー、デバイスなど) の識別情報を検証するプロセス。認証は、システム内のリソースへのアクセスを許可する場合の必須条件となることが多い
- 承認 – ユーザー、プログラム、またはプロセスにアクセス権を与えること
- 再実行保護 – クライアントは、CIM Object Manager に送られた別のクライアントの最後のメッセージをコピーできない。CIM Object Manager は、クライアント鍵を使用し、後続のクライアントサーバーセッションすべての通信を同じクライアントと行うことを保証する

  CIM Object Manager は、ディジタル形式で署名された秘密セッション鍵を検証することにより、サーバーに対するクライアントのメッセージを別のクライアントが取得、送信することを防ぐ。CIM Object Manager は、有効な秘密セッション鍵のないクライアントからは同一のバイトストリームでも受け付けない
- デジタル署名 – CIM Object Manager は、Java ディジタル署名クラスを使用してサーバーに対するクライアントの応答にディジタル形式で署名を行うが、クライアントに対するサーバーの応答にはディジタル署名を行わない

認証

ユーザーがログインしユーザー名とパスワードを入力する場合、クライアントはそのパスワードを暗号化し、暗号化されたパスワードを CIM Object Manager に送ります。ユーザーが認証されると、CIM Object Manager はクライアントセッションを設定します。その後のオペレーションはすべて、セキュリティが保護されたそのクライアントセッションで行われます。

承認

ユーザーがログインし、ユーザー名とパスワードを入力すると、クライアントはパスワードを暗号化し、それを CIM Object Manager に送信します。ユーザーが認証されると、CIM Object Manager はクライアントセッションを設定します。それ以後のすべての操作は、その安全なクライアントセッションの中で行われます。

ユーザーの識別情報が CIM Object Manager によって認証されると、その識別情報を使用して、アプリケーションまたはそのタスクの実行をそのユーザーに許可すべきかどうかを検証できます。CIM Object Manager は資格ベースの承認を行うことができるため、特権を持つユーザーは読み取り権と書き込み権を特定のユーザーに割

り当てることができます。これらの承認は、既存の Solaris ユーザーアカウントに追加されます。

Sun WBEM User Manager によるアクセス制御の設定

Sun WBEM User Manager では、特権を持つユーザーは承認されたユーザーの追加と削除、およびアクセス権の設定などが行えます。このアプリケーションは、ユーザー認証の管理、および WBEM 対応のシステム上の CIM オブジェクトへのアクセスに使用してください。ユーザーには、Solaris ユーザーアカウントが必要です。

アクセス特権は、個別のネームスペースや、ユーザーとネームスペースの組み合せに対して設定できます。ユーザーを追加し、ネームスペースを選択すると、ユーザーにはデフォルトで、選択したネームスペース内の CIM オブジェクトに対する読み取り権が与えられます。ユーザーとネームスペースのアクセス権を効果的に組み合せるには、アクセス権をあるネームスペースに限定してから、個別のユーザーにそのネームスペースに対する読み取り権、読み取り/書き込み権、または書き込み権を与えます。

個別の管理オブジェクトにアクセス権を設定することはできませんが、ネームスペースまたは個々のユーザーのすべての管理オブジェクトに対してアクセス権を設定できます。

root アカウントにログインすると、ユーザーおよび CIM オブジェクトに対し次のアクセス権を設定できます。

- 読み取り権のみ – CIM スキーマオブジェクトへの読み取り権だけを許可する。このアクセス権を持つユーザーは、インスタンスとクラスの検索は行えるが、CIM オブジェクトの作成、削除、および変更は行えない
- 読み取り権と書き込み権 – すべての CIM クラスおよびインスタンスへの読み取り権、書き込み権、および削除権を与える
- 書き込み権 – すべての CIM クラスとインスタンスに対する書き込み権と削除権は許可するが、読み取り権は許可しない
- アクセス権なし – CIM クラスとインスタンスへのアクセス権がない

▼ Sun WBEM User Manager を起動する方法

1. コマンドウィンドウで、次のコマンドを入力します。

```
% /usr/sadm/bin/wbemadmin
```

Sun WBEM User Manager が起動され、「ログイン (Login)」ダイアログボックスが表示されます。ダイアログボックスのいずれかのフィールドをクリックすると、そのコンテキストヘルプが「コンテキストヘルプ (Context Help)」パネルに表示されます。

2. 「ログイン **(Login)**」ダイアログボックスで、次の操作を行います。
   - 「ユーザー名 (User Name)」フィールドにユーザー名を入力します。

     ログインするには、`root\security` ネームスペースへの読み取り権が必要です。Solaris ユーザーはデフォルトでゲスト特権をもっているため、Solaris ユーザーにはデフォルトのネームスペースに対する読み取り権が与えられます。読み取り権をもつユーザーはユーザー特権の表示はできますが、変更はできません。

     他のユーザーに権限を与えるためには、root としてログインするか、`root\security` ネームスペースへの書き込み権をもつユーザーとしてログインする必要があります。
   - 「パスワード (Password)」フィールドで、ユーザーアカウントのパスワードを入力します。

3. 「了解 **(OK)**」をクリックします。

「ユーザーマネージャ (User Manager)」ダイアログボックスが開かれ、ユーザー名と、現在のホスト上のネームスペース内の WBEM オブジェクトへのアクセス権の一覧が表示されます。

▼ ユーザーにデフォルトのアクセス権を与える方法

1. **Sun WBEM User Manager** を起動します。

2. ダイアログボックスの「ユーザーのアクセス権 **(Users Access)**」で、「追加 **(Add)**」をクリックします。

   使用できるネームスペースを示したダイアログボックスが開かれます。

3. テキスト入力フィールド「ユーザー名 **(User Name)**」に、**Solaris** ユーザーアカウント名を入力します。

4. 表示されたネームスペースの中からネームスペースを **1** つ選択します。

5. 「了解 **(OK)**」をクリックします。

   このユーザー名が「ユーザーマネージャ (User Manager)」ダイアログボックスに追加されます。

6. 「了解 **(OK)**」をクリックして変更を保存し、「ユーザーマネージャ **(User manager)**」ダイアログボックスを閉じます。または、「適用 **(Apply)**」をクリックして変更を保存し、ダイアログボックスを開いたままにしておきます。

   選択されたネームスペース内の CIM オブジェクトへの読み取り権がこのユーザーに与えられます。

▼ ユーザーアクセス権を変更する方法

1. **Sun WBEM User Manager** を起動します。

2. アクセス権を変更したいユーザーを選択します。

3. ユーザーに読み取り権だけを与えるには、「読み取り権 **(Read)**」チェックボックスをクリックします。ユーザーに書き込み権を与えるには、「書き込み権 **(Write)**」チェックボックスをクリックします。

4. 「了解 **(OK)**」をクリックして変更を保存し、「ユーザーマネージャ **(User manager)**」ダイアログボックスを閉じます。または、「適用 **(Apply)**」をクリックして変更を保存し、ダイアログボックスを開いたままにしておきます。

▼ ユーザーアクセス権を削除する方法

1. **Sun WBEM User Manager** を起動します。

2. ダイアログボックスの「ユーザーのアクセス権 **(Users Access)**」で、アクセス権を削除したいユーザー名を選択します。

3. ネームスペースに対するこのユーザーのアクセス権を削除するには「削除 **(Delete)**」をクリックします。

   ユーザーのアクセス権を削除してもよいか確認を求めるダイアログボックスが表示されます。「了解 (OK)」をクリックします。

4. 「了解 **(OK)**」をクリックして変更を保存し、「ユーザーマネージャ **(User manager)**」ダイアログボックスを閉じます。または、「適用 **(Apply)**」をクリックして変更を保存し、ダイアログボックスを開いたままにしておきます。

▼ ネームスペースのアクセス権を設定する方法

1. **Sun WBEM User Manager** を起動します。

2. ダイアログボックスの「ネームスペースのアクセス権 **(Namespace Access)**」で「追加 **(Add)**」をクリックします。

   使用できるネームスペースを示したダイアログボックスが開かれます。

3. アクセス権を設定したいネームスペースを選択します。

   デフォルトでは、ネームスペースに対する読み取り権だけがユーザーに与えられます。

   - ネームスペースに対してアクセスをまったく許可しない場合は、「読み取り権 (Read)」と「書き込み権 (Write)」チェックボックスを選択しないでください。
   - 書き込み権を与えるには、「書き込み権 (Write)」チェックボックスをクリックします。

- 読み取り権を与えるには、「読み取り権 (Read)」チェックボックスをクリックします。

4. 「了解 **(OK)**」をクリックして変更を保存し、「ユーザーマネージャ **(User manager)**」ダイアログボックスを閉じます。または、「適用 **(Apply)**」をクリックして変更を保存し、ダイアログボックスを開いたままにしておきます。

▼ ネームスペースのアクセス権を削除する方法

1. **Sun WBEM User Manager** を起動します。

2. ダイアログボックスの「ネームスペースのアクセス権 **(Namespace Access)**」で、アクセス権を削除したいネームスペースを選択し、「削除 **(Delete)**」をクリックします。

   ネームスペースからアクセス制御が削除され、「ユーザーマネージャ (User Manager)」ダイアログボックスのネームスペースリストからそのネームスペースが削除されます。

3. 「了解 **(OK)**」をクリックして変更を保存し、「ユーザーマネージャ **(User manager)**」ダイアログボックスを閉じます。または、「適用 **(Apply)**」をクリックして変更を保存し、ダイアログボックスを開いたままにしておきます。

API によるアクセス制御の設定

Sun WBEM SDK API を使用して、ネームスペースのアクセス制御または個々のユーザーのアクセス制御を設定できます。次のセキュリティクラスが `root\security` ネームスペースに格納されています。

- Solaris_Acl – Solaris Access Control Lists (ACL) の基底クラス。このクラスは、文字列プロパティ *capability* を定義し、そのデフォルト値を r (読み取り権) に設定する
- Solaris_UserAcl – 指定されたネームスペース内の CIM オブジェクトに対してユーザーが持つアクセス制御を表す
- Solaris_NamespaceAcl – ネームスペースに対するアクセス制御を表す

Solaris_UserACL クラスのインスタンスを作成し、続いて API を使用してそのインスタンスに対するアクセス権を変更することにより、ネームスペース内の CIM オブジェクトに対して個々のユーザーのアクセス制御を設定できます。同様に、Solaris_NameSpaceACL クラスのインスタンスを作成し、続いて setInstance メソッドのような API を使用してそのインスタンスへのアクセス権を設定することにより、ネームスペースのアクセス制御を設定できます。

これらの 2 つのクラスを効率よく使用するには、まず Solaris_NameSpaceACL クラスを使用してネームスペース内のオブジェクトに対し全ユーザーのアクセス権を制限します。続いて、Solaris_UserACL クラスを使用して、選択したユーザーにそのネームスペースに対するアクセス権を与えます。

注 - Access Control Lists (ACL) には、DMTF によって開発された標準が適用されます。Solaris ACL スキーマは現在 CIM に準拠していますが、DMTF が ACL 標準を最終的に決定する時点で変更の必要があります。このため Solaris ACL スキーマを使用して作成したプログラムでは、あとで変更の必要が生じる可能性があります。

Solaris_UserAclクラス

Solaris_UserAclクラスは、Solaris_Acl 基底クラスのサブクラスです。Solaris_UserAcl クラスは、この基底クラスからデフォルト値 r (読み取り権) を持つ文字列プロパティ *capability* を継承します。

capability プロパティには、次に示すアクセス権を設定できます。

アクセス権	説明
r	読み取り権
rw	読み取り権と書き込み権

アクセス権	説明
w	書き込み権
none	アクセス不可

Solaris_UserAcl クラスは、次の 2 つキープロパティを定義します。ネームスペースに存在できるのは、ネームスペースとユーザー名の ACL ペアのインスタンス 1 つだけです。

プロパティ	データ型	目的
nspace	文字列	この ACL が適用されるネームスペースを識別する
username	文字列	この ACL が適用されるユーザーを識別する

▼ ユーザーに対するアクセス制御を設定する方法

1. Solaris_UserAcl クラスのインスタンスを作成します。次に例を示します。

```
…
/* ネームスペースオブジェクトをローカルホストに作成し、root\security
   （ネームスペースの名前） で初期化する。 */

CIMNameSpace cns = new CIMNameSpace("", "root\security");

// root で root\security ネームスペースに接続する。
cc = new CIMClient(cns, "root", "root_password");

// Solaris_UserAcl クラスを取得
cimclass = cc.getClass(new CIMObjectPath("Solaris_UserAcl");

// Solaris_UserAcl クラスの新しいインスタンスを作成

ci = cimclass.newInstance();
…
```

2. ***capability*** プロパティに目的のアクセス権を設定します。次に例を示します。

```
…
/* root\molly ネームスペースのオブジェクトに対するユーザー Guest の
アクセス権 (capability) を rw (読み取り権と書き込み権) に変更 */

ci.setProperty("capability", new CIMValue(new String("rw"));
ci.setProperty("nspace", new CIMValue(new String("root\molly"));
ci.setProperty("username", new CIMValue(new String("guest"));
…
```

3. インスタンスを更新します。次に例を示します。

```
…
// 更新されたインスタンスを CIM Object Manager に渡す。
cc.setInstance(new CIMObjectPath(), ci);
…
```

Solaris_NamespaceAcl クラス

Solaris_NamespaceAcl は、Solaris_Acl 基底クラスを拡張したサブクラスです。Solaris_NamespaceAcl は、この基底クラスからデフォルト値 r (GUEST およびすべてのユーザーの読み取り権) を持つ文字列プロパティ *capability* を継承します。Solaris_NamespaceAcl クラスは、次の重要なプロパティを定義します。

プロパティ	データ型	目的
nspace	文字列	このアクセス制御リストが適用されるネームスペースを識別する。ネームスペースに存在できるのは、ネームスペース ACL のインスタンス 1 つだけである

▼ ネームスペースに対するアクセス制御を設定する方法

1. Solaris_namespaceAcl クラスのインスタンスを作成します。次に例を示します。

```
…
/* ネームスペースオブジェクトをローカルホストに作成し、root\security
    （ネームスペースの名前）で初期化する。 */

CIMNameSpace cns = new CIMNameSpace("", "root\security");

// root で root\security ネームスペースに接続する。
cc = new CIMClient(cns, "root", "root_password");

// Solaris_namespaceAcl クラスを取得
cimclass = cc.getClass(new CIMObjectPath("Solaris_namespaceAcl");

// Solaris_namespaceAcl クラスの新しいインスタンスを作成
ci = cimclass.newInstance();
…
```

2. ***capability*** プロパティに目的のアクセス権を設定します。次に例を示します。

```
…
/* root\molly ネームスペースに対するアクセス権 (capability) を
rw（読み取り権と書き込み権）に変更  */

ci.updatePropertyValue("capability",new CIMValue("rw"));
ci.updatePropertyValue("nspace",new CIMValue("root\molly"));
…
```

3. インスタンスを更新します。次に例を示します。

```
// 更新されたインスタンスを CIM Object Manager に渡す。
cc.setInstance(new CIMObjectPath(), ci);
```

例外メッセージ

例外メッセージの説明については、第 6 章を参照してください。

第 4 章

MOF コンパイラ

この章では、Managed Object Format (MOF) コンパイラについて説明します。内容は次のとおりです。

- MOF コンパイラについて
- `mofcomp` コマンド
- MOF ファイルのコンパイル

MOF コンパイラについて

Managed Object Format (MOF) コンパイラは、MOF を宣言しているファイルを解析し、ファイルに定義されているクラスやインスタンスを Java クラスに変換し、管理データを一元的に保存する CIM Object Manager Repository にそれを追加します。コンパイラは、MOF ファイルに `#pragma namespace(''`*namespace_path*`'')` が指定されていなければ、Java クラスをデフォルトのネームスペース `root\cimv2` に読み込みます。

MOF コンパイラを起動する `mofcomp` コマンドは 各 CIM スキーマのインストール前に実行され、CIM スキーマと Solaris スキーマを記述する MOF ファイルをコンパイルします。CIM スキーマは、どの管理環境にもある管理オブジェクトを表すためのクラス定義の集合です。Solaris スキーマは CIM スキーマを拡張したもので、一般的な Solaris オペレーティング環境の管理オブジェクトを表すクラス定義の集合です。

MOF は、CIM のクラスやインスタンスを定義する言語です。MOF ファイルは、MOF 言語を使って CIM オブジェクトを記述する ASCII テキストファイルです。

CIM オブジェクトは、プリンタ、ディスクドライブ、CPU などの管理リソースをコンピュータ用に表現したモデルです。

管理リソースの情報は MOF ファイルに格納されることがあります。MOF は Java に変換できるため、Java Virtual Machine をもつシステムで動作するアプリケーションならこの情報の解釈や交換を行うことができます。さらに、インストール後に、mofcomp コマンドを使って MOF ファイルをいつでもコンパイルできます。

mofcomp コマンド

mofcomp コマンドは、指定された MOF ファイルを CIM のクラスやインスタンスにコンパイルします。これらのクラスやインスタンスは Java クラスとして CIM Object Manager Repository に格納され、CIM Object Manager に渡されます。

mofcomp コマンドは、root として実行するか、コンパイルが行われているネームスペースに書き込み権をもつユーザーとして実行する必要があります。

/usr/sadm/bin/mofcomp [–h] [–v] [–sc] [–si] [–sq] [–version] [–c *cimom_hostname*] [–p *password*] [–u *username*] *file*

–help	mofcomp コマンドに対する引数を一覧表示します。
–c [***cimom-host***]	CIM Object Manager が動作するシステムを指定します。
–p [***password***]	CIM Object Manager に接続するためのパスワードを指定します。このオプションは、コンパイルで CIM Object Manager への特権アクセスが必要なときに指定します。–p と –u を両方とも指定するとコマンド行にパスワードを入力する必要があるため、セキュリティリスクを伴います。パスワードをより安全に入力したい場合は、–u だけを指定し、–p を指定しなければ、コンパイラがパスワードのプロンプトを表示します。
–sc	クラスの設定 (set class) オプションを使ってコンパイラを実行します。このオプションは、指定されたクラスが存在し、それにインスタンスがないとそのクラスを更新し、クラスが存在していないとエラーを返します。このオプションを指定しないと、コンパイラは、接続されているネームスペースに CIM クラスを追加し、そのクラスがすでに存在していればエラーを返します。

`–si`	インスタンスの設定 (set instance) オプションを使ってコンパイラを実行します。このオプションは、指定されたインスタンスが存在しているとそのインスタンスを更新し、存在していないとエラーを返します。このオプションを指定しないと、コンパイラは、接続されているネームスペースに CIM インスタンスを追加し、そのインスタンスがすでに存在していればエラーを返します。
`–sq`	修飾子タイプ (set qualifier types) オプションを使ってコンパイラを実行します。このオプションは、指定された修飾子タイプが存在しているとその修飾子タイプを更新し、存在していないとエラーを返します。このオプションを指定しないと、コンパイラは、接続されているネームスペースに CIM 修飾子タイプを追加し、その修飾子タイプがすでに存在していればエラーを返します。
`–u`	CIM Object Manager に接続するためのユーザー名を指定します。このオプションは、コンパイルで CIM Object Manager への特権アクセスが必要なときに指定します。–p と –u を両方とも指定するとコマンド行にパスワードを入力する必要があるため、セキュリティリスクを伴います。パスワードをより安全に入力したい場合は、–u だけを指定し、–p を指定しなければ、コンパイラがパスワードのプロンプトを表示します。
`–v`	コンパイラを詳細モードで実行します。コンパイラメッセージが表示されます。
`–version`	MOF コンパイラのバージョンを表示します。

`mofcomp` コマンドは、正常に終了した場合は `0`、失敗した場合は正の整数を返します。

MOF ファイルのコンパイル

コンパイルする MOF ファイルには、`.mof` 拡張子が付いていても付いていなくてもかまいません。CIM スキーマや Solaris スキーマを記述する MOF ファイルは `/usr/sadm/mof` にあります。

▼ MOF ファイルをコンパイルする方法

1. パラメータを指定せずに **MOF** コンパイラ を実行するには、次のコマンドを入力します。

```
% mofcomp filename
```

たとえば、mofcomp /usr/sadm/mof/Solaris_Schema1.0.mof と入力すると、指定した MOF ファイルがコンパイルされます。

例

次の例は、Solaris_System1.0.mof ファイルの例で、Solaris オペレーティング環境でのプロセス、オペレーティングシステム、ファイルシステムなどの管理リソースを表しています。

例 **4–1** Solaris_System1.0 ファイル

```
// ===================================================================
// Title:       Solaris System MOF specification 1.0
// Filename:    Solaris_System1.0.mof
// Version:     1.0
// Author:      Sun Microsystems, Inc.
// Date:        02/01/1999
// Description:
// ===================================================================

// ===================================================================
// Pragmas
// ===================================================================
#pragma Locale ("en-US")

// ===================================================================
// Solaris_Process
// ===================================================================

 [Provider("com.sun.wbem.solarisprovider.Solaris"),
  Description ("A Solaris process that is running.")
 ]
class Solaris_Process:CIM_Process
{
     [Description (
        "Time in user mode and kernel mode, in milliseconds."
        "If this information is not available, a value of 0 should be used."),
       Units("MilliSeconds")
      ]
```

(続く)

```
   uint64 UserKernelModeTime;
      [Description (
          "A string used to identify the Parent Process. A Process ID is a "
          "kind of Process Handle."),
       MaxLen (256)
      ]
   string ParentHandle;
};
:
:
// ===================================================================
// Solaris_FileSystem
// ===================================================================

 [Provider("com.sun.wbem.solarisprovider.Solaris"),
  Description ("A Solaris FileSystem.")
 ]
class Solaris_FileSystem:CIM_FileSystem
{
};
```

次の例は、デフォルトオプションを使って Solaris_System1.0.mof ファイルを再コンパイルしたものです。このオプションは、コンパイラが、すでにあるクラスを追加しようとするたびにエラーを返します。この例では、詳細モード (–v) を使用しています。さらに、デフォルトのネームスペース root\cimv2 への書き込み権が必要なため、スーパーユーザーアカウントを指定します。Solaris_System1.0.mof ファイルに定義されている CIM 要素は、デフォルトで root\cimv2 ネームスペースに追加されます。

例 **4–2**　デフォルトオプションを使用した MOF ファイルのコンパイル

```
mofcomp -v -u root ../mof/Solaris_Schema1.0.mof
Starting MOF Compiler
java com.sun.wbem.compiler.mofc.CIM_Mofc -v -u root ../mof/Solaris_Schema1.0.mof
Enter password:******
Adding class Solaris_ComputerSystem
Warning at line 27 in file /usr/sadm/mof/Solaris_Core1.0.mof
- compilation proceeding ...
Semantic Error:
The following exception was thrown by setClass:
  CIM_ERR_ALREADY_EXISTS:Element Solaris_ComputerSystem already exists.
Adding class Solaris_SerialPortSetting
Warning at line 39 in file /usr/sadm/mof/Solaris_Core1.0.mof
```

(続く)

```
- compilation proceeding ...
Semantic Error:
The following exception was thrown by setClass:
  CIM_ERR_ALREADY_EXISTS:Element Solaris_SerialPortSetting already exists.
Adding class Solaris_LogRecord
Warning at line 104 in file /usr/sadm/mof/Solaris_Core1.0.mof
- compilation proceeding ...
⋮
Semantic Error:
The following exception was thrown by setClass:
  CIM_ERR_ALREADY_EXISTS:Element Solaris_NamespaceAcl already exists.
Adding class Solaris_UserAcl
Warning at line 39 in file /usr/sadm/mof/Solaris_Acl1.0.mof
- compilation proceeding ...
Semantic Error:
The following exception was thrown by setClass:
  CIM_ERR_ALREADY_EXISTS:Element Solaris_UserAcl already exists.
Compilation succeeded.
```

セキュリティ上の注意

-p パラメータを使ってコマンドをパスワードを指定して実行すると、指定したパスワードを他のユーザーが `ps` コマンドや `history` コマンドを使って知ることができます。

注 - 実行するコマンドにパスワードを指定する場合は、コマンドを実行した後でただちにパスワードを変更してください。

例 4–3 セキュリティ上安全でない構文の例

次の例は、*-p* パラメータを使って `mofcomp` コマンドを実行する例です。

```
mofcomp -p Log8Rif
```

次の例は、*-u* および *-p* パラメータを使って `mofcomp` コマンドを実行する例です。

```
mofcomp -u molly -p Log8Rif
```

パスワードを指定するオプションを使って modcomp コマンドを実行したら、ただちにパスワードを変更してください。

第 5 章

イベントのロギング

ロギングは、WBEM 管理者がイベントを追跡して発生原因を調べることができるためのサービスです。イベントとは、たとえば、シリアルポートが提供されているのにそれにアクセスできない場合や、ファイルシステムのマウントでエラーメッセージが生成された場合、システムディスクが容量の限界に達した場合などのことです。

この章で取り上げる内容は次のとおりです。

- ロギングについて
- ログファイル
- ログクラス
- API によるロギングの有効化
- ログデータの表示

ロギングについて

ロギングサービスは、サービスプロバイダが返すことになっているすべてのアクションや、Solaris WBEM Services のコンポーネントが完了したすべてのアクションを記録します。 このサービスでは、通知用のメッセージとエラーメッセージがログに記録されます。たとえば、ユーザーがシリアルポートを無効にすると、シリアルポートプロバイダによってこの情報が自動的に記録されます。システムエラーなどの障害が発生した場合には、WBEM 管理者はログ記録を参照して発生原因を調べることができます。

コンポーネント、アプリケーション、およびプロバイダはすべて、イベントに応答してロギングを自動的に開始します。たとえば、CIM Object Manager は、インストールと起動が終わると自動的にイベントを記録します。

WBEM 環境向けに開発されるアプリケーションとプロバイダには、ロギングを設定できます。詳細は、62ページの「API によるロギングの有効化」を参照してください。ログデータは、ログビューアで表示してアプリケーション用に開発するロギング機能のデバッグを行うこともできます。

ログファイル

アプリケーションまたはプロバイダがイベントを記録するように設定すると、イベントはログファイルに書き込まれます。ログ記録はすべて、パス `/usr/sadm/wbem/log` に格納されます。ログファイルは、次の命名規則を使用します。

```
wbem_log.#
```

`#` は、ログファイルのバージョンを示すために付加される番号です。`.1` が付いたログファイル (例: `wbem_log.1`) は、最も新しいバージョンです。`.2` が付いたログファイルは、その次に新しいバージョンです。拡張子が大きいファイル (例: `wbem_log.16`) は、バージョンが古いことを示します。最も新しいバージョンのファイル、それ以前のバージョンのファイルとも、`/usr/sadm/wbem/log` 内に 1 つのアーカイブとして同時に存在します。

次の状況の 1 つが発生する場合、ファイル拡張子 `.1` によるログファイルの名前変更および、保存が行われます。

- 現在のファイルが、`Solaris_LogServiceProperties` クラスによって指定されたファイルサイズ限度に達した。デフォルト値は `wbemService.properties` ファイルに設定されている

  `Solaris_LogServiceProperties` クラスのプロパティによるログファイルの使用方法の制御については、59ページの「ログファイルの規則」を参照してください。

- `Solaris_LogService` クラスの `clearLog()` メソッドが、現在のログファイルに対して呼び出された

  `Solaris_LogService` クラスとそのメソッドの詳細は、61ページの「Solaris_LogService」を参照してください。

ログファイルの規則

Solaris_LogServiceProperties クラスは、Solaris_Core1.0.mof で定義されています。Solaris_LogServiceProperties クラスには、次のログファイル属性を制御するプロパティがあります。

- ログファイルが記録されるディレクトリ
- ログファイル名
- ログファイルに許可されるサイズ。このサイズに達すると、/usr/sadm/wbem/log でファイル拡張子 .1 によるログファイルの名前変更、保存、およびアーカイブが行われる
- アーカイブ内に格納できるログファイルの数
- SysLog (Solaris オペレーティング環境のデフォルトのロギングシステム) にログデータを書き込む機能

データをログファイルに書き込むアプリケーションにこれらの属性のどれかを指定するには、Solaris_LogServiceProperties の新しいインスタンスを作成し、その関連プロパティの値を設定します。新しいインスタンスのプロパティ値を設定する方法については、例 5–14 を参照してください。

ログファイルの形式

ロギングサービスは、一般的なログ記録として、アプリケーションログ、システムログ、およびセキュリティログの 3 種類を提供しています。ログ記録は、通知用のメッセージのことも、エラーまたは警告を記述したデータのこともあります。ログに表示できる標準のフィールドがデータに定義されていますが、ログはすべてのフィールドを使用するとは限りません。たとえば、通知用のログではイベントについて説明した簡単なメッセージが示され、エラーログでは詳細なメッセージが示される場合があります。

ログデータフィールドの中には、CIM Repository 内のデータを識別するものがあります。これらのフィールドは、Solaris_LogRecord クラス内の読み取り専用のキー修飾子が付けられたプロパティです。これらのフィールドの値は設定できません。ログファイル内の次のフィールドの値は設定できます。

- Category – ログ記録の種類
- Severity – ログファイルに記録されたデータの原因となった状況の重大度
- AppName – データを出したアプリケーションの名前
- UserName – ログデータが生成された時にそのアプリケーションを使用していたユーザーの名前
- ClientMachineName – ログデータを生成した状況が起きたコンピュータの名前
- ServerMachineName – ログデータを生成した状況が起きたサーバーの名前
- SummaryMessage – 状況を説明する簡単なメッセージ
- DetailedMessage – 状況を説明する詳しいメッセージ
- data – ログメッセージの解釈のためにアプリケーションとプロバイダが提示できる背景情報

ログクラス

ロギングには、2 つの Solaris スキーマクラス、`Solaris_LogRecord` と `Solaris_LogService` が使用されます。

Solaris_LogRecord

`Solaris_LogRecord` は、`Solaris_Core1.0.mof` でログファイル内のエントリをモデル化するように定義されています。イベントに応答してアプリケーションまたはプロバイダが `Solaris_LogRecord` クラスを呼び出す場合、`Solaris_LogRecord` クラスはそのイベントによって生成されるデータをすべてログファイルに書き込みます。`Solaris_LogRecord` クラスの定義を Solaris プロバイダの一部として見るには、任意のテキストエディタで `Solaris_Core1.0.mof` ファイルを表示してください。`Solaris_Core1.0.mof` ファイルは、`/usr/sadm/mof` に入っています。

`Solaris_LogRecord` は、プロパティのベクトルとキー修飾子を使用して、データを生成したイベント、システム、ユーザー、およびアプリケーションまたはプロバイダの属性を指定します。アプリケーションと CIM Repository 間で使用されるように、読み取り専用修飾子の値が透過的に生成されます。たとえば、値

RecordID はログエントリを個別に識別しますが、生成されたデータが表示される場合にログ書式の一部としては示されません。

書き込み可能修飾子の値は設定できます。たとえば、イベントが発生したシステムを識別する ClientMachineName と ServerMachineName のようなプロパティの修飾子の値を設定できます。

SysLogFlag プロパティが真に設定されていると、ログレコードの詳しいメッセージが UNIX システムの syslog デーモンに自動的に送信されます。

Solaris_LogService

Solaris_LogService クラスは、ロギングサービスのオペレーションの制御と、ログデータの処理方法の定義を行います。このクラスには、アプリケーションが特定のイベントについてのデータを発行元アプリケーションから CIM Object Manager へ送信するために使用できる一連のメソッドが含まれます。データは、CIM Object Manager からの応答 (CIM Repository からのデータ検出など) を生成するトリガーとなります

Solaris_LogService クラスは、次のメソッドを使用します。

- clearLog – 現在のログファイルの名前変更、保存、または保存されたログファイルの削除を行う
- getNumRecords – 特定のログファイルに記録されているログの数を返す
- listLogFiles – /usr/sadm/wbem/log に格納されているすべてのログファイルの一覧を返す
- getCurrentLogFileName – 最新のログファイルの名前を返す
- getNumLogFiles – /usr/sadm/wbem/log に格納されているログファイルの数を返す
- getLogFileSize – 特定のログファイルのサイズをメガバイト単位で返す
- getSyslogSwitch – ログデータを SysLog (Solaris オペレーティング環境のロギングサービス) に送る
- getLogStorageName – ログファイルが格納されるホストコンピュータまたはデバイスの名前を返す
- getLogFileDir – ログファイルが格納されるディレクトリのパスと名前を返す
- setProperties – ロギングプロパティを設定する

Solaris_Core1.0.mof ファイル内の Solaris_LogService の定義は、任意のテキストエディタでこのファイルを開いて表示できます。Solaris_Core1.0.mof ファイルは、/usr/sadm/mof に入っています。

API によるロギングの有効化

ログファイルの内容は現在 Log Viewer で表示できますが、独自の方法でログファイルを表示したい場合はロギング API を使用して Log Viewer を作成できます。この API を使用すると、次のことができます。

- アプリケーションからログファイルへデータを書き込む
- ログファイルから Log Viewer へデータを読み込む
- ロギングをどのように扱うかを指定するロギングプロパティを設定する

ログファイルへのデータの書き込み

アプリケーションでデータをログファイルに書き込むには、次の作業を行います。

- Solaris_LogRecord クラスの新しいインスタンスを作成する
- ログファイルに書き込まれるプロパティを設定し、プロパティの修飾子の値を設定する
- 出力する新しいインスタンスとプロパティを設定する

▼ データを書き込むために Solaris_LogRecord のインスタンスを作成する方法

1. 必要なすべての **java** クラスをインポートします。例 5–1 に示すクラスは必要な最小限のクラスです。

例 **5–1** クラスのインポート

```
import java.rmi.*;
import com.sun.wbem.client.CIMClient;
import com.sun.wbem.cim.CIMInstance;
import com.sun.wbem.cim.CIMValue;
import com.sun.wbem.cim.CIMProperty;
import com.sun.wbem.cim.CIMNameSpace;
import com.sun.wbem.cim.CIMObjectPath;
import com.sun.wbem.cim.CIMClass;
import com.sun.wbem.cim.CIMException;
import com.sun.wbem.solarisprovider.*;
import java.util.*;
```

2. 公開クラス `CreateLog` と次の値を宣言します。

- `CIMClient` の値
- `CIMObjectPath` の値
- `CIMNameSpace` の値

例 **5–2** `CreateLog` クラスと値の宣言

```
public class CreateLog {
    public static void main(String args[]) throws CIMException {

 if ( args.length != 3) {
     System.out.println("Usage: CreateLog host username password");
     System.exit(1);
 }

 CIMClient cc = null;
 CIMObjectPath cop = null;
 try {
     CIMNameSpace cns = new CIMNameSpace(args[0]);
     cc = new CIMClient(cns, args[1], args[2]);
```

3. プロパティのベクトルが返されるように指定します。修飾子のプロパティに値を設定します。

例 **5–3**　プロパティのベクトルと値の指定

```
Vector keys = new Vector();
CIMProperty logsvcKey;
logsvcKey = new CIMProperty("category");
logsvcKey.setValue(new CIMValue(new Integer(2)));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("severity");
logsvcKey.setValue(new CIMValue(new Integer(2)));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("AppName");
logsvcKey.setValue(new CIMValue("SomeApp"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("UserName");
logsvcKey.setValue(new CIMValue("molly"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("ClientMachineName");
logsvcKey.setValue(new CIMValue("dragonfly"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("ServerMachineName");
logsvcKey.setValue(new CIMValue("spider"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("SummaryMessage");
logsvcKey.setValue(new CIMValue("brief_description"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("DetailedMessage");
logsvcKey.setValue(new CIMValue("detailed_description"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("data");
logsvcKey.setValue(new CIMValue("0xfe 0x45 0xae 0xda"));
keys.addElement(logsvcKey);
logsvcKey = new CIMProperty("SyslogFlag");
logsvcKey.setValue(new CIMValue(new Boolean(true)));
keys.addElement(logsvcKey);
```

4. ログ記録用に `CIMObjectPath` の新しいインスタンスを宣言します。

例 **5–4**　`CIMObjectPath` の新しいインスタンスの宣言

```
CIMObjectPath logreccop = new CIMObjectPath("Solaris_LogRecord", keys);
```

5. `Solaris_LogRecord` の新しいインスタンスを宣言します。ファイルに書き込むようにプロパティのベクトルを設定します。

例 **5–5**　インスタンスとプロパティの設定

```
CIMInstance ci = new CIMInstance();
  ci.setClassName("Solaris_LogRecord");
  ci.setProperties(keys);
  //System.out.println(ci.toString());
  cc.setInstance(logreccop,ci);
 }
 catch (Exception e) {
     System.out.println("Exception: "+e);
  e.printStackTrace();
 }
```

6. データがログファイルに書きこまれたあと、セッションを閉じます

例 **5–6**　セッションを閉じる

```
 // セッションを閉じる。
 if(cc != null) {
     cc.close();
 }
    }
}
```

ログファイルからのデータの読み取り

アプリケーションでログファイルから Log Viewer にデータを読み込むには、次の作業を行います

- `Solaris_LogRecord` クラスのインスタンスを列挙する
- 目的のインスタンスを取得する
- 出力デバイス (通常はユーザーインタフェース) にインスタンスのプロパティを出力する

▼ Solaris_LogRecord のインスタンスを取得し、データを読み取る方法

1. 必要なすべての **java** クラスをインポートします。例 5–7 に示すクラスは、インポートする必要がある最小限のクラスです。

例 **5–7** クラスのインポート

```
import java.rmi.*;
import com.sun.wbem.client.CIMClient;
import com.sun.wbem.cim.CIMInstance;
import com.sun.wbem.cim.CIMValue;
import com.sun.wbem.cim.CIMProperty;
import com.sun.wbem.cim.CIMNameSpace;
import com.sun.wbem.cim.CIMObjectPath;
import com.sun.wbem.cim.CIMClass;
import com.sun.wbem.cim.CIMException;
import com.sun.wbem.solarisprovider.*;
import java.util.*;
import java.util.Enumeration;
```

2. クラス `ReadLog` を宣言します。

例 **5–8** `ReadLog` クラスの宣言

```
public class ReadLog
    {
    public static void main(String args[]) throws
    CIMException
    {
    if ( args.length != 3)
    {
    System.out.println("Usage: ReadLog host username
    password");
     System.exit(1);
```

3. `ReadLog` クラスのクライアント、オブジェクトパス、およびネームスペースの値を設定します。

例 **5–9**　Solaris ログ記録の作成

```
 }
CIMClient cc = null;
 CIMObjectPath cop = null;
try {      CIMNameSpace cns = new CIMNameSpace(args[0]);
cc = new CIMClient(cns, args[1], args[2]);
cop = new CIMObjectPath("Solaris_LogRecord");
```

4. Solaris_LogRecord のインスタンスを列挙します。

例 **5–10**　インスタンスの列挙

```
 Enumeration e = cc.enumInstances(cop, true);
  for (; e.hasMoreElements(); ) {
```

5. プロパティの値を出力デバイスに送ります。

例 **5–11**　プロパティ値の送信

```
System.out.println("------------------------
---------");
   CIMObjectPath op = (CIMObjectPath)e.nextElement();
   CIMInstance ci = cc.getInstance(op);
   System.out.println("Record ID : " +
      (((Long)ci.getProperty("RecordID").getValue().
      getValue()).longValue()));
   System.out.println("Log filename : " +
      ((String)ci.getProperty("FileName").getValue().
      getValue()));
   int categ = (((Integer)ci.getProperty("category").
      getValue().getValue()).intValue());
   if (categ == 0)
    System.out.println("Category : Application Log");
   else if (categ == 1)
    System.out.println("Category : Security Log");
   else if (categ == 2)
    System.out.println("Category : System Log");
   int severity = (((Integer)ci.getProperty
      ("severity").getValue().getValue()).intValue());
   if (severity == 0)
```

(続く)

```
 System.out.println("Severity : Informational");
else if (severity == 1)
 System.out.println("Severity : Warning Log!");
else if (severity == 2)
 System.out.println("Severity : Error!!");
System.out.println("Log Record written by :" +
   ((String)ci.getProperty("AppName").getValue().
   getValue()));
System.out.println("User : " + ((String)ci.
   getProperty("UserName").getValue().getValue()));
System.out.println("Client Machine : " + ((String)ci.
   getProperty("ClientMachineName").getValue().getValue
   ()));
System.out.println("Server Machine : " + ((String)ci.
   getProperty("ServerMachineName").getValue().getValue
   ()));
System.out.println("Summary Message : " + ((String)
   ci.getProperty("SummaryMessage").getValue().getValue
   ()));
System.out.println("Detailed Message : " + ((String)
   ci.getProperty("DetailedMessage").getValue().getValue
   ()));
System.out.println("Additional data : " + ((String)
   ci.getProperty("data").getValue().getValue()));
boolean syslogflag =
 ((Boolean)ci.getProperty("syslogflag").getValue().
   getValue()).booleanValue();
  if (syslogflag == true) {
System.out.println("Record was written to syslog as
   well");
} else {
System.out.println("Record was not written to
   syslog");
}
System.out.println("-----------------------------
   ----");

}
```

6. エラーが発生した場合は、ユーザーにエラーメッセージを返します。

例 5–12　エラーメッセージを返す

```
…
}
 catch (Exception e) {
     System.out.println("Exception: "+e);
  e.printStackTrace();
 }
…
```

7. ファイルからデータが読み取られたあとセッションを閉じます。

例 5–13　セッションを閉じる

```
 // セッションを閉じる。
 if(cc != null) {
     cc.close();
 }
}
}
```

ロギングプロパティの設定

Solaris_LogServiceProperties クラスのインスタンスを作成し、そのプロパティ値を設定すれば、アプリケーションやプロバイダでロギングをどのように扱うかを制御できます。次の例は、ロギングプロパティの設定方法を示したものです。プロパティは、/usr/sadm/bin/wbem/wbemservices.properties ファイルに格納されます。

例 **5–14**　ロギングプロパティの設定

```
public class SetProps {
    public static void main(String args[]) throws CIMException {

 if ( args.length != 3) {
     System.out.println("Usage: SetProps host username password");
     System.exit(1);
 }

 CIMClient cc = null;
 try {
     CIMNameSpace cns = new CIMNameSpace(args[0]);
     cc = new CIMClient(cns, args[1], args[2]);

   CIMObjectPath logpropcop = new CIMObjectPath("Solaris_Log
   ServiceProperties");
     Enumeration e = cc.enumInstances(logpropcop, true);
  for (; e.hasMoreElements(); ) {
   CIMObjectPath op = (CIMObjectPath)e.nextElement();
   CIMInstance ci = cc.getInstance(op);
   ci.setProperty("Directory", new CIMValue("/tmp/bar1/"));
   ci.setProperty("FileSize", new CIMValue("10"));
   ci.setProperty("NumFiles", new CIMValue("2"));
   ci.setProperty("SyslogSwitch", new CIMValue("off"));
   cc.setInstance(logpropcop,ci);
  }
 }
 catch (Exception e) {
     System.out.println("Exception: "+e);
  e.printStackTrace();
 }

 // セッションを閉じる。
 if(cc != null) {
     cc.close();
 }
}
```

ログデータの表示

Log Viewer を使えば、ログレコードをすべて詳細に表示できます。記録されたデータは、Log Viewer のグラフィカルユーザーインタフェースを使って表示します。Log Viewer は、Solaris WBEM Services のインストール時に `/usr/sadm/bin` にイ

ンストールされます。Log Viewer を実行するには、ログレコードがあらかじめ作成されている必要があります。

Log Viewer の起動

ログレコードを作成したら Log Viewer を起動します。

▼ Log Viewer を起動する方法

1. 次のコマンドを入力して **Log Viewer** のディレクトリに移動します。

```
% cd /usr/sadm/bin
```

2. 次のコマンドを入力して **Log Viewer** を起動します。

```
% ./wbemlogviewer
```

第 6 章

CIM 例外メッセージ

この章では、Solaris WBEM Services の CIM Object Manager が生成する例外メッセージについて説明します。取り上げる内容は次のとおりです。

- CIM 例外の生成
- CIM 例外の構成
- CIM 例外情報の検索
- 生成される CIM 例外

CIM 例外の生成

CIM Object Manager が、すべてのクライアントで使用される例外メッセージを生成します。MOF コンパイラにより、`.mof` ファイルのどこでそのエラーが発生したかを示す行がそのあとに追加されます。クライアントアプリケーションは、これらの例外メッセージから、エンドユーザーにとって理解しやすいエラーメッセージを生成できます。

CIM クライアントは、XML クライアントか RMI クライアントとして使用されます。現在、XML はこれらの例外のサブセットだけをサポートします。XML クライアントを使用する場合は、例外メッセージに含まれるすべての情報を受信できるとは限らないことや、パラメータ情報が含まれていないことがあるので注意してください。

CIM 例外の構成

CIM 例外は、次の要素から構成されます。

- 固有の識別子 – そのエラーメッセージをほかのエラーメッセージと区別するための文字列
- 1つまたは複数のパラメータ – 例外メッセージに示される特定のクラス、メソッド、および修飾子の可変部分

例外メッセージの例

MOF コンパイラは、たとえば次のような例外メッセージを返します。

```
REF_REQUIRED
 CIM_Docked
```

- REF_REQUIRED は固有の識別子
- CIM_Docked はパラメータ。パラメータは、該当するクラス、プロパティ、メソッド、または修飾子の名前に置換される

この例外メッセージは、ユーザーにとって理解しやすい次のようなメッセージに変換できます。

```
REF_REQUIRED
 = Association class CIM_Docked needs
 at least two refs. Error in line 12.
```

開発者向け: エラーメッセージテンプレート

WBEM は発生し得るすべてのエラーメッセージを例外テンプレート (CIMError_ja.properties ファイル) として提供しています。パラメータが必要な例外テンプレートでは、最初のパラメータは {0}、2 つめのパラメータは {1} として示されています。

前述の例では、次の例外テンプレートが使用されています。

```
REF_REQUIRED = Association class {0} needs at least two refs.
```

CIM 例外情報の検索

次の節では、CIM 例外について詳しく説明しています。これらの例外は、固有の識別子でアルファベット順に記述されています。各例外メッセージごとに、次の中から該当する情報を示します。

- 固有の識別子: ヘッダーとして表示される
- 説明: 例外メッセージ内に使用されているパラメータの説明
- 例: ユーザーに表示される例外やメッセージの例。通常は、MOF コンパイラや CIM Object Manager の出力
- 原因: その例外メッセージが生成された理由と、メッセージの理解に役立つ背景 (参照) 情報を示す
- 解決方法: そのエラーを解決する方法がある場合、その手順などが示される

生成される CIM 例外

この節では、MOF コンパイラ、CIM Object Manager、および WBEM クライアントアプリケーションが生成する CIM 例外について説明します。

`ABSTRACT_INSTANCE`

説明

`ABSTRACT_INSTANCE` の例外は 1 つのパラメータを使用しますが、このパラメータは abstract クラスの名前です。

例

`ABSTRACT_INSTANCE` = Abstract class `ExampleClass` cannot have instances.

原因

指定されたクラスにインスタンスを作成しようとしましたが、このクラスは abstract クラスです。abstract クラスは、インスタンスを持てません。

解決方法

クライアントアプリケーションがそのようなインスタンスを作成することはできないので、プログラムで指定しているインスタンスを削除します。

`CHECKSUM_ERROR`

説明

`CHECKSUM_ERROR` 例外メッセージは、パラメータを使用しません。

例

`CHECKSUM_ERROR` = Checksum not valid.

原因

メッセージは、壊れているため送信できませんでした。この損傷は、送信中に偶然に生じたか、あるいは第三者によって故意に壊された可能性があります。

注 - このエラーメッセージは、CIM Object Manager が無効なチェックサムを受け取る場合に表示されます。チェックサムは、ネットワーク上で転送されるデータパケットのビット数です。この数は、伝送が安全であり、かつ送信中にデータの破損や意図的な変更がなかったことを情報の送信側と受信側が確認するために使用されます。

送信前に、データに対してアルゴリズムが実行されます。この実行により生成されたチェックサムがデータに含まれ、データパケットのサイズを示します。メッセージを受信すると、受信側はチェックサムを再計算し、送信側のチェックサムと比較できます。チェックサムが一致すれば、送信は安全に行われ、データの破損や変更が起きなかったと言えます。

解決方法

再送信します。

`CIM_ERR_ACCESS_DENIED`

説明

`CIM_ERR_ACCESS_DENIED` 例外メッセージは、パラメータを使用しません。

例

`CIM_ERR_ACCESS_DENIED` = Insufficient privileges.

原因

この例外メッセージは、アクションを実行するための適切な特権がユーザーにない場合に表示されます。

解決方法

WBEM の管理者に、処理を行うための特権を要求します。

```
CIM_ERR_ALREADY_EXISTS
```

例 *1: CIM_ERR_ALREADY_EXISTS*

説明

この場合の `CIM_ERR_ALREADY_EXISTS` 例外メッセージは 1 つのパラメータを使用しますが、このパラメータは重複した要素の名前に置換されています。

例

`CIM_ERR_ALREADY_EXISTS` = Duplicate class `CIM_Rack`

原因

作成しようとした要素に、既存の要素と同じ名前が使用されています。

解決方法

CIM WorkShop で既存の要素を検索して使用されている名前を確認し、固有の名前を使用して要素を作成します

例 *2: CIM_ERR_ALREADY_EXISTS*

説明

この場合の `CIM_ERR_ALREADY_EXISTS` 例外メッセージは 1 つのパラメータを使用しますが、このパラメータは重複したインスタンスの名前に置換されています。

例

`CIM_ERR_ALREADY_EXISTS` = Duplicate instance `SolarisRack`

原因

作成しようとしたクラスのインスタンスに、既存のインスタンスと同じ名前が使用されています。

解決方法

CIM WorkShop で既存のインスタンスを検索して使用されている名前を確認し、固有の名前を使用してインスタンスを作成します。

例 *3: CIM_ERR_ALREADY_EXISTS*

説明

この場合の CIM_ERR_ALREADY_EXISTS エラーメッセージは 1 つのパラメータを使用しますが、このパラメータは重複したネームスペースの名前に置換されています。

例

CIM_ERR_ALREADY_EXISTS = Duplicate namespace root\cimv2

原因

作成が試みられたネームスペースに、既存のネームスペースと同じ名前が使用されています。

解決方法

既存のネームスペースを検索して使用されている名前を確認し、固有の名前を使用してネームスペースを作成します。

例 *4: CIM_ERR_ALREADY_EXISTS*

説明

この場合の CIM_ERR_ALREADY_EXISTS エラーメッセージは 1 つのパラメータを使用しますが、このパラメータは重複した修飾子型の名前に置換されています。

例

CIM_ERR_ALREADY_EXISTS = Duplicate qualifier type Key

原因

作成しようとした修飾子型に、変更されるプロパティの既存の修飾子型と同じ名前が使用されています。

解決方法

CIM WorkShop でプロパティの既存の修飾子型を検索して使用されている名前を確認し、固有の名前を使用して修飾子型を作成します。

CIM_ERR_FAILED

説明

CIM_ERR_FAILED 例外メッセージは 1 つのパラメータを使用しますが、このパラメータは文字列 (エラー状態と推測される原因を説明したメッセージ) に置換されています。

例

CIM_ERR_FAILED=Invalid entry.

原因

CIM_ERR_FAILED 例外メッセージは、さまざまなエラー状況に対して表示される一般的なメッセージです。

解決方法

CIM_ERR_FAILED は一般的なエラーメッセージであるため、このメッセージの原因となり得る状況は多数考えられます。解決方法は、エラー状況によって異なります。

CIM_ERR_INVALID_PARAMETER

説明

CIM_ERR_INVALID_PARAMETER 例外は、エラーを引き起こしたパラメータの詳しい情報を示す 1 つのパラメータを使用します。

例

CIM_ERR_INVALID_PARAMETER = Class System has no schema prefix.

原因

操作が行われたが、パラメータが無効でした。たとえばクラス名の前にスキーマ接頭辞のないクラスが作成されました。Common Information Model では、すべてのクラス名にスキーマ接頭辞を付ける必要があります。たとえば、CIM スキーマの一部として作成されるクラスには、CIM 接頭辞 CIM を付けます。Solaris スキーマの一部として作成されたクラスには、Solaris 接頭辞 Solaris を付けます。

解決方法

正しいパラメータを指定してください。上の例の場合、正しいパラメータは、たとえば、CIM です。接頭辞のないクラスのインスタンスをすべて見つけ、クラス名と接頭辞に置き換えます。

CIM_ERR_INVALID_SUPERCLASS

説明

メッセージ CIM_ERR_INVALID_SUPERCLASS は、次の 2 つのパラメータを使用します。

- 指定されたスーパークラスの名前

■ エラーを引き起こしたサブクラスの名前

例

CIM_ERR_INVALID_SUPERCLASS = Superclass CIM_Chassis for class CIM_Container does not exist.

原因

特定のスーパークラスに属するクラスが指定されましたが、そのスーパークラスは存在しません。指定されたスーパークラスにスペルミスがあるか、あるいは意図したスーパークラス名の代わりに誤って存在しないスーパークラス名が指定されたことが考えられます。また、そのスーパークラスとサブクラスに変更があった可能性もあります。たとえば、指定されたスーパークラスは、実際は指定されたクラスのサブクラスであるかもしれません。この例では、CIM_Container のスーパークラスとして CIM_Chassis が指定されていますが、これは逆で、CIM_Chassis は CIM_Container のサブクラスです。

解決方法

スーパークラスのスペルと名前が正しいか確認し、ネームスペース内にそのスーパークラスが存在することを確認します。

CIM_ERR_NOT_FOUND

例 *1: CIM_ERR_NOT_FOUND*

説明

CIM_ERR_NOT_FOUND 例外メッセージは 1 つのパラメータを使用しますが、このパラメータは存在しないクラスの名前に置換されています。

例

CIM_ERR_NOT_FOUND = Element Solaris_Device does not exist.

原因

特定の操作 (たとえば、削除) に対し要素が指定されましたが、その要素は存在しません。指定された要素にスペルミスがあるか、あるいは意図した要素名の代わりに誤って存在しない要素名が指定されたことが考えられます。

解決方法

要素のスペルと名前が正しいか確認し、ネームスペース内にそのクラスが存在することを確認します。

例 *2: CIM_ERR_NOT_FOUND*

説明

この場合の CIM_ERR_NOT_FOUND エラーメッセージ は、次の 2 つのパラメータを使用します。

- 指定されたインスタンスの名前に置換されます。
- 指定されたクラスの名前に置換されます。

例

CIM_ERR_NOT_FOUND = Instance Solaris_EnterpriseData does not exist for class Solaris_ComputerSystem.

原因

インスタンスが存在しません。

解決方法

インスタンスを作成します。

例 *3: CIM_ERR_NOT_FOUND*

説明

この場合の CIM_ERR_NOT_FOUND エラーメッセージは 1 つのパラメータを使用しますが、このパラメータは指定されたネームスペースの名前に置換されています。

例

CIM_ERR_NOT_FOUND = Namespace verdant does not exist.

原因

指定されたネームスペースが見つかりません。このエラーは、入力ミスまたはスペルミスのために入力されたネームスペースの名前が正しくない場合に発生します。

解決方法

ネームスペースの名前を入力し直し、入力とスペルが正しいことを確認します。

CLASS_REFERENCE

説明

CLASS_REFERENCE 例外メッセージは、次の 2 つのパラメータを使用します。

- 参照関係を定義されたクラスの名前
- 参照プロパティの名前

例

CLASS_REFERENCE = Class SolarisExample1 must be declared as an association to have reference SolarisExample2

原因

クラスの定義に参照プロパティが使用されています。しかし、そのクラスは関連ではありません。クラスが参照をプロパティとして持つことを定義できるのは、別のクラスとの関連がある場合だけです。

解決方法

–association 修飾子を使ってクラスを関連として宣言します。

INVALID_CREDENTIAL

説明

INVALID_CREDENTIAL 例外メッセージは、パラメータを使用しません。

例

INVALID_CREDENTIAL = Invalid credentials.

原因

この例外メッセージは、無効なパスワードが入力された場合に表示されます。

解決方法

コマンドを再入力し、正しいパスワードを入力してください。

INVALID_QUALIFIER_NAME

説明

INVALID_QUALIFIER_NAME 例外メッセージは 1 つのパラメータを使用しますが、このパラメータは空の修飾子名を表す MOF (Managed Object Format) の表記に置換されています。

例

INVALID_QUALIFIER_NAME = Invalid qualifier name `` ``

原因

プロパティの修飾子が作成されましたが、修飾子の名前が指定されませんでした。

解決方法

修飾子の定義文に修飾子の名前を含めます。

KEY_OVERRIDE

説明

KEY_OVERRIDE 例外メッセージは、次の 2 つのパラメータを使用します。

- オーバーライドするプロパティ
- オーバーライドされるプロパティ

例

KEY_OVERRIDE = Non-key Qualifier `SolarisCard` cannot override key Qualifier `SolarisLock`.

原因

クライアントは、非キープロパティがキープロパティをオーバーライドするクラスを定義しています。CIM では、すべての非 abstract クラスは 1 つ以上のキー修飾子を必要とし、キークラス以外のクラスはキーを持つクラスをオーバーライドできません。

解決方法

CIM 仕様に指定されているように、この操作はできません。

KEY_REQUIRED

説明

KEY_REQUIRED 例外メッセージは 1 つのパラメータを使用しますが、このパラメータはキーを必要とするクラスの名前に置換されています。

例

KEY_REQUIRED = Concrete (non-abstract) class `ClassName` needs at least one key.

原因

非 abstract であるクラスにキー修飾子が指定されませんでした。CIM では、非 abstract クラスはすべて、1 つ以上の修飾子を必要とします。

解決方法

クラスにキー修飾子を作成します。

METHOD_OVERRIDDEN

説明

METHOD_OVERRIDDEN コマンドは、次の 3 つのパラメータを使用します。

- オーバーライドするメソッドの名前
- オーバーライドされるメソッドの名前
- 2 つめのパラメータをすでにオーバーライドしているメソッドの名前

例

METHOD_OVERRIDDEN = Method `Resume ()` cannot override `Stop()` which is already overridden by `Start()`

原因

別のメソッドによってすでにオーバーライドされているメソッドのオーバーライドを試みるメソッドが指定されました。オーバーライド済みのメソッドを再度オーバーライドすることはできません。

解決方法

この操作は正しくありません。

NEW_KEY

説明

NEW_KEY 例外メッセージは、次の 2 つのパラメータを使用します。

- キーの名前
- 新しいキーの定義を試みているクラスの名前

例

NEW_KEY = Class `CIM_PhysicalPackage` cannot define new key `[Key]`

原因

あるクラスが新しいキーの定義を試みていますが、スーパークラス内にキーがすでに定義されています。スーパークラスにいったんキーが定義されると、サブクラスに新しいキーを設定することはできません。

解決方法

そのようなクラスの作成をやめます。

`NO_CIMOM`

説明

`NO_CIMOM` 例外メッセージは 1 つのパラメータを使用しますが、このパラメータは CIM Object Manager の実行ホストに指定されたホストの名前に置換されています。

例

`NO_CIMOM = CIMOM  molly` not detected.

原因

CIM Object Manager が指定されたホストで動作していません。

解決方法

`/etc/init.d/init.wbem start` コマンドを使って CIM Object Manager を起動するか、CIM Object Manager が動作しているホストに接続します。

`NO_INSTANCE_PROVIDER`

説明

`NO_INSTANCE_PROVIDER` 例外メッセージは、次の 2 つのパラメータを使用します。

- インスタンスプロバイダが見つからないクラスの名前
- 指定されたインスタンスプロバイダの名前

例

`NO_INSTANCE_PROVIDER` = Instance provider `RPC_prop` for class `RPC_Agent` not found.

原因

指定されたインスタンスプロバイダの Java クラスが見つかりません。このエラーメッセージは、CIM Object Manager のクラスパスに以下のすべてが適切であるクラスが含まれていないことを示します。

- プロバイダクラスの名前
- プロバイダクラスのパラメータ
- プロバイダが定義される CIM クラス

解決方法

インスタンスプロバイダが CIM Object Manager のクラスパスにあるか確認します。

`NO_METHOD_PROVIDER`

説明

`NO_METHOD_PROVIDER` 例外メッセージは、次の 2 つのパラメータを使用します。

- メソッドプロバイダが見つからないクラスの名前
- 指定されたメソッドプロバイダの名前

例

`NO_METHOD_PROVIDER` = Method provider `Start_prop` for class `RPC_Agent` not found.

原因

指定されたメソッドプロバイダの Java クラスが見つかりません。このエラーメッセージは、CIM Object Manager のクラスパスに以下のすべてが適切であるクラスが含まれていないことを示します。

- プロバイダクラスの名前
- プロバイダクラスのパラメータ
- プロバイダが定義される CIM クラス

解決方法

メソッドプロバイダが CIM Object Manager のクラスパスにあるか確認します。

NO_OVERRIDDEN_METHOD

説明

NO_OVERRIDDEN_METHOD 例外は、次の 2 つのパラメータを使用します。

- オーバーライドしたメソッドの名前
- オーバーライドされたメソッドの名前

例

NO_OVERRIDDEN_METHOD = Method Write overridden by Read does not exist in class hierarchy.

原因

サブクラスのメソッドがスーパークラスのメソッドのオーバーライドを試みていますが、スーパークラスのメソッドはクラス階層内に存在しません。

解決方法

上位のクラス階層内にそのメソッドが存在することを確認します。

NO_OVERRIDDEN_PROPERTY

説明

NO_OVERRIDDEN_PROPERTY 例外メッセージは、次の 2 つのパラメータを使用します。

- オーバーライドされるプロパティの名前
- プロパティをオーバーライドする名前

例

NO_OVERRIDDEN_PROPERTY = Property A overridden by B does not exist in class hierarchy.

原因

サブクラスのプロパティがスーパークラスのプロパティのオーバーライドを試みていますが、スーパークラスのプロパティはクラス階層内に存在しません。

解決方法

上位のクラス階層内にそのプロパティが存在することを確認します。

`NO_PROPERTY_PROVIDER`

説明

`NO_PROPERTY_PROVIDER` 例外メッセージは、次の 2 つのパラメータを使用します。

- プロパティプロバイダが見つからないクラスの名前
- 指定されたプロパティプロバイダの名前

例

`NO_PROPERTY_PROVIDER` = Property provider `Write_prop` for class `RPC_Agent` not found.

原因

指定されたプロパティプロバイダの Java クラスが見つかりません。このエラーメッセージは、CIM Object Manager のクラスパスに、2 番目に指定されたクラスを含んでいないことを示します。

解決方法

CIM Object Manager のクラスパスを設定します。

`NO_QUALIFIER_VALUE`

説明

`NO_QUALIFIER_VALUE` 例外メッセージは、次の 2 つのパラメータを使用します。

- エラーを引き起こした修飾子の名前
- 修飾子が参照する要素。修飾子により、2 つめのパラメータはクラス、プロパティ、メソッド、参照のどれかになります。

例

`NO_QUALIFIER_VALUE` = Qualifier `[SOURCE]` for `Solaris_ComputerSystem` has no value.

原因

プロパティまたはメソッドに修飾子が指定されましたが、修飾子に値が含まれていません。たとえば、修飾子 VALUES には文字列配列を指定する必要があります。必要な文字列配列なしで VALUES 修飾子が指定されると、NO_QUALIFIER_VALUE エラーメッセージが表示されます。

解決方法

修飾子に必要なパラメータを指定します。各修飾子に必要な属性については、Distributed Management Task Force 作成の CIM Specification (URL: `http://dmtf.org/spec/cims.html`) を参照してください。

`NO_SUCH_METHOD`

説明

NO_SUCH_METHOD 例外メッセージは、次の 2 つのパラメータを使用します。

- 指定されたメソッドの名前
- 指定されたクラスの名前

例

NO_SUCH_METHOD = Method `Configure()` does not exist in class `Solaris_ComputerSystem`

原因

指定されたクラスにメソッドが定義されなかったことが考えられます。指定されたクラスにメソッドが定義されている場合には、定義の際にスペルミスにより別のメソッドが指定されたか、入力ミスの可能性があります。

解決方法

指定されたクラスにメソッドを定義するか、あるいはメソッド名とクラス名が正しく入力されているか確認します。

`NO_SUCH_PRINCIPAL`

説明

NO_SUCH_PRINCIPAL 例外メッセージは 1 つのパラメータを使用しますが、このパラメータはプリンシパル (ユーザーアカウント) の名前です。

例

NO_SUCH_PRINCIPAL = Principal `molly` not found.

原因

指定されたユーザーアカウントが見つかりません。ログイン時にユーザー名の入力が正しく行われなかったか、あるいはそのユーザーにユーザーアカウントが設定されていません。

解決方法

ログイン時にユーザー名を正しく入力します。そのユーザーにユーザーアカウントが設定されていることを確認します。

`NO_SUCH_QUALIFIER1`

説明

`NO_SUCH_QUALIFIER1` 例外メッセージは 1 つのパラメータを使用しますが、このパラメータは未定義の修飾子の名前です。

例

`NO_SUCH_QUALIFIER1` = Qualifier `[LOCAL]` not found.

原因

その修飾子はネームスペースに存在しません。

解決方法

この修飾子を定義します。標準の CIM 修飾子と CIM スキーマの修飾子の使用法については、Distributed Management Task Force 作成の CIM Specification (URL: `http://www.dmtf.org/spec/cims.html`) を参照してください。

`NO_SUCH_QUALIFIER2`

説明

`NO_SUCH_QUALIFIER2` 例外メッセージは、次の 2 つのパラメータを使用します。

- 定義されていない修飾子の名前
- 修飾子が変更するクラス、プロパティ、またはメソッドの名前

例

`NO_SUCH_QUALIFIER2` = Qualifier `[LOCAL]` not found for `CIM_LogicalElement`

原因

特定のクラスのプロパティまたはメソッドを変更するために新しい修飾子が指定されましたが、その修飾子はいくつかのスキーマの一部として定義されていません。有効な修飾子として認識されるように、この修飾子を CIM スキーマまたは拡張スキーマの一部として定義する必要があります。

解決方法

この修飾子を拡張スキーマの一部として定義するか、あるいは標準の CIM 修飾子を使用します。標準の CIM 修飾子と CIM スキーマの修飾子の使用法については、Distributed Management Task Force 作成の CIM Specification (URL: `http://www.dmtf.org/spec/cims.html`) を参照してください。

`NO_SUCH_SESSION`

説明

`NO_SUCH_SESSION` 例外は 1 つのパラメータ を使用しますが、このパラメータはセッション識別子です。

例

`NO_SUCH_SESSION` = No such session `4002`.

原因

クライアントセッションが見つからないと、この例外が表示されます。CIM Object Manager は、セキュリティ上の理由でそのセッションを削除します。第 3 章を参照してください。

解決方法

CIM 環境のセキュリティが保護されていることを確認します。

`NOT_HELLO`

説明

`NOT_HELLO` 例外メッセージは、パラメータを使用しません。

例

`NOT_HELLO` = Not a Hello message.

原因

このエラーメッセージは、hello メッセージ (CIM Object Manager に送信される最初のメッセージ) 内のデータが破損している場合に表示されます。

解決方法

セキュリティが侵害されていないことを確認し、再び接続してみてください。

NOT_INSTANCE_PROVIDER

説明

NOT_INSTANCE_PROVIDER 例外メッセージは、次の 2 つのパラメータを使用します。

- 問題のある Java クラスの名前
- インスタンスプロバイダを定義しているクラスの名前

例

NOT_INSTANCE_PROVIDER = device_prop_provider for class Solaris_Provider does not implement InstanceProvider.

原因

プロバイダで指定されている Java クラスへのパスに、InstanceProvider インタフェースが実装されていません。

解決方法

1 つめのパラメータの Java クラスに InstanceProvider インタフェースが実装されているか確認します。

NOT_METHOD_PROVIDER

説明

NOT_METHOD_PROVIDER 例外メッセージは、次の 2 つのパラメータを使用します。

- 問題のある Java クラスの名前
- メソッドプロバイダを定義しているクラスの名前

例

NOT_METHOD_PROVIDER = Provider device_method_provider for class Solaris_Provider does not implement MethodProvider.

原因

1 つめのパラメータに指定された Java クラスに MethodProvider インタフェースが実装されていません。

解決方法

1 つめのパラメータの Java クラスに MethodProvider インタフェースが実装されているか確認します。

NOT_PROPERTY_PROVIDER

説明

NOT_PROPERTY_PROVIDER 例外メッセージは、次の 2 つのパラメータを使用します。

- 問題のある Java クラスの名前
- プロパティプロバイダの定義が試みられているクラスの名前

例

NOT_PROPERTY_PROVIDER = Provider device_property_provider for class Solaris_Provider does not implement PropertyProvider.

原因

1 つめのパラメータの Java クラスに PropertyProvider インタフェースが実装されていません。

解決方法

1 つめのパラメータの Java クラスに PropertyProvider インタフェースが実装されているか確認します。

NOT_RESPONSE

説明

NOT_RESPONSE 例外メッセージは、パラメータを使用しません。

例

NOT_RESPONSE = Not a response message.

原因

この例外メッセージは、CIM Object Manager からの最初の応答メッセージが壊れている場合に表示されます。

解決方法

再び接続してみてください。

PROPERTY_OVERRIDDEN

説明

PROPERTY_OVERRIDDEN 例外メッセージは、次の 3 つのパラメータを使用します。

- オーバーライドするプロパティの名前
- オーバーライドされるプロパティの名前
- 2 つめのパラメータをすでにオーバーライドしているメソッドの名前

例

PROPERTY_OVERRIDDEN = Property Volume cannot override MaxCapacity which is already overridden by RawCapacity

原因

別のプロパティによってすでにオーバーライドされているプロパティのオーバーライドを試みるプロパティが指定されました。オーバーライド済みのプロパティを再度オーバーライドすることはできません。

解決方法

オーバーライドする別のプロパティを指定します。

PS_UNAVAILABLE

説明

PS_UNAVAILABLE 例外メッセージは 1 つのパラメータを使用しますが、このパラメータは固定記憶域 (CIM レポジトリ) が使用できなくなった理由を説明するメッセージです。

例

PS_UNAVAILABLE = The persistent store is unavailable.

原因

リポジトリが使用できなくなった場合は、原因についてこれより詳しい情報が最初のパラメータに示されます。

解決方法

この例外は一般的なエラー条件なので、記述の内容からエラーの原因を判断してください。

QUALIFIER_UNOVERRIDABLE

説明

QUALIFIER_UNOVERRIDABLE 例外メッセージは、次の 2 つのパラメータを使用します。

- DisableOverride フレーバが設定されている修飾子の名前に置換されます。
- 上記のパラメータによってオーバーライドしようとしている修飾子の名前に置換されます。

例

QUALIFIER_UNOVERRIDABLE = Test cannot override qualifier Standard because it has DisableOverride flavor.

原因

指定された修飾子のフレーバが DisableOverride または Override=False に設定されているため、この修飾子は別の修飾子をオーバーライドできません。

解決方法

この修飾子の特性を、EnableOverride または Override=True に設定し直します。

REF_REQUIRED

説明

REF_REQUIRED 例外メッセージは 1 つのパラメータを使用しますが、このパラメータは関連を持つクラスの名前です。

例

REF_REQUIRED = Association class CIM_Chassis needs at least two references.

原因

関連が定義されたが、必要な参照が指定されていません。Common Information Model では、関連は 2 つ以上の参照を含む必要があります。

解決方法

最初のパラメータの関連に、必要な参照を追加します。

SCOPE_ERROR

説明

SCOPE_ERROR 例外は、次の 3 つのパラメータを使用します。

- 指定された修飾子が変更する要素の名前
- 指定された修飾子の名前
- 最初のパラメータの Meta 要素型

例

SCOPE_ERROR = Qualifier [UNITS] for CIM_Container does not have a Property scope.

原因

修飾子の指定方法がスコープ定義の要件と矛盾しています。たとえば、CIM Specification では、[READ] 修飾子の定義にはプロパティのスコープが使用されます。したがって、[READ] を使ってクラスを修飾すると、スコープ例外になります。

注 - CIM Specification は、CIM 修飾子が変更できる CIM 要素の種類を定義しています。修飾子の使用方法についてのこの定義は、修飾子のスコープと呼ばれます。ほとんどの修飾子は、プロパティまたはメソッド、あるいはこの両方の変更を指示するスコープを持ちます。また、ほとんどの修飾子は、パラメータ、クラス、関連、インジケーション、またはスキーマの変更を指示するスコープを持ちます。

解決方法

指定された修飾子のスコープを確認します。CIM 修飾子の標準の定義については、Distributed Management Task Force により提供されている CIM Specification の「1. Qualifiers」(URL: http://www.dmtf.org/spec/cim_spec_v20) を参照してください。別の修飾子を使用するか、あるいは CIM 定義に従って修飾子を使用するようにプログラムを変更します。

SIGNATURE_ERROR

説明

SIGNATURE_ERROR 例外メッセージは、パラメータを使用しません。

例

SIGNATURE_ERROR = Signature not verified

原因

この例外メッセージは、メッセージが偶然に、または故意に壊された場合に表示されます。このメッセージは、メッセージが有効なチェックサムを持つチェックサムエラーとは異なりますが、署名はクライアントの公開鍵では検証できません。この保護により、セッションキーが解読され、第 3 者が使用しようとしても、セッションを作成した最初のクライアントだけが認証されます。

解決方法

セッションが不正侵入者によって侵害される場合に表示されるこのメッセージに対しての解決方法はありません。Solaris WBEM Services のセキュリティ機能の詳細は、第 3 章を参照してください。

`TYPE_ERROR`

説明

`TYPE_ERROR` 例外メッセージは、次の 5 つのパラメータを使用します。

- 指定された要素 (プロパティ、メソッド、修飾子など) の名前
- 指定された要素が属するクラスの名前
- 要素に定義されたデータ型
- 割り当てられた値のデータ型
- 割り当てられた実際の値

例

`TYPE_ERROR` = Cannot convert `sint16 4` to a `string` for `VolumeLabel` in class `Solaris_DiskPartition`

原因

プロパティまたはメソッドのパラメータ値と、定義されたそのデータ型が一致しません。

解決方法

プロパティまたはメソッドの値を、定義されたそのデータ型に一致させます。

`UNKNOWNHOST`

説明

UNKNOWNHOST 例外メッセージは 1 つのパラメータを使用しますが、このパラメータはホストの名前です。

例

UNKNOWNHOST = Unknown host molly

原因

クライアントが接続しようとしたホストはありません。

解決方法

ホスト名のスペルを確認し、正しい場合は、管理者に連絡します。

VER_ERROR

説明

VER_ERROR 例外は、1 つのパラメータを使用しますが、このパラメータはクライアントが接続しようとした CIM Object Manager のバージョン番号を示します。

例

VER_ERROR = Unsupported version 0.

原因

クライアントが接続しようとした CIM Object Manager は、このクライアントバージョンをサポートしていません。

解決方法

クライアント API か CIM Object Manager をアップグレードします。

付録 **A**

Common Information Model (CIM) の用語と概念

CIM の概念

ネットワークエンティティと管理機能がどのように表現され、関連しているかを理解する上で重要な CIM の基本的な用語と概念について説明します。Common Information Model とオブジェクト指向モデルの慣例 (独自のスキーマのモデル化など) の詳細は、Distributed Management Task Force が提供している `http://dmtf.org/spec/cim_tutorial` の CIM Tutorial を参照してください。

オブジェクト指向モデル

物理的または論理的に存在するオブジェクト、エンティティ、概念、または機能の表現手段としてオブジェクト指向モデルの原理を使用しています。オブジェクト指向モデルの目的は、物理的なエンティティをフレームワーク (モデル) で設定し、エンティティの特性と機能、およびエンティティとほかのエンティティとの関係を表現することです。CIM では、オブジェクト指向モデルを使用して、ハードウェア要素とソフトウェア要素をモデル化します。

Uniform Modeling Language

Uniform Modeling Language (UML) モデルは、図と言語で表されます。モデルを表現するための CIM 規則は、UML の図示概念に基づいています。UML は、図を使

用して物理的なエンティティを表現し、線を使用して関係を表現します。たとえば、UML ではクラスは矩形として表されます。各矩形には、表現対象であるクラスの名前が入ります。2 つの矩形間の線は、それらのクラスの関係を示します。2 つのクラスを上位クラスに結合するために分岐する線は、関連を示します。

CIM ダイアグラムは、色を使用して関係を詳しく説明します。

- 赤色の線 → 関連
- 青色の線 → 継承関係
- 緑色の線 → 集合

CIM の用語

次の用語は、CIM 固有の意味を持ちます。

スキーマ

モデル、スキーマ、およびフレームワークは同義語です。これらはそれぞれ、物理的または論理的に存在するエンティティの抽象表現です。CIM では、スキーマはクラスの名前付けと管理に使用される、名前の付けられたクラスの集まりを意味します。スキーマ内では、クラスとそのサブクラスは構文 `Schemaname_classname` によって階層的に表現されます。スキーマ内の各クラス名は、一意である必要があります。Solaris WBEM Services には、Solaris スキーマが付属しています。Solaris スキーマには、CIM 機能を Solaris 用に独自に拡張したすべてのクラスが含まれます。

クラスとインスタンス

WBEM では、クラスは基本的な管理ユニットのほとんどを表現するオブジェクトの集まりを意味します。たとえば、Solaris WBEM Services の主要な機能クラスとして、`CIMClass`、`CIMProperty`、`CIMInstance` が挙げられます。

クラスは、管理対象オブジェクトを作成するために使用される抽象的な概念です。クラスの特性は、そのクラスから作成される子オブジェクト (インスタンス) によって実現されます。たとえば、`CIMClass` を使用してインスタンス `CIMClass (Solaris_Computer_System)` を作成できます。

この CIMClass インスタンスは、「そのコンピュータシステムは何か」という質問に答えます。インスタンス名は、Solaris_Computer_System です。同じクラスタイプのインスタンスはすべて、同じクラステンプレートから作成されます。この例では、CIMClass が、タイプ Computer_System の管理対象オブジェクトを作成するテンプレートを提供します。

クラスには、動的クラスと静的クラスがあります。静的クラスのインスタンスは、CIM Object Manager によって格納され、要求がある場合に CIM Repository から取り出すことができます。動的クラス (システム使用量のような常に変化するデータを含むクラス) のインスタンスは、データの変化に伴いプロバイダアプリケーションによって作成されます。

カスタムクラス: CIM の拡張機能

管理環境に固有の管理対象オブジェクトをサポートするために、CIM の拡張機能としてカスタムクラスを開発できます。CIM Object Manager API は、Solaris オペレーティング環境向けに CIM を拡張する新しいクラスを提供します。

プロパティ

プロパティは、クラスの特性を定義します。たとえば、CIMProperty クラスを使用して、キーを特定の CIM クラスのプロパティとして定義できます。プロパティの値は、文字列、またはプロパティ範囲のベクトルとして CIM Object Manager から渡すことができます。各プロパティは、固有の名前と 1 つのドメイン (そのプロパティを所有するクラス) を持ちます。一定のクラスのプロパティは、そのサブクラスのプロパティによってオーバーライドされます。

プロパティには、CIMClass のプロパティである CIMProperty などがあります。

メソッド

プロパティと同様に、メソッドはそれらを所有するクラスに属します。メソッドは、一定のクラスのオブジェクトが実行するアクションです。たとえば、メソッド public String getName() は、インスタンスの名前を、そのキーとキーの値を連続した値として返します。これらのアクションは、集合的にクラスの動作を表現します。メソッドは、そのメソッドを所有するクラスにしか属すことができません。クラスとの関係上、各メソッドは固有の名前を持つ必要があります。一定のクラスのメソッドは、そのサブクラスのメソッドによってオーバーライドできます。

新しいクラスはスーパークラスからメソッドの定義を継承しますが、スーパークラスの実装は継承しません。修飾子によって示されるメソッドの定義は、実装される新しいメソッドを提供できるプレースホルダとしての役割を果たします。CIM Object Manager は、ツリー内で下位レベルクラスからルートクラスの方向にメソッドを順に検査し、メソッドを示す修飾子型を検索します。

ドメイン

プロパティおよびメソッドは、クラス内で宣言されます。プロパティまたはメソッドを所有するクラスは、プロパティまたはメソッドのドメインと呼ばれます。

修飾子とフレーバ

CIM 修飾子は、CIM のクラス、プロパティ、メソッド、およびパラメータの特性を示すために使用されます。修飾子は、新しいクラスによって継承される固有の属性 (名前、型、値など) を持ちます。

インジケーション

インジケーション (オブジェクトとクラスのタイプ) は、イベント発生の結果として作成され、タイプ階層で示されます。インジケーションは、プロパティ、メソッド、およびトリガーを持つことができます。トリガーは、既存のクラスに対する変更や、新しいインジケーションインスタンスの作成を引き起こすイベントなどのシステムオペレーションです。

関連

関連は、2 つ以上のクラス間の関係を表現するクラスです。関連を使用すると、一定のクラスに複数の関連インスタンスを作成し、システムコンポーネントをさまざまな方法で関連付けることができます。関連は、システムコンポーネントの関係を表現する手段を提供します。

関連の定義方法のおかげで、関連するどのクラスにも影響を与えずにクラス間の関係を構築できます。関連を追加しても、関連するクラスのインタフェースには影響しません。影響があるのは関連だけです。

参照と範囲

参照はプロパティの一種で、関連に関係するオブジェクトの役割を定義します。参照は、関連におけるクラスの役割名を指定します。参照のドメインは、関連です。参照の範囲は、参照の種類を示す文字列で表されます。

オーバーライド

オーバーライド関係は、スーパークラスから継承されたプロパティまたはメソッドを、サブクラスから継承されたプロパティまたはメソッドで置換することを示すために使用されます。CIM では、どの修飾子を持つプロパティとメソッドがオーバーライドできるかをガイドラインで定めています。たとえば、CIM ガイドラインはキープロパティはオーバーライドできないと定めているため、クラスの修飾子型がキーと設定される場合、このプロパティをオーバーライドできません。

コアモデルの概念

次の節では、CIM のコアモデルについて説明しています。

システムとしてのコアモデル

コアモデルでは、システムとそれらの機能が管理対象オブジェクトとして記述されるアプリケーションを開発するために使用できるクラスと関連を提供します。これらのクラスと関連は、システムを構成するすべての要素 (物理的な要素および論理的な要素) に固有の特性を与えています。物理的な特性とは、空間を占有し、物理的な基本原則に従う性質を意味します。論理的な特性とは、物理的な環境面 (システム状態やシステムの能力など) を管理、調整するために使用される抽象概念を意味します。

コアモデルには次の論理要素が存在します。

表 **A–1** コアモデルの要素

要素名	説明
システム	ほかの論理要素をグループ化したもの。システムはそれ自体が論理要素であるため、ほかのシステムのコンポーネントとなり得る
ネットワークコンポーネント	ネットワークの接続形態を提供するクラス
サービスとアクセスポイント	システム機能にアクセスできる構造を構成するための機構を提供する
デバイス	ハードウェアエンティティの抽象概念またはエミュレーション。デバイスは、実際のハードウェアとしての形をとらないこともある

次の節では、システムの特性をエミュレートするためにコアモデルに提供されているクラスと関連について説明します。

コアモデルが提供するシステムクラス

次の表は、システムとしてのコアスキーマを表すクラスを示しています。これらのクラスのインスタンスは、通常、そのクラスに含まれるオブジェクトの下位に属します。

表 **A–2** コアモデルのシステムクラス

クラス名	説明	例
`Managed System Element`	システム要素階層の基底クラス。識別可能なシステムコンポーネントはすべて、このクラスに含めることができる	ファイルやデバイス (ディスクドライブやコントローラなど) のようなソフトウェアコンポーネント、およびチップやカードのような物理的なコンポーネント
`Logical Element`	抽象的なシステムコンポーネントを表すすべてのシステムコンポーネントの基底クラス	論理デバイスの形をしたプロファイル、プロセス、またはシステム機能

表 A–2　コアモデルのシステムクラス　続く

System	一連の管理対象システム要素の集合である論理要素。この集合は、機能的に一体として動作する。System のすべてのサブクラスには、インスタンスの集合が必要な Managed System Element クラスの明確なリストが存在する	LAN、WAN、サブネット、イントラネット
Service	デバイスまたはソフトウェア機能 (あるいはこの両方) によって提供される機能の記述と管理に必要な情報を含む論理要素。Service は、機能の実装を設定、管理する多目的オブジェクトである。Service 自体は機能ではない	プリンタ、モデム、ファックスマシン

コアモデルが提供するシステム関連

関連は、ほかのクラスによって共有される関係を定義するクラスです。関連クラスには、そのクラスの目的を示す ASSOCIATION 修飾子が付けられます。関連クラスは、2 つ以上の参照 (特定の関係を共有するクラスの名前) を持つ必要があります。関連のインスタンスは、常に関連クラスに属します。

関連には、次のような形式があります。

- 1 対 1
- 1 対多
- 1 対ゼロ
- 集約 (システムとそのコンポーネント間の包含関係など)

関連は、システムとそのコンポーネントである管理対象要素との間の関係を表現します。クラス間の関係の定義には、一般的な 2 種類の関連が使用されます。

CIM スキーマは、次に示す基本的な 2 種類の関連を定義します。

- コンポーネント関連 — あるクラスが別のクラスの一部であることを示す

- 依存関連 — あるクラスが別のクラスなしには機能することも、存在することも不可能であることを示す

これらの関連タイプは抽象型です。つまり、関連クラスは、インスタンスを単独では所有しません。

コンポーネント関連

コンポーネント関連は、システムのコンポーネントとシステム自体の関係を表します。コンポーネント関連は、どの要素がシステムを構成するかを示します。コンポーネント関連を表す abstract クラスは、下位クラスにこのタイプの具体的な関連を作成するために使用されます。下位の具体的な関連は、「コンポーネント (クラス) が、ほかのコンポーネントとどのような構成関係にあるか」を表します。

コンポーネント関連は、その特有の役割として、システムと、システムの論理的なコンポーネントおよび物理的なコンポーネント間の関係を表します。

依存関連

依存関連は、互いに依存し合うオブジェクト間の関係を確立します。コアモデルは、次の依存関係を提供します。

- 機能的—依存オブジェクトは、依存対象であるオブジェクトなしでは機能できない
- 存在 — 依存オブジェクトは、依存対象であるオブジェクトなしでは存在できない

コアモデルには、次の依存関係があります。

表 A–3 Core Model の依存関係

依存関連	説明
HostedService	サービスとその機能が存在するシステム間の関連。この関連の多重度は、1 対多である。1 つのシステムは、多くのサービスを管理できるため、サービスは、それらを管理しているシステムに強く依存している。一般に、サービスは、そのサービスを実装する論理デバイスまたはソフトウェア機能が入ったシステム上で管理されるので、このモデルでは、複数のシステムに渡って管理されるサービスは表されない。この場合システムは、単一のホストにそれぞれ置かれているサービスの集合ポイントの役割を果たすアプリケーションシステムとしてモデル化される
HostedAccessPoint	サービスアクセスポイント (SAP) と SAP 上で提供されているシステム間の関連。この関連の多重度は 1 対多であり、システムに強く依存している。各システムは多数の SAP を管理できる。このモデルの特徴は、サービスアクセスポイントを、サービスがアクセスできるシステムと同じホストに置くことも別のホストに置くこともできることである。このため、このモデルは分散システム (コンポーネントサービスが複数のホストに置かれるアプリケーションシステム) と分散アクセス (アクセスポイントがほかのシステム上で管理されるサービス) の両方を表すことができる
ServiceSAPDependency	サービスとサービスアクセスポイント間の関連。この関連は、サービスがその機能を提供するためには参照されている SAP が必要であることを示す
SAPSAPDependency	2 つの SAP 間の関連のある SAP がそのサービスを利用またはそのサービスに接続するためには別の SAP が必要であることを示す。
ServiceAccessBySAP	サービスのアクセスポイントを識別する関連。たとえば、プリンタは、複数のシステム上で管理される Netware、Apple Macintosh、または Windows のサービスアクセスポイントによってアクセスされる可能性がある

コアモデルの拡張例

コアモデルから、多数の拡張機能を開発できます。たとえば、Managed System Element クラスの抽象化した Managed Element クラスを追加できます。コアモデルには、この Managed Element クラスの下位クラス (ユーザーや管理者など、管理対象のシステムドメインに含まれないオブジェクトを表現するクラス) を追加できます。

共通モデルスキーマ

共通モデルスキーマは、以下のようなモデルを提供します。

システムモデル

システムモデルは、管理対象の環境を構成する最上位レベルのシステムオブジェクトであるコンピュータ、アプリケーション、およびネットワークシステムを表しています。

デバイスモデル

デバイスモデルは、システムの基本機能 (ストレージ、処理、通信、入出力など) を提供しているシステム上の個々の論理ユニットを表します。システムデバイスはシステムの物理的なコンポーネントそのものと考えがちですが、これは誤りです。これは、管理されるのは実際のコンポーネント自体ではなく、オペレーティングシステム上でのデバイスの記述であるためです。

オペレーティングシステムの記述は、システムの実際のコンポーネントと 1 対 1 では対応していません。たとえば、モデムは個々の実際のコンポーネントに対応できます。モデムは、LAN アダプタとモデムをサポートする多機能カードによって提供することも、システム上で動作する通常のプロセスによって提供することもできます。このモデルを使用または拡張するためには、論理的なデバイスと物理的なコンポーネント間の違いを混同することなく理解することが必要です。

アプリケーション管理モデル

CIM アプリケーション管理モデルは、ソフトウェア製品とアプリケーションの管理に通常必要な情報を詳細に記述した情報モデルです。このモデルは、スタンドアロンのデスクトップアプリケーションから、高性能、マルチプラットフォーム、分散型、インターネットベースのアプリケーションなど、多様なアプリケーション構造に使用できます。このモデルは、単一のソフトウェア製品を記述するのにも、ビジネスシステムを形成している相互に依存した複数のアプリケーションを記述するのにも使用できます。

このアプリケーションモデルの主要な特性の 1 つに、アプリケーションのライフサイクルという考えがあります。アプリケーションは、配付可能、インストール可能、実行可能、実行中のいずれかの状態にあります。各種のオブジェクトの解釈と特性は、アプリケーションをある状態から別の状態に変換するために使用される機構と多くの場合関連づけられて、アプリケーションを記述しています。

ネットワークモデル

ネットワークモデルは、ネットワークトポロジ、ネットワークの接続性、ネットワークアクセスの制御と提供に必要な各種のプロトコルとサービスなど、ネットワーク環境のさまざまな側面を表現します。

物理モデル

物理モデルは、実際の物理的な環境を表します。管理される環境のほとんどは、論理オブジェクト (実際のオブジェクトではなく環境の情報面を記述するオブジェクト) で表されます。システム管理のほとんどは、システム状態を表現、制御する情報の操作に関係しています。実際の物理的な環境に対する操作 (物理ドライブの読み取りヘッドの移動やファンの起動など) はすべて、通常、論理的な環境の操作による間接的な結果として発生するにすぎません。そのため、物理的な環境は一般に直接的な重要性はありません。

一般にシステムの物理的な各部には、計測機構は付けられていません。それらの現在の状態 (およびそれらの存在そのもの) は、システムについてのほかの情報から間接的に推測できるだけです。CIM では、物理モデルは環境のこの側面を表します。このモデルは、システムごとに大きく異なり、また技術の進展に伴い劇的に変化すると考えられます。また、管理環境の物理的な側面についての情報提供を具体的な目的として、アプリケーション、ツール、および環境を個別に使用するために、物理的な環境を追跡、計測することは、常に困難を極めると予測されます。

付録 **B**

Solaris スキーマ

インストール時に CIM Object Manager は、/usr/sadm/mof/ ディレクトリにある MOF ファイルをコンパイルします。MOF ファイルは CIM スキーマや Solaris スキーマを記述するファイルです。CIM スキーマファイルには、Common Information Model の Core Model と Common Model が実装されています。このファイル名の一部には CIM が使用されています。Solaris スキーマファイルには、Common Information Model に対する Solaris 拡張が実装されています。このファイル名の一部には Solaris が使用されています。この付録では次の Solaris スキーマファイルについて説明します。

- Solaris スキーマファイル
- Solaris_Schema1.0.mof ファイル
- Solaris_Core1.0.mof ファイル
- Solaris_Application1.0.mof ファイル
- Solaris_System1.0.mof ファイル
- Solaris_Device1.0.mof ファイル
- Solaris_Acl1.0.mof ファイル

Solaris スキーマファイル

次の表に、/usr/sadm/mof/ にある Solaris スキーマファイルの簡単な概要を示します。

表 **B–1** Solaris スキーマファイル

Solaris スキーマファイル	説明
`Solaris_Schema1.0.mof`	Solaris スキーマのすべてのコンポーネントを読み込む。Solaris スキーマの各 MOF ファイルを実行する順序を指定する
`Solaris_Core1.0.mof`	WBEM のコア機能を実装する。これにより、ユーザーはロケール、修飾子、プロバイダを設定できる
`Solaris_Application1.0.mof`	Solaris のパッケージやパッチを CIM に作成する
`Solaris_System1.0.mof`	システムに対する Solaris スキーマコンポーネントを作成する。これらのコンポーネントは、システムのオペレーティングシステムやプロセスなど。Solaris_Process や Solaris_OperatingSystem の定義によって CIM スキーマの定義を拡張する
`Solaris_Device1.0.mof`	コンピュータが CIM Object Manager とともに動作するように、システムのプロセッサを記述する
`Solaris_Acl1.0.mof`	ユーザー ACL の基底クラスと修飾子を設定する

ファイルの詳細は、下記の各項を参照してください。

Solaris_Schema1.0.mof ファイル

`Solaris_Schema1.0.mof` ファイルは、Solaris スキーマに属するすべての MOF ファイルの最上位レベルのコンテナです。このファイルには、MOF ファイルがコンパイル順に表示されています。コンパイルによって生成される Java クラスは CIM Object Manager に送信されます。CIM Object Manager は、それらの Java クラスをイベントとして規定するか、CIM Object Manager Repository に送信し、オブジェクトとして格納します。次のコードは、`Include` 宣言がコンパイル順に指定されています。

```
// ===================================================================
// Title:       Solaris Master MOF 1.0
// Filename:    Solaris_Schema1.0.mof
```

(続く)

```
// Version:     1.0
// Author:      Sun Microsystems, Inc.
// Date:        02/01/1999
// Description:


// ===================================================================
// ===================================================================
// Includes
// ===================================================================
#pragma Include ("usr/sadm/mof/Solaris_Core1.0.mof")
#pragma Include ("usr/sadm/mof/Solaris_Application1.0.mof")
#pragma Include ("usr/sadm/mof/Solaris_System1.0.mof")
#pragma Include ("usr/sadm/mof/Solaris_Device1.0.mof")
//#pragma Include ("/opt/SUNWconn/wbem/schema/Solaris_Physical1.0.mof")
#pragma Include ("usr/sadm/mof/Solaris_Acl1.0.mof")
```

コンパイラは、Solaris_Schema1.0.mof ファイルの宣言を 1 行ずつ解析し、Include ステートメントに指定されたファイルをコンパイルします。このようにしてコンパイラは、Solaris_Schema1.0.mof ファイルに記述されているすべてのファイルをコンパイルします。

Solaris_Core1.0.mof ファイル

Solaris_Core1.0.mof ファイルは、Solaris_Schema1.0.mof ファイルの次にコンパイルする Solaris スキーマファイルです。このファイルには、Solaris Provider の Solaris_ComputerSystem と Solaris_SerialPortSetting 部分の定義が含まれています。この定義には、LogRecord、Solaris_Product、Solaris_LogService などがあります。

Solaris_ComputerSystem の定義

Solaris_Core1.0.mof ファイルの最初のセクションでは、Solaris_ComputerSystem の定義を CIM_UnitaryComputerSystem クラスの拡張として設定します。

```
[Provider("com.sun.wbem.solarisprovider.Solaris")]
class Solaris_ComputerSystem:CIM_UnitaryComputerSystem
{
};
```

Solaris_SerialPortSetting とロギングの定義

Solaris_SerialPortSetting 定義では、CIM_SerialPortSetting クラスを Solaris オペレーティング環境に拡張します。Solaris_LogRecord クラスでは、WBEM システムログに書き込めるデータの型を定義します。

```
 [Provider ("com.sun.wbem.solarisprovider.Solaris")]
class Solaris_SerialPortSetting:CIM_ElementSetting
{
        [override("Element")]
    Solaris_SerialPort REF Element;
       [override("Setting")]
    Solaris_SerialPortConfiguration REF Setting;
};

 [Provider ("com.sun.wbem.solarisprovider.Solaris")]
class Solaris_LogRecord
{
```

WBEM システムログは、アプリケーションログ、システムログ、セキュリティログという 3 つの一般的なカテゴリに分類できます。ログレコードには、情報ログ、警告ログ、エラーログといった異なる重要度レベルが割り当てられます。アプリケーションの中で Solaris_LogRecord 呼び出しを使ってロギングを有効にする方法については、第 5 章 の「API によるロギングの有効化」を参照してください。

すべてのログレコードには、Solaris_LogRecord クラスに定義されている標準形式を使用します。Solaris_LogRecord のプロパティには、アプリケーションからログレコードに渡されるデータの型が指定されています。CIM Object Manager や CIM Object Manager Repository は、記録されるデータを識別するためにデータの一

部を必要とします。このようなプロパティには、読み取り専用であることを示すために [read, key] 修飾子が付けられます。このようなプロパティのデータは表示できますが、変更はできません。次のリストは、Solaris_Core1.0.mof ファイルの Solaris_LogRecord に割り当てられるプロパティを示しています。

```
{
 [read, key]
 sint64 RecordID;

 [read, key]
 sint32 RecordHashCode;

 [read, key]
 string Filename;

 [read]
 datetime RecordDate;

 [read, write]
 sint32 category;

 [read, write]
 sint32 severity;

 [read, write]
 string AppName;

 [read, write]
 string UserName;

 [read, write]
 string ClientMachineName;

 [read, write]
 string ServerMachineName;

 [read, write]
 string SummaryMessage;

 [read, write]
 string DetailedMessage;
 [read, write]
 string data;

 [read, write]
 boolean SyslogFlag;
};
```

Solaris_LogRecord のプロパティを定義したら、Solaris_LogService クラスを CIM_Service の拡張として定義します。Solaris_LogService ではロギングサービスの操作を制御します。

```
 [Provider ("com.sun.wbem.solarisprovider.Solaris")]
class Solaris_LogService : CIM_Service
{
```

Solaris_LogService クラスに指定するそれぞれの関数では、ログファイルのデータをどのように処理するかを定義します。たとえば、clearLog 関数では、ログファイルのすべてのデータを消去し、ログファイルを一新して新しいデータを受け取れるようにします。

```
{
 sint32 clearLog([IN] string fileName);

 sint64 getNumRecords([IN] string fileName, [OUT] sint64 numRec);

 sint32 listLogFiles([OUT]string logFiles[]);

 sint32 getCurrentLogFileName([OUT] string fileName);

 sint32 getNumLogFiles([OUT] sint32 numFiles);

 sint64 getLogFileSize([OUT] sint64 fileSize);

 sint32 getSyslogSwitch([OUT] string switch);

 sint32 getLogStorageName([OUT] string fileName);

 sint32 getLogFileDir([OUT] string dirName);

 sint32 setProperties([IN] string data[]);
};
```

Solaris_LogServicePropertiesクラスでは、ログファイルの次の属性を制御します。

- ログファイルを書き込むディレクトリ
- ログファイルの名前
- ログファイルが作成された日付
- ログファイルの名前を変更し、ログファイルを保存するために必要なログファイルのサイズ
- アーカイブに保存できるログファイルの数

- ログデータを SysLog に書き込めるかどうか。SysLog は、Solaris オペレーティング環境のデフォルトのロギングシステム

Solaris_Application1.0.mof ファイル

Solaris_Application1.0.mofファイルでは、Solaris スキーマを拡張するアプリケーションパッケージやパッチを設定します。

パッケージ

Solaris_Application1.0.mof ファイルには、標準の Solaris パッケージを表すクラスが含まれています。これらのパッケージは、Solaris オペレーティング環境に個別にインストールしたり、削除したりすることができます。

次の表は、設定できるアプリケーションパッケージの属性です。pkginfo フィールドは、pkginfo コマンドを実行したときにパッケージ属性が表示されるフィールド、説明はパッケージ属性の説明です。

表 B–2 指定可能なパッケージ情報

パッケージ属性	pkginfo フィールド	説明
名前	PKGINST	パッケージに付ける名前。通常、名前は、ベンダー名を表す 3〜4 文字の大文字とパッケージを固有に識別する 5 文字までの小文字で表す
説明	DESC	パッケージの簡単な説明を文字列で指定する
キャプション	NAME	パッケージの簡単な追加の説明を文字列で指定する

表 B–2 指定可能なパッケージ情報 続く

カテゴリ	CATEGORY	パッケージに含まれる情報の種類。これは、パッケージにビデオアプリケーション、グラフィックアプリケーション、Java アプリケーションなどが含まれる場合に使用する。カテゴリは自由形式の文字列で、通常、文字列には *system* や *application* などの単語が含まれる。カテゴリ文字列には、複数の単語をコンマで区切って指定できる。値としては、*ALE, graphics, java, video, JFP, SyMON.* などが指定できる
アーキテクチャ	ARCH	このパッケージが適用されるシステムアーキテクチャ。アーキテクチャ属性は文字列かその列挙。マニュアルページだけからなるパッケージなど汎用テキストパッケージを指定する場合は、`all` を使用する。バイナリを表す場合は、プロセッサタイプを示す `sparc` や `i386` を使用する。カーネルを表す場合は、プロセッサタイプのサブタイプを示す `sparc.sun4u` を使用する
ベースディレクトリ	BASEDIR	パッケージがインストールされているトップレベルのディレクトリを示す有効な UNIX のパス
メーカー	VENDOR	製品の製造メーカー
ビルド番号	PSTAMP	ビルドホスト名とタイムスタンプを含む文字列
インストール日付	INSIDE	オペレーティングシステムがインストールされた日付と時刻

表 B–2　指定可能なパッケージ情報　続く

パッケージの状態	STATUS	パッケージが完全にインストールされているかどうかを示す値。unknown、completely installed、partially installed のいずれか
サポート情報	HOTLINE	サポートが必要なときの連絡先を示す文字列

パッチ

Solaris_Application1.0.mof ファイルでは、さらに、アプリケーションの問題や更新バージョンに対する修正をパッチの形で提供します。次の表は、指定できるパッチの属性とその説明です。

表 B–3　提供可能なパッチ情報

パッチ属性	説明
廃止	廃止されたパッチや現在のパッチに統合されているパッチを示す
必須	現在のパッチが動作するために必要なパッチのリスト
非互換	現在のパッチと互換性のないパッチのリスト
メーカー	メーカー名のリスト
インストール日付	パッチがインストールされた日付と時刻を示す日付/時刻を示す値

Solaris_System1.0.mof ファイル

Solaris_System1.0.mof ファイルでは、Solaris_Process クラスと Solaris_OperatingSystem クラスを定義します。

```
// ===================================================================
// Solaris_Process
// ===================================================================

 [Provider("Solaris"),
  Description ("A Solaris process that is running.")
 ]
class Solaris_Process:CIM_Process
{
};

// ===================================================================
// Solaris_OperatingSystem
// ===================================================================

 [Provider("Solaris"),
  Description ("The Solaris Operating System.")
 ]
class Solaris_OperatingSystem:CIM_OperatingSystem
{
};
```

Solaris_Device1.0.mof ファイル

Solaris_Device1.0.mof ファイルでは、次のクラスが定義されます。

- Solaris_Processor クラス。CIM_Processor の拡張として定義される
- Solaris_DiskDrive クラス。CIM_DiskDrive の拡張として定義される
- Solaris_SerialPort クラス。CIM_SerialController の拡張として定義される。この構成クラス Solaris_SerialPortConfiguration が CIM_Setting の拡張として定義される

Solaris_Processor

Solaris_Processor は CIM_Processor の拡張として次のプロパティで定義されます。

```
class Solaris_Processor:CIM_Processor
{
    string Name;
    string Description;
    string Architecture;
    string ClockSpeed;
    string SparcVersion;
    uint32 D_Cache;
    uint32 E_Cache;
    uint32 I_Cache;
};
```

Solaris_DiskDrive

現在、Solaris_DiskDrive に対し次のクラス定義されています。

```
]
class Solaris_DiskDrive:CIM_DiskDrive
{
```

Solaris_SerialPort

Solaris_SerialPort クラスはブール値プロパティとして定義されているため、ボーレートやパリティなどのシリアルポート特性をポートでどのように処理するかを制御できます。

```
// ==================================================================
// Solaris_SerialPort
// ==================================================================

 [Provider("com.sun.wbem.solarisprovider.Solaris"),
  Description (
               "This is the MOF file that defines a Solaris serial port.")
 ]
class Solaris_SerialPort:CIM_SerialController
{
 [read]
    boolean ServiceEnabled;
 [read]
```

(続く)

```
    boolean SettableBaudRate;
        [read]
    boolean SettableDataBits;
        [read]
    boolean SettableFlowControl;
        [read]
    boolean SettableParity;
        [read]
    boolean SettableParityCheck;
        [read]
    boolean SettableStopBits;
        [read]
    boolean SupportsParityCheck;
        [read]
    boolean SupportsXOnXOff;
        [read]
    boolean SupportsXOnXOffSet;
  [read]
    string PortMonitor;
  [read]
    string ServiceTag;
  [read]
    string Comment;
    boolean DisablePortService();
};
```

Solaris_PortConfiguration

Solaris_PortConfiguration のプロパティでは、データの値をユーザーが表示したり、変更したりできるかどうかを指定します。

```
[Provider ("Solaris") ]
class Solaris_SerialPortConfiguration:CIM_Setting
{
[read, write]
boolean ServiceEnabled;
[read, write]
uint32 BaudRate;
[read, write]
string TerminalType;
[read, write]
boolean TTYFlag_Init;
[read, write]
boolean TTYFlag_Bidirectional;
[read, write]
boolean TTYFlag_CarrierConnect;
[read, write]
```

(続く)

```
boolean SoftwareCarrier;
[read, write]
boolean CreateUtmp;
[read, write]
string LoginPrompt;
[read, write]
string Comment;
[read, key]
string ServiceTag;
[read, key]
string PortName;
[read]
string deviceName;
[read, write]
string PortmonTag;
[read, write]
string ServiceProgram;
[read, write]
string StreamsModules;
[read, write]
string Timeout; };
```

Solaris_Acl1.0.mof ファイル

このファイルでは Solaris WBEM Services のセキュリティクラスを指定します。このファイルには、Solaris オペレーティング環境のアクセス制御リスト、ユーザー、およびネームスペースの基底クラスが定義されます。このファイルについては、第 3 章の「API によるアクセス制御の設定」を参照してください。Solaris WBEM Services のセキュリティ機能については、第 3 章を参照してください。

用語集

この用語集では、Solaris WBEM Services マニュアルで使用されている用語について説明します。これらの用語は開発者の間ではよく使用されていますが、WBEM 環境独自のものや、異なる意味で使用されているものも含まれています。

Backus-Naur Form (BNF)
プログラミング言語の構文を指定するメタ言語。

CIM Object Manager Repository
Common Information Model Object Manager (CIM Object Manager) によって管理される主要な格納領域。このリポジトリには、管理対象オブジェクトを表現するクラスとインスタンスの定義、およびそれらの間の関係が格納される。

CIM スキーマ
各管理環境で発生する管理対象オブジェクトの表現に使用されるクラス定義の集まり。

「コアモデル」、「共通モデル」、および「拡張スキーマ」も参照。

CIM は、メタモデルと標準スキーマに分類される。メタモデルは、スキーマを構成するエンティティの種類を示すとともに、管理対象オブジェクトを表現するオブジェクトにそれらのエンティティをどのように結合できるかを定義する。

Distributed Management Task Force (DMTF)
パソコンの使用、理解、設定、および管理の簡易化を行なっている業界のコンソーシアム。

Interface Definition Language (IDL)	ある言語で記述されたプログラムまたはオブジェクトが、認識できない言語で記述された別のプログラムと通信できるようにする言語の総称。
Managed Object Format (MOF)	クラスとインスタンスを定義するコンパイラ型言語。MOF コンパイラ (`mofcomp`) は、`.mof` テキストファイルを Java クラスにコンパイルし、そのデータを CIM Object Manager Repository に追加する。MOF を使用するとコードを記述する必要がないため、CIM Object Manager Repository を簡単にかつ速く変更できる。
MOF ファイル	Managed Object Format (MOF) 言語を使用するクラスとインスタンスの定義が入ったテキストファイル。
Solaris スキーマ	CIM スキーマを Sun の Solaris 用に固有に拡張したスキーマ。Solaris スキーマには、一般的な Solaris オペレーティング環境に存在する管理対象オブジェクトを表現するクラスとインスタンスの定義が含まれる。
Simple Network Management Protocol (SNMP)	インターネット参照モデルのプロトコルの 1 つで、ネットワーク管理に使用される。
Unified Modeling Language (UML)	ソフトウェアシステムを説明するために使用される表記言語。ボックスと線を使用してオブジェクトと関係を表す。
Unicode	多くの既知の文字をコード化できる 16 ビットの文字セット。Unicode は、全世界的に使用されている。
UTF-8	Unicode 文字データの変換形式としての機能を果たす 8 ビットの変換形式。
Win32 スキーマ	CIM スキーマの Microsoft 固有の拡張機能。Win32 スキーマには、一般的な Win32 環境に存在する管理対象オブジェクトを表現するクラスとインスタンスの定義が含まれる。
インジケーション	インスタンスの作成、変更、または削除、インスタンスのアクセス、プロパティの変更またはアクセスなどのアクションの結果として実行されるオペレーション。インジケーションは、指定された時間の経過からも起きる。インジケーションの結果、通常はイベントが発生する。

インスタンス	特定のクラス、または特定のイベントから作成された管理対象オブジェクトを表す。インスタンスには、実際のデータが含まれる。
インスタンスプロバイダ	システム固有のクラスおよびプロパティ固有のクラスのインスタンスをサポートするプロバイダ。インスタンスプロバイダは、データの検索、変更、削除、および列挙をサポートできる。インスタンスプロバイダは、メソッドの呼び出しも行える。 「プロパティプロバイダ」も参照。
インタフェースクラス	オブジェクトセットへのアクセスに使用されるクラス。インタフェースクラスには、列挙範囲を表す abstract クラスを使用できる。 「列挙」と「スコープ」も参照。
オーバーライド	派生クラス内のプロパティ、メソッド、または参照が、継承ツリー内の親クラスまたは指定された親クラス内の類似した構成体を上書きすること。
オブジェクトパス	ネームスペース、クラス、およびインスタンスへのアクセスに使用される定形の文字列。システム上の各オブジェクトは、ローカルに、またはネットワーク上でそれ自体を識別する一意のパスを持つ。オブジェクトパスは、概念的には URL (Universal Resource Locator) に似ている。
拡張スキーマ	CIM スキーマの 3 つめの層であり、Solaris や UNIX などのプラットフォーム固有の CIM スキーマの拡張機能を含む。 「共通モデル」と「コアモデル」も参照。
仮想関数テーブル **(Virtual Function Table、VTBL)**	関数ポインタ (クラスの実装など) のテーブル。VTBL 内のポインタは、オブジェクトがサポートするインタフェースのメンバーを指す。
管理アプリケーション	管理環境内の 1 つ以上の管理対象オブジェクトから発生する情報を使用するアプリケーションまたはサービス。管理アプリケーションは、CIM Object Manager およびプロバイダから CIM Object Manager API を呼び出すことによってこの情報を検索する。
管理情報ベース	管理対象オブジェクトのデータベース。

管理対象オブジェクト	WBEM のクラスとして表現される、ハードウェアコンポーネントまたはソフトウェアコンポーネント。管理対象オブジェクトについての情報は、インスタンスプロバイダ、プロパティプロバイダ、および CIM Object Manager Repository から提供される。
関連クラス	2 つのクラス間、または 2 つのクラスのインスタンス間の関係を表現するクラス。関連クラスのプロパティには、2 つのクラスまたはインスタンスを指すポインタ (参照) が含まれる。WBEM クラスはすべて、1 つ以上の関連に含めることができる。
キー	クラスインスタンスに一意の識別子を提供するために使用されるプロパティ。キープロパティは、キー修飾子によって示される。
キー修飾子	クラス内のプロパティ (そのクラスのキーの一部として機能する) に付加される修飾子。
共通モデル	CIM スキーマの 2 つめの層であり、ドメイン固有であるがプラットフォームには依存しない一連のクラスを含む。ここでのドメインは、管理関連のデータ (システム、ネットワーク、アプリケーションなど) を意味する。共通モデルは、コアモデルを元に作成されたもの。 「拡張スキーマ」も参照。
クラス	類似したプロパティを持ち、類似した目的を果たすオブジェクトの集まり。
継承	クラスとインスタンスが親クラス (スーパークラス) からどのように派生するかを表す関係。クラスは、新しいサブクラス (子クラスとも言う) を生成できる。サブクラスは、その親クラスのメソッドとプロパティをすべて含む。継承は、WBEM 環境内で WBEM のクラスが実際の管理対象オブジェクトのテンプレートとしての役割を果たすために必要な機能の 1 つである。
コアモデル	CIM スキーマの最初の層であり、最上位レベルのクラス、およびそれらのプロパティと関連を含む。コアモデルは、ドメインにもプラットフォームにも依存しない。 「共通モデル」と「拡張スキーマ」も参照。

サブクラス	スーパークラスから派生するクラス。サブクラスはそのスーパークラスのすべての機能を継承するが、新しい機能を追加することも、あるいは既存の機能を定義し直すこともできる。
サブスキーマ	スキーマの一部であり、特定の編成によって所有される。サブスキーマには、Win32 スキーマ、Solaris スキーマなどがある。
参照	参照修飾子によって示される特殊な文字列のプロパティであり、ほかのインスタンスを指すポインタであることを示す。
修飾子	クラス、インスタンス、プロパティ、メソッド、またはパラメータを表現する情報を含む。修飾子には、3 つのカテゴリがあり、Common Information Model (CIM) によって定義されるもの、WBEM (標準の修飾子) によって定義されるもの、および開発者によって定義されるものに分類される。標準の修飾子は、CIM Object Manager によって自動的に付加される。
修飾子フレーバ	CIM 修飾子の属性の 1 つで、修飾子の使用を制御する。修飾子フレーバは、修飾子が派生クラスと派生インスタンスに影響するかどうか、および派生クラスまたは派生インスタンスが修飾子の本来の値を上書きできるかどうかを指定する規則について記述する。
集約関係	1 つのエンティティがいくつかのほかのエンティティの集合から構成されている関係。
推移的な依存性	少なくても 3 つの属性を持つ関係、R (A, B, C)。この場合、A は B を決定し、B は C を決定するが、B は A を決定しない。
スーパークラス	サブクラスの継承元であるクラス。
スキーマ	特定の環境における管理対象オブジェクトについて説明するクラス定義の集まり。
スコープ	CIM 修飾子の属性の 1 つであり、どの CIM 要素がその修飾子を使用できるかを示す。スコープを定義できるのは Qualifier Type の宣言内だけであり、修飾子内では変更できない。
静的クラス	定義が永続的な WBEM クラス。定義は、明示的に削除されるまで CIM Object Manager Repository に格納される。CIM Object Manager は、プロバイダの支援を受けずに静的クラスの定義を提供

できる。静的クラスは、静的インスタンス、動的インスタンスのどちらでもサポートできる。

静的インスタンス　CIM Object Manager Repository に永続的に格納されるインスタンス。

選択的な継承　下位クラスが上位クラスのプロパティを削除または上書きできること。

多重度　特定のエンティティの属性に適用できる値の数。

多相性　名前またはインタフェースを変更することなく、派生クラス内のメソッドとプロパティを変更できること。たとえば、サブクラスは、そのスーパークラスから継承されたメソッドまたはプロパティの実装を再定義できる。そのため、プロパティまたはメソッドは、スーパークラスがインタフェースクラスとして使用される場合でも再定義される。

したがって、LogicalDevice クラスは変数の状態を文字列として定義し、値として「オン」または「オフ」を返すことができる。LogicalDevice の Modem サブクラスは、「オン」、「オフ」、および「接続中」を返すことによって、状態を再定義 (オーバーライド) できる。LogicalDevice がすべて列挙される場合、その時点でモデムの役割を務める LogicalDevice はすべて、状態プロパティに対して「接続中」の値を返すことができる。

単一クラス　単一インスタンスだけをサポートする WBEM クラス。

動的インスタンス　必要時にプロバイダから提供されるインスタンスであり、CIM Object Manager Repositoty には格納されない。動的インスタンスは、静的クラス、動的クラスのどちらからも提供される。クラスのインスタンスを動的にサポートすることにより、プロバイダは最新のプロパティ値を提供できる。

動的クラス　必要に応じて実行時にプロバイダから定義が提供されるクラス。動的クラスはプロバイダ固有の管理対象オブジェクトを表現するのに使用され、CIM Object Manager Repository に永続的に格納されることはない。代わりに、動的クラスを提供するプロバイダがその場所についての情報を格納する責任を持つ。アプリケーションが動的クラスを要求する場合、CIM Object Manager はそのプロバイダを

	見つけて要求を送る。動的クラスがサポートするのは、動的インスタンスだけである。
ドメイン	プロパティまたはメソッドが属するクラス。たとえば、状態が Logical Device のプロパティの場合、Logical Device ドメインに属すると考えられる。
トリガー	クラスインスタンスの状態変化 (作成、削除、更新、アクセスなど)、およびプロパティの更新またはアクセスが発生すること。WBEM の実装には、トリガーを表現する明示的なオブジェクトは存在しない。トリガーは、システムの基本オブジェクトに対するオペレーション (クラス、インスタンス、およびネームスペースの作成、削除、変更) が起るか、または管理環境内のイベントによっては特定の指定をしなくても動作する。
名前付き要素	メタスキーマ内でオブジェクトとして表現できる実体。
ネームスペース	クラス、インスタンス、およびほかのネームスペースを含むことができる、ディレクトリに似た構造。
範囲	参照プロパティによって参照されるクラス。
必須プロパティ	値が必要なプロパティ。
標準スキーマ	システム、ネットワーク、またはアプリケーションの現在のオペレーション状態を表現する各種のクラスの編成と関連付けを行う、概念的な共通の枠組み。標準スキーマは、Common Information Model (CIM) では Distributed Management Task Force (DMTF) によって定義される。
フレーバ	「修飾子フレーバ」を参照。
プロパティ	クラスインスタンスの特性を示すために使用される値。プロパティ名は数字で開始することはできず、空白を含めることもできない。プロパティ値は、有効な Managed Object Format (MOF) のデータ型でなければならない。
プロパティプロバイダ	さまざまなソース (Solaris オペレーティング環境、Simple Network Management Protocol (SNMP) デバイスなど) のデータとイベント通知にアクセスするために管理対象オブジェクトと通信を行うプロ

グラム。プロバイダは、統合と解釈を行うためにこの情報を CIM Object Manager に送る。

別名　Managed Object Format (MOF) ファイルの別の場所にあるオブジェクトに対するクラス宣言またはインスタンス宣言内のシンボリック参照。別名は、インスタンス名およびクラス名と同じ規則に従う。別名は、一般に比較的長いパスのショートカットとして使用される。

メソッド　あるクラスの動作について述べた関数。クラスにメソッドを組み込んでも、メソッドの実装を保証することにはならない。

メタスキーマ　Common Information Model の公式の定義であり、このモデルの内容、使用法、および意味の説明に使用される用語を定義する。

メタモデル　管理対象オブジェクトの表現とエンティティと関係について説明する CIM コンポーネント。たとえば、クラス、インスタンス、アソシエーションなど。

列挙　オブジェクトリストの取得を意味する Java 用語。Java は、オブジェクトリストを列挙するメソッドが含まれる `Enumeration` インタフェースを提供する。このリスト上の列挙される個々のオブジェクトは、要素と呼ばれる。

索引