

HP Universal CMDB

Windows および Solaris オペレーティング・システム用

ソフトウェア・バージョン : 8.00

インテグレーション

ドキュメント・リリース日 : 2009 年 1 月 (英語版)

ソフトウェア・リリース日 : 2009 年 1 月 (英語版)



利用条件

保証

HP の製品およびサービスの保証は、かかる製品およびサービスに付属する明示的な保証の声明において定められている保証に限ります。本ドキュメントの内容は、追加の保証を構成するものではありません。HP は、本ドキュメントに技術的な間違いまたは編集上の間違い、あるいは欠落があった場合でも責任を負わないものとします。

本ドキュメントに含まれる情報は、事前の予告なく変更されることがあります。

制限事項

本コンピュータ・ソフトウェアは、機密性があります。これらを所有、使用、または複製するには、HP からの有効なライセンスが必要です。FAR 12.211 および 12.212 に従って、商用コンピュータ・ソフトウェア、コンピュータ・ソフトウェアのドキュメント、および商用アイテムの技術データは、HP の標準商用ライセンス条件に基づいて米国政府にライセンスされています。

サードパーティ Web サイト

HP は、補足情報の検索に役立つ外部サードパーティ Web サイトへのリンクを提供します。サイトの内容と利用の可否は予告なしに変更される場合があります。HP は、サイトの内容または利用の可否について、いかなる表明も保証も行いません。

著作権

© Copyright 2005 - 2009 Mercury Interactive (Israel) Ltd.

商標

Adobe® および Acrobat® は、Adobe Systems Incorporated の商標です。

Intel® Pentium®, および Intel® Xeon™ は、米国およびその他の国における Intel Corporation またはその子会社の商標または登録商標です。

Java™ は、Sun Microsystems, Inc. の米国商標です。

Microsoft®, Windows®, Windows NT® および Windows® XP は、Microsoft Corporation の米国登録商標です。

Oracle® は、カリフォルニア州レッドウッド市の Oracle Corporation の米国登録商標です。

Unix® は、The Open Group の登録商標です。

文書の更新

本書のタイトル・ページには、次の識別情報が含まれています。

- ソフトウェアのバージョンを示すソフトウェア・バージョン番号
 - ドキュメントが更新されるたびに更新されるドキュメント発行日
 - 本バージョンのソフトウェアをリリースした日付を示す、ソフトウェア・リリース日付
- 最新のアップデートまたはドキュメントの最新版を使用していることを確認するには、次の URL にアクセスしてください：

<http://h20230.www2.hp.com/selfsolve/manuals>

このサイトでは、HP Passport に登録してサインインする必要があります。HP Passport ID の登録は、次の URL にアクセスしてください：

<http://h20229.www2.hp.com/passport-registration.html>

または、HP Passport のログイン・ページの [**New users - please register**] リンクをクリックしてください。

適切な製品サポート・サービスに登録すると、更新情報や最新情報も入手できます。詳細については HP の営業担当にお問い合わせください。

サポート

HP ソフトウェアのサポート Web サイトは、次の場所にあります。

<http://support.openview.hp.com>

この Web サイトでは、連絡先情報と、HP ソフトウェアが提供する製品、サービス、およびサポートについての詳細が掲載されています。

HP ソフトウェア・オンライン・ソフトウェア・サポートでは、お客様にセルフ・ソルブ機能を提供しています。ビジネス管理に必要な、インタラクティブなテクニカル・サポート・ツールに迅速かつ効率的にアクセスできます。有償サポートをご利用のお客様は、サポート・サイトの以下の機能をご利用いただけます。

- 関心のある内容の技術情報の検索
- サポート・ケースおよび機能強化要求の提出および追跡
- ソフトウェア・パッチのダウンロード
- サポート契約の管理
- HP サポートの連絡先の表示
- 利用可能なサービスに関する情報の確認
- ほかのソフトウェア顧客との議論に参加
- ソフトウェアのトレーニングに関する調査と登録

ほとんどのサポート・エリアでは、HP Passport ユーザとして登録し、ログインする必要があります。また、多くの場合、サポート契約も必要です。

HP Passport ID の登録は、次の場所で行います。

<http://h20229.www2.hp.com/passport-registration.html>

アクセス・レベルの詳細に関しては次を参照してください。

http://h20230.www2.hp.com/new_access_levels.jsp

目次

ようこそ	9
第 I 部 : HP Universal CMDB API	
第 1 章 : API の概要	13
API の概要	13
第 2 章 : HP Universal CMDB Web サービス API	15
表記規則	16
HP Universal CMDB Web Service API の使用方法	16
HP Universal CMDB Web サービス API の参考情報	18
明確なトポロジ・マップ要素を返す	19
Web サービスの呼び出し	22
UCMDB への問い合わせ	23
UCMDB の更新	27
UCMDB クラス・モデルの問い合わせ	29
影響分析のための問い合わせ	31
UCMDB クエリ・メソッド	32
UCMDB 更新メソッド	47
UCMDB の影響分析メソッド	50
使用例	53
例	54
UCMDB の一般的なパラメータ	85
UCMDB 出力パラメータ	89
第 3 章 : HP Universal CMDB Java API	91
規則	92
HP Universal CMDB Java API の使用	92
アプリケーションの一般的な構造	94
API Jar ファイルの取得	95
インテグレーション・ユーザの作成	95
HP Universal CMDB Java API リファレンス	96
使用例	96
例	98

第 4 章： ディスカバリおよび依存関係マップ Web サービス API 103

第 II 部： フェデレーションと調整

第 5 章： フェデレート CMDB の概要	107
フェデレート CMDB – 概要.....	108
アダプタ	108
複数のデータ・ストアからのデータの取得	109
外部データ・ストアからの属性の取得.....	111
マッピング情報	112
フェデレート・データを使った作業 – ワークフロー	112
フェデレート・アダプタの暗号化パスワードの変更	114
フェデレート CMDB のユーザ・インタフェース	115
第 6 章： 汎用データベース・アダプタ	131
データベース・アダプタの概要	132
サポートしていない TQL クエリ.....	133
調整	134
JPA プロバイダとしての Hibernate	136
データベース・アダプタのデプロイ – 最小メソッド	138
データベース・アダプタのデプロイ – 詳細メソッド	144
フェデレート・データベース構成ファイル	168
adapter.conf ファイル	169
simplifiedConfiguration.xml ファイル	170
orm.xml ファイル.....	177
reconciliation_rules.txt ファイル.....	182
transformations.txt ファイル.....	182
persistence.xml ファイル	183
discriminator.properties ファイル.....	185
replication_config.txt ファイル.....	186
fixed_values.txt ファイル.....	186
用意済みのコンバータ	186
プラグイン.....	190
設定例	190
フェデレート・データベース・ログ・ファイル	203
外部参照	205
トラブルシューティングと制限事項	206
第 7 章： Federation Framework SDK	209
Federation Framework – 概要.....	210
アダプタおよびマッピングの Federation Framework とのやり取り	213
FTQL 用の Federation Framework フロー	214
レプリケーション用の Federation Framework フロー	225
HP Release Control フェデレーション・アダプタ	226

アダプタ・インタフェース	230
新しい外部データ・ストア用のアダプタの追加	231
アダプタ機能	241
第 8 章： HP ServiceCenter/Service Manager Adapter	243
アダプタの使用方法	244
アダプタ構成ファイル	245
アダプタのデプロイ	254
ServiceDesk アダプタのデプロイ	255
ServiceCenter/Service Manager CIT への属性の追加	262
第 9 章： トラブルシューティングと制限事項	271
フェデレート CMDB のトラブルシューティングと制限事項	271
第 10 章： 調整の概要	275
調整 - 概要	275
ホスト調整ルール	275
クラスタ調整ルール	276
ソフトウェア要素調整ルール	277

第 III 部： インテグレーション

第 11 章： ダイレクト・リンクを使った UCMDB アプレットの	
埋め込み	281
ダイレクト・リンクを使った UCMDB アプレットの埋め込み	282
UCMDB アプレット・タブ - 概要	283
ダイレクト・リンクの操作フロー	284
索引	291

ようこそ

本書では、HP Universal CMDB で使用できる各種の統合について説明します。

本章の内容

- ▶ 本書の構成（9 ページ）
- ▶ 対象読者（10 ページ）
- ▶ 詳細情報の入手（10 ページ）

本書の構成

本書は、次の各部で構成されています。

第 I 部 HP Universal CMDB API

CMDB API を使用して HP Universal CMDB からデータを抽出する方法について説明します。

第 II 部 フェデレーションと調整

アダプタを定義して、データの制御はデータ・ソースが引き続き行い、ほかのソースから CMDB にデータを取り込む方法について説明します。

第 III 部 インテグレーション

直接リンクを使用してアプレットを外部アプリケーションに組み込む方法について説明します。

対象読者

本書は、次の HP Universal CMDB 利用者を対象としています。

- ▶ HP Universal CMDB 管理者
- ▶ HP Universal CMDB エンド・ユーザ
- ▶ HP Universal CMDB 統合開発者

本書の読者は、エンタープライズ・アプリケーションの操作と使用に精通し、HP Universal CMDB とエンタープライズ監視の概念および管理の概念を理解している必要があります。

詳細情報の入手

HP Universal CMDB に含まれているすべてのオンライン・ドキュメントの一覧、その他のオンライン・リソース、最新版のドキュメントの入手情報、本書で使用する表記規則については、『**HP Universal CMDB デプロイメント・ガイド**』（PDF）を参照してください。

第 I 部

HP Universal CMDB API

第 1 章

API の概要

本章では、HP Universal CMDB に含まれている API について説明します。

本章の内容

概念

- ▶ API の概要（13 ページ）

API の概要

HP Universal CMDB には次の API が含まれています。

- ▶ **UCMDB API** : UCMDB (Universal Configuration Management database) に構成アイテム定義やトポロジ関係を記述できます。また、TQL やアドホック・クエリを使用して情報をクエリすることもできます。詳細については、15 ページ「HP Universal CMDB Web サービス API」を参照してください。
- ▶ **UCMDB Java API** : サードパーティー製ツールまたはカスタム・ツールで Java API を使用して、データや計算を抽出したり UCMDB (Universal Configuration Management database) にデータを書き込む方法について説明します。詳細については、91 ページ「HP Universal CMDB Java API」を参照してください。
- ▶ **DDM Web サービス** : サードパーティー製ツールまたはカスタム・ツールで HP Discovery and Dependency Mapping Web Service を使用して ディスカバリおよび依存関係マップ (DDM) を管理する方法について説明します。詳細については、『ディスカバリおよび依存関係マップ』の「HP Discovery and Dependency Mapping Web Service API」を参照してください。

第 2 章

HP Universal CMDB Web サービス API

本章では、サードパーティ製ツールやカスタム・ツールで HP Universal CMDB Web Service API を使用して、データの抽出と計算を行い、データを UCMDB (Universal Configuration Management database) に書き込む方法について説明します。

本章は、オンラインのドキュメント・ライブラリで入手できる UCMDB スキーマに関するドキュメントと併せてご利用ください。

本章の内容

概念

- ▶ 表記規則 (16 ページ)
- ▶ HP Universal CMDB Web Service API の使用方法 (16 ページ)
- ▶ HP Universal CMDB Web サービス API の参考情報 (18 ページ)
- ▶ 明確なトポロジ・マップ要素を返す (19 ページ)

タスク

- ▶ Web サービスの呼び出し (22 ページ)
- ▶ UCMDB への問い合わせ (23 ページ)
- ▶ UCMDB の更新 (27 ページ)
- ▶ UCMDB クラス・モデルの問い合わせ (29 ページ)
- ▶ 影響分析のための問い合わせ (31 ページ)

参照先

- ▶ UCMDB クエリ・メソッド (32 ページ)
- ▶ UCMDB 更新メソッド (47 ページ)

- ▶ UCMDB の影響分析メソッド (50 ページ)
- ▶ 使用例 (53 ページ)
- ▶ 例 (54 ページ)
- ▶ UCMDB の一般的なパラメータ (85 ページ)
- ▶ UCMDB 出力パラメータ (89 ページ)

表記規則

本章では、次の表記規則を使用します。

- ▶ **UCMDB** は、Universal Configuration Management database 自体を指します。
HP Universal CMDB は、アプリケーションを指します。
- ▶ UCMDB 要素とメソッド引数は、スキーマで指定したのと同じように大文字と小文字を区別して入力します。メソッドの要素または引数は大文字にしません。たとえば、`relation` は、メソッドに渡される **Relation** タイプの要素です。

HP Universal CMDB Web Service API の使用方法

HP Universal CMDB Web Service API は、アプリケーションを Universal CMDB (UCMDB) に統合するために使用します。この API により、次を実施するメソッドが提供されます。

- ▶ CMDB での CI と関係の追加、削除、および更新
- ▶ クラス・モデルに関する情報の取得
- ▶ 影響分析の取得
- ▶ 構成アイテムおよび関係に関する情報の取得

構成アイテムと関係に関する情報を取得するメソッドでは、一般的にトポロジ・クエリ言語 (TQL) を使用します。詳細については、『**モデル管理**』の「トポロジ・クエリ言語」を参照してください。

HP Universal CMDB Web サービス API のユーザは、次に関する知識が必要です。

- ▶ SOAP の仕様
- ▶ オブジェクト指向プログラミング言語 (C++ や C#, Java など)
- ▶ HP Universal CMDB

本項の内容

- ▶ 17 ページ「API の使用」
- ▶ 18 ページ「権限」

API の使用

API を使用すると、多くのビジネス要件を満たすことができます。次に例を示します。

- ▶ サードパーティ製のシステムは、利用できる構成アイテム (CI) に関する情報をクラス・モデルに問い合わせることができます。
- ▶ サードパーティ製のアセット管理ツールは、そのツールのみで利用できる情報を使って UCMDB を更新できるため、アセット管理ツールのデータを HP アプリケーションで収集したデータと統一できます。
- ▶ 多くのサードパーティ製のシステムは、UCMDB にデータを入力して、変更内容を追跡し影響分析を実行できる中心的な UCMDB を作成できます。
- ▶ サードパーティ製のシステムは、ビジネス・ロジックに従ってエンティティと関係を作成し、データを UCMDB に書き込んで UCMDB のクエリ機能を活用できます。
- ▶ Change Control Management (CCM) システムなど、ほかのシステムは影響分析手法を使用して変更の分析を行えます。

権限

管理者は、Web サービスに接続するためのログイン証明書を提供します。必要な資格情報は、ユーザが HP Universal CMDB をスタンドアロンのアプリケーションとして使用するか、HP Business Availability Center 内から使用するかによって異なります。

- ▶ **HP Universal CMDB をスタンドアロンで使用する場合**：ディスカバリ・リソースに対する権限を付与されている DDM ユーザの資格情報を使用してログインします。

詳細については、『モデル管理』の「[セキュリティ マネージャ] ウィンドウ」を参照してください。

- ▶ **HP Business Availability Center に組み込まれた HP Universal CMDB を使用する場合**：Business Availability Center ユーザの資格情報を使用してログインします。ユーザは、Business Availability Center の HP Universal CMDB リソースに対して関連する権限を付与されている必要があります。

HP Universal CMDB Web サービス API の参考情報

要求と応答の構造に関する完全なドキュメントについては、HP UCMDB Web Service API の参考情報を参照してください。これらのファイルは、次のフォルダにあります。

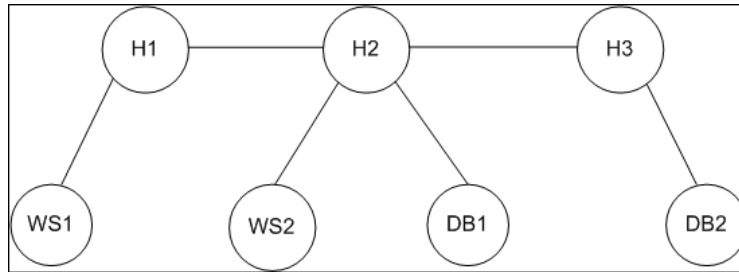
```
¥¥ < HP Universal CMDB ルート・ディレクトリ > ¥UCMDBServer¥j2f¥  
AppServer¥webapps¥site.war¥amdocs¥eng¥doc_lib¥Integrations¥CMDB_Schema¥  
webframe.html
```

🔗 明確なトポロジ・マップ要素を返す

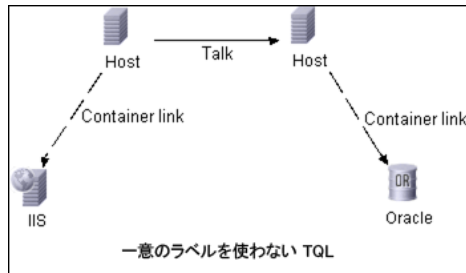
`topology` または `topologyMap` 要素のデータを返すクエリ・メソッドは、システムを検索して TQL クエリとの一致を探します。次の図は、結果として得られる `topology` と `topologyMap` の構造が、クエリで一意的なラベルを使用すると、どのような影響を受けるかを示しています。

ラベルは、特定の設定における関係および構成アイテムに対して、クエリでユーザが指定した名前です。クエリで指定したラベルは、返されるマップでノード・ラベルとして使用されます。ラベルが指定されていない場合は、`CI` または `Relation` タイプ名が結果として得られるマップでラベルとして使用されます。次の例では、`IISHost` ラベルと `DBHost` ラベルを標準設定の `Host` ラベルの代わりに指定し、`ContainerIIS` ラベルと `ContainsDB` ラベルを標準設定の `Container Link` ラベルの代わりに指定しています。

次の例は、小規模な IT ユニバース・モデルを示します。ここでは、`H1`、`H2`、`H3` という 3 つのホストがあり、`Web` サーバ (`WS`) とデータベース・マネージャ (`DB`) をホストしています。`WS1` は `H1` 上に存在します。`DB1` と `WS2` は、どちらも `H2` 上に存在します。`DB2` は `H3` 上に存在します。



このクエリは、標準設定のラベルを使用して定義します。



この TQL クエリを IT ユニバースで実行した結果、Topology または TopologyMap 要素が得られます。

トポロジの応答

```
Cls : H1, H2, H3, WS1, WS2, DB1, DB2  
Relations : H1-WS1, H1-H2, H2-H3, WS2-H2, DB1-H2, DB2-H3
```

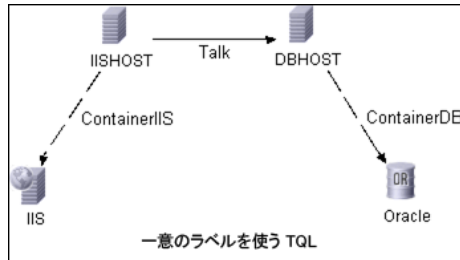
TopologyMap の応答

```
CINode :  
  label : Host  
  Cls : H1, H2  
  
CINode :  
  label : Host  
  Cls : H2, H3  
  
CINode :  
  label : DB  
  Cls : DB1, DB2  
  
CINode :  
  label : Webserver  
  Cls : IIS  
  
relationNode :  
  label : talk  
  relations : H1-H2, H2-H3  
  
relationNode :  
  label : Container Link  
  relations : WS1-H1, WS2-H2  
  
relationNode :  
  label : Container Link  
  relations : DB2-H3, DB1-H2
```

前述の TopologyMap の応答では、最初の2つの CINode に同一の Host ラベルが含まれ、クエリ内にある2つの Host CI に対応しています。これらの CINodes の両方に host H2 が含まれ、H2 が重複する理由は示されていません。

最後の 2 つの **relationNode** には、同一の **Contained** ラベルが含まれ、クエリ内の 2 つの **Container link** 関係に対応しています。

重複が発生するのは、一意のラベルがクエリで指定されなかったためで、その結果、マップ内で標準設定のラベル（タイプ名は、**Host** と **Container**）を使用したことが原因です。もっと使いやすいマップを抽出するには、次のクエリに示すように、各設定が一致するように、一意のラベルを使ってクエリを定義します。



topology の結果は、一意のラベルを使わない **TQL** のものと同一です。しかし、**topologyMap** の結果は異なります。各ラベルが一意になります。

```

CINode :
  label : IISHOST
  CIs : H1, H2

CINode :
  label : DBHOST
  CIs : H2, H3

...

relationNode :
  label : ContainerIIS
  relations : WS1-H1, WS2-H2

relationNode :
  label : ContainerDB
  relations : DB2-H3, DB1-H2
  
```

このマップでは、H2 が 2 度返された理由は明らかです。一意のラベルは、H2 が Web サーバ・ホストとして 1 度、データベース・ホストとして 1 度返されたことを示しています。

ヒント : UCMDB で可能な場合には、一意のユーザ定義ラベルを特定の設定に適用します。

Web サービスの呼び出し

HP Universal CMDB Web サービスで標準の SOAP プログラミング技術を使用すると、サーバ側のメソッドを呼び出すことができます。ステートメントを解析できない場合、またはメソッドの呼び出しに問題がある場合は、API メソッドにより、**SoapFault** 例外がスローされます。**SoapFault** 例外がスローされると、UCMDB によってエラー・メッセージ、エラー・コード、および例外メッセージ・フィールドの 1 つ以上にデータが入力されます。エラーがなければ、呼び出しの結果が返されます。

SOAP プログラマは、以下のアドレスで WSDL にアクセスできます。

[http:// <サーバ> \[:port\]/axis2/services/UcmdbService?wsdl](http://<サーバ>[:port]/axis2/services/UcmdbService?wsdl)

ポートの指定は、標準とは異なる設定でインストールされている場合のみ必要です。正しいポート番号についてはシステム管理者に問い合わせてください。

サービスを呼び出すための URL は、次のとおりです。

[http:// <サーバ> \[:port\]/axis2/services/UcmdbService](http://<サーバ>[:port]/axis2/services/UcmdbService)

たとえば、UCMDB への接続例については、53 ページ「使用例」を参照してください。

UCMDB への問い合わせ

UCMDB に問い合わせるには、32 ページ「UCMDB クエリ・メソッド」で説明した API を使用します。

クエリと返される UCMDB 要素には、常に実際の UCMDB ID が含まれています。

クエリ・メソッドの使用例については、58 ページ「クエリの例」を参照してください。

本項の内容

- ▶ 23 ページ「実行時の応答計算」
- ▶ 24 ページ「サイズの大きい応答の処理」
- ▶ 24 ページ「返されるプロパティの指定」
- ▶ 25 ページ「コンクリート (Concrete) プロパティ」
- ▶ 26 ページ「派生 (Derived) プロパティ」
- ▶ 26 ページ「命名 (Naming) プロパティ」
- ▶ 26 ページ「その他のプロパティ指定要素」

実行時の応答計算

すべてのクエリ・メソッドについて、要求を受信したときに、クエリ・メソッドによって要求された値が UCMDB サーバによって計算され、最新のデータに基づいて結果が返されます。結果は、TQL クエリがアクティブで、以前計算した結果が存在する場合でも、要求を受信したときに必ず計算されます。このため、クライアント・アプリケーションに返されるクエリの実行結果は、ユーザ・インタフェースに表示される同じクエリの結果とは異なる場合があります。

ヒント：アプリケーションで特定のクエリの結果を複数回使用し、結果データの使用のたびにデータが大きく変わらないことが期待される場合は、同じクエリを繰り返し実行するのではなく、クライアント・アプリケーションにデータを保存することによってパフォーマンスを向上できます。

サイズの大きい応答の処理

クエリに対する応答には、実際のデータは転送されない場合でも、クエリ・メソッドによって要求されたデータの構造が常に含まれます。データがコレクションまたはマップである多くのメソッドについて、応答には **ChunkInfo** 構造も含まれています。これは、**chunksKey** と **numberOfChunks** から構成されています。**numberOfChunks** フィールドは、取得する必要のあるデータを含むチャンクの数を示します。

データの最大転送サイズは、システム管理者が設定します。クエリから返されるデータが最大サイズより大きい場合は、最初の応答のデータ構造には意味のある情報は含まれず、**numberOfChunks** フィールドは 2 以上になります。データが最大サイズより大きくない場合、**numberOfChunks** フィールドは 0 (ゼロ) になり、データは最初の応答時に転送されます。このため、応答を処理するときには、**numberOfChunks** 値を最初にチェックしてください。この値が 1 より大きい場合は、転送されたデータを破棄し、データのチャンクを要求します。この値が 1 を超えていない場合は、応答に含まれているデータを使用します。

チャンクに分割したデータの処理の詳細については、45 ページ「pullTopologyMapChunks」および 46 ページ「releaseChunks」を参照してください。

返されるプロパティの指定

CI と関係には、一般的に多くのプロパティがあります。これらのアイテムのコレクションまたはグラフを返す一部のメソッドは、クエリに一致する各アイテムについて、返すプロパティ値を指定する入力パラメータを受け付けます。UCMDB は、空のプロパティを返しません。このため、クエリに対する応答では、クエリで指定したよりもプロパティが少ないことがあります。

本項では、返されるプロパティを指定するために使用するセットの種類について説明します。

プロパティを参照するには、次の 2 つの方法があります。

- ▶ 名前を使う。
- ▶ 事前に定義したプロパティ・ルールの名前を使う。事前に定義したプロパティ・ルールは、実際のプロパティ名のリストを作成するために UCMDB によって使用されます。

アプリケーションが名前を使ってプロパティを参照する場合、アプリケーションによって **PropertiesList** 要素が渡されます。

ヒント：可能な場合には、ルール・ベースのセットではなく **PropertiesList** を使用して、必要なプロパティの名前を指定してください。事前に定義したプロパティ・ルールを使用すると、ほぼ必ず必要以上のプロパティが返されるため、パフォーマンスが低下します。

事前定義したプロパティには、修飾子 (**qualifier**) プロパティとシンプル (**simple**) プロパティの 2 種類があります。

- ▶ **修飾子 (Qualifier) プロパティ**：クライアント・アプリケーションが **QualifierProperties** 要素 (プロパティに適用できる修飾子のリスト) を渡す必要がある場合に使用します。クライアント・アプリケーションによって渡された修飾子のリストは、UCMDB によって、修飾子の少なくとも 1 つを適用するプロパティのリストに変換されます。これらのプロパティの値は、**CI** または **Relation** 要素とともに返されます。
- ▶ **シンプル (Simple) プロパティ**：シンプルなルール・ベースのプロパティを使用するには、クライアント・アプリケーションは、**SimplePredefinedProperty** または **SimpleTypedPredefinedProperty** 要素を渡します。これらの要素には、返すプロパティのリストを UCMDB が生成するのに使うルールの名前が含まれています。**SimplePredefinedProperty** 要素または **SimpleTypedPredefinedProperty** 要素で指定できるルールは、**CONCRETE**、**DERIVED**、および **NAMING** です。

コンクリート (Concrete) プロパティ

コンクリート (**Concrete**) プロパティは、指定した **CIT** に対して定義されたプロパティのセットです。派生クラスによって追加されたプロパティは、これらの派生クラスのインスタンスに対しては返されません。

メソッドによって返されるインスタンスのコレクションは、メソッド呼び出しで指定した **CIT** のインスタンス、およびその **CIT** から継承した **CIT** のインスタンスから構成されます。派生した **CIT** は、指定した **CIT** のプロパティを継承します。また、派生した **CIT** は、プロパティを追加することによって親 **CIT** を拡張します。

コンクリート (Concrete) プロパティの例

CIT T1 には、プロパティ P1 と P2 があります。CIT T11 は、T1 から継承し、T1 を、プロパティ P21 と P22 を使って拡張します。

T1 タイプの CI のコレクションには、T1 と T11 のインスタンスが含まれます。このコレクション内のすべてのインスタンスのコンクリート (concrete) プロパティは、P1 と P2 です。

派生 (Derived) プロパティ

派生 (Derived) プロパティは、指定した CIT に対して定義されたプロパティ、および派生 CIT ごとに、派生 CIT によって追加されたプロパティのセットです。

派生 (Derived) プロパティの例

コンクリート (concrete) プロパティの例から続けると、T1 のインスタンスの派生 (derived) プロパティは、P1 と P2 です。T11 のインスタンスの派生 (derived) プロパティは、P1、P2、P21、および P22 です。

命名 (Naming) プロパティ

命名 (naming) プロパティには、`display_label` と `data_name` があります。

その他のプロパティ指定要素

- ▶ **PredefinedProperties** : `PredefinedProperties` は、利用可能なほかのルールごとに、`QualifierProperties` 要素と `SimplePredefinedProperty` 要素を含むことができます。`PredefinedProperties` のセットには、すべての種類のリストが含まれている必要はありません。
- ▶ **PredefinedTypedProperties** : `PredefinedTypedProperties` は、異なるセットのプロパティを各 CIT に適用するために使用します。`PredefinedTypedProperties` は、利用可能なほかのルールごとに、`QualifierProperties` 要素と `SimpleTypedPredefinedProperty` 要素を含むことができます。`PredefinedTypedProperties` は、各 CIT に個々に適用されるため、派生 (derived) プロパティは関係ありません。`PredefinedProperties` のセットには、すべての適用可能な種類のリストが含まれている必要はありません。
- ▶ **CustomProperties** : `CustomProperties` は、基本的な `PropertiesList` とルールベースのプロパティ・リストの組み合わせを含むことができます。プロパティ・フィルタは、すべてのリストによって返されるすべてのプロパティを結合したものです。

- ▶ **CustomTypedProperties** : CustomTypedProperties は、基本的な PropertiesList と適用可能なルール・ベースのプロパティ・リストの組み合わせを含みます。プロパティ・フィルタは、すべてのリストによって返されるすべてのプロパティを結合したものです。
- ▶ **TypedProperties** : TypedProperties は、CIT ごとに異なるセットのプロパティを渡すために使用します。TypedProperties は、タイプ名と、すべてのタイプのプロパティ・セットから構成されるペアのコレクションです。各プロパティ・セットは、対応するタイプのみ適用されます。

UCMDB の更新

UCMD の更新は、更新 API を使って実施します。API メソッドの詳細については、47 ページ「UCMDB 更新メソッド」を参照してください。

更新メソッドの使用例については、75 ページ「更新の例」を参照してください。

本項の内容

- ▶ 27 ページ「UCMDB の更新パラメータ」
- ▶ 28 ページ「更新メソッドを使った ID タイプの使用」
- ▶ 47 ページ「UCMDB 更新メソッド」

UCMDB の更新パラメータ

このトピックでは、サービスの更新メソッドによってのみ使用されるパラメータについて説明します。詳細については、スキーマのドキュメントを参照してください。

CIsAndRelationsUpdates

CIsAndRelationsUpdates タイプは、CIsForUpdate、relationsForUpdate、referencedRelations、および referencedCIs から構成されます。

CIsAndRelationsUpdates インスタンスには、3つの要素がすべて含まれている必要はありません。

`CIsForUpdate` は、CI コレクションです。`relationsForUpdate` は、`Relations` コレクションです。コレクション内の CI と `relation` 要素には、`props` 要素があります。CI または関係を作成するときは、**必須**の属性または**キー**属性を CI タイプの定義に持っているプロパティに値を指定する必要があります。これらのコレクション内のアイテムは、メソッドによって更新または作成されます。

`referencedCIs` および `referencedRelations` は、すでに UCMDb に定義されている CI のコレクションです。コレクション内の要素は、すべてのキー・プロパティとともに一時 ID を使って識別されます。これらのアイテムは、更新するために CI と関係の識別に使用します。これらは、メソッドによって作成、更新されることはありません。

これらのコレクション内の各 CI 要素と `relation` 要素は、プロパティのコレクションを持っています。新しいアイテムは、これらのコレクション内のプロパティ値を使って作成されます。

更新メソッドを使った ID タイプの使用

次に、ID CIT、および CI と関係について説明します。ID が実際の UCMDb ID でない場合、タイプ属性とキー属性が必要になります。

構成アイテムの削除と更新

アイテムを削除または更新するメソッドの呼び出し時に、一時 ID または空の ID が、クライアントによって使用されることがあります。この場合、CI を識別する CI タイプとキー属性を設定する必要があります。

関係の削除と更新

関係を削除または更新する場合、関係 ID は空、一時、または本物のいずれでもかまいません。

CI の ID が一時の場合、CI を `referencedCIs` コレクションで渡して、そのキー属性を指定する必要があります。詳細については、27 ページ「`CIsAndRelationsUpdates`」の `referencedCIs` を参照してください。

UCMDb への新しい構成アイテムの挿入

空の ID または一時 ID を使用して新しい CI を挿入することができます。しかし、ID が空の場合、`clientID` がないため、サーバは `createIdsMap` 構造内の実際の UCMDb ID を返すことができません。詳細については、47 ページ「`addCIsAndRelations`」と 32 ページ「UCMDb クエリ・メソッド」を参照してください。

UCMDB への新しい関係の挿入

関係 ID は、一時的または空です。しかし、関係が新しく、関係のいずれか一端の構成アイテムが UCMDB ですでに定義されている場合、すでに存在するこれらの CI は、実際の UCMDB ID によって識別するか、または `referencedCIs` コレクションで指定する必要があります。

UCMDB クラス・モデルの問い合わせ

クラス・モデル・メソッドは、CIT と関係に関する情報を返します。クラス・モデルは、CI タイプ・マネージャを使用して設定します。詳細については、『**モデル管理**』の「CI タイプ・マネージャ」を参照してください。

クラス・モデル・メソッドの使用例については、79 ページ「クラス・モデルの例」を参照してください。

本項では、CIT と関係に関する情報を返す、次のメソッドに関する情報を提供します。

- ▶ 29 ページ 「`getClassAncestors`」
- ▶ 30 ページ 「`getAllClassesHierarchy`」
- ▶ 30 ページ 「`getCmdbClassDefinition`」

`getClassAncestors`

`getClassAncestors` メソッドは、特定の CIT とそのルート間のパスを取得します（ルートを含む）。

入力

パラメータ	コメント
<code>cmdbContext</code>	詳細については、85 ページ「 <code>CmdbContext</code> 」を参照してください。
<code>className</code>	タイプ名。詳細については、87 ページ「タイプ名」を参照してください。

出力

パラメータ	コメント
classHierarchy	クラス名と親クラス名のペアのコレクション。
comments	内部使用専用。

 **getAllClassesHierarchy**

getAllClassesHierarchy メソッドは、クラス・モデル・ツリー全体を取得します。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。

出力

パラメータ	コメント
classesHierarchy	クラス名と親クラス名のペアのコレクション。
comments	内部使用専用。

 **getCmdbClassDefinition**

getCmdbClassDefinition メソッドは、指定したクラスに関する情報を取得します。

getCmdbClassDefinition を使用してキー属性を取得する場合、基本クラスとともに親クラスも問い合わせる必要があります。getCmdbClassDefinition は、className によって指定したクラス定義で設定した ID_ATTRIBUTE を持つ属性のみをキー属性として識別します。継承したキー属性は、指定したクラスのキー属性として認識されません。このため、指定したクラスのキー属性の完全なリストは、クラスとそのすべての親のキーをすべて結合したものです（ルートを含む）。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
className	タイプ名。詳細については、87 ページ「タイプ名」を参照してください。

出力

パラメータ	コメント
cmdbClass	name, classType, displayLabel, description, parentName, 修飾子, および属性から構成されるクラス定義。
comments	内部使用専用。

影響分析のための問い合わせ

影響分析メソッドの **識別子** は、サービスの応答データをポイントします。識別子は現在の応答に固有のもので、10 分間使用されなければ、サーバのメモリ・キャッシュから破棄されます。

影響分析メソッドの使用例については、81 ページ「影響分析の例」を参照してください。

UCMDB クエリ・メソッド

本項では、次のメソッドに関する情報を提供します。

- ▶ 32 ページ 「executeTopologyQueryByName」
- ▶ 33 ページ 「executeTopologyQueryByNameWithParameters」
- ▶ 34 ページ 「executeTopologyQueryWithParameters」
- ▶ 35 ページ 「getChangedCIs」
- ▶ 36 ページ 「getCINeighbours」
- ▶ 37 ページ 「getCIsByID」
- ▶ 38 ページ 「getCIsByType」
- ▶ 39 ページ 「getFilteredCIsByType」
- ▶ 43 ページ 「getQueryNameOfView」
- ▶ 44 ページ 「getTopologyQueryExistingResultByName」
- ▶ 44 ページ 「getTopologyQueryResultCountByName」
- ▶ 45 ページ 「pullTopologyMapChunks」
- ▶ 46 ページ 「releaseChunks」

executeTopologyQueryByName

executeTopologyQueryByName メソッドは、指定したクエリに一致するトポロジ・マップを取得します。

ヒント：マップには多くの情報が含まれています。TQL 内の各 CINode と各 relationNode のラベルが一意であれば、理解しやすくなります。詳細については、19 ページ「明確なトポロジ・マップ要素を返す」を参照してください。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
queryName	マップの取得に使う UCMDDB 内の TQL の名前。
queryTypedProperties	特定の構成アイテム・タイプのアイテムを取得するための、プロパティのセットのコレクション。

出力

パラメータ	コメント
topologyMap	詳細については、90 ページ「TopologyMap」を参照してください。

executeTopologyQueryByNameWithParameters

`executeTopologyQueryByNameWithParameters` メソッドは、指定したパラメータ化されたクエリに一致する `topologyMap` 要素を取得します。

クエリ・パラメータの値は、`parameterizedNodes` 引数で渡されます。指定した TQL には、各 `CINode` および各 `relationNode` に対して一意のラベルが定義されている必要があります。定義されていないければ、メソッドの呼び出しは失敗します。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
queryName	マップを取得する UCMDDB 内のパラメータ化された TQL の名前。

パラメータ	コメント
parameterizedNodes	クエリ結果の対象となるために各ノードが満たす必要がある条件。
queryTypedProperties	特定の構成アイテム・タイプのアイテムを取得するための、プロパティのセットのコレクション。

出力

パラメータ	コメント
topologyMap	詳細については、90 ページ「TopologyMap」を参照してください。
chunkInfo	詳細については、90 ページ「ChunkInfo」と 24 ページ「サイズの大きい応答の処理」を参照してください。

executeTopologyQueryWithParameters

`executeTopologyQueryWithParameters` メソッドは、指定したパラメータ化されたクエリに一致する `topologyMap` 要素を取得します。

クエリは、`queryXML` 引数で渡されます。クエリ・パラメータの値は、`parameterizedNodes` 引数で渡されます。TQL には、各 `CINode` および各 `relationNode` に対して一意のラベルが定義されている必要があります。

`executeTopologyQueryWithParameters` メソッドは、UCMDB で定義されているクエリにアクセスするためではなく、アドホック・クエリを渡すために使用します。このメソッドは、UCMDB ユーザ・インタフェースにアクセスしてクエリを定義する権限がないときや、クエリをデータベースに保存したくないときに使用できます。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。

パラメータ	コメント
queryXML	TQL を XML 形式で表現したもの。
parameterizedNodes	クエリ結果の対象となるために各ノードが満たす必要がある条件。

出力

パラメータ	コメント
topologyMap	詳細については、90 ページ「TopologyMap」を参照してください。
chunkInfo	詳細については、90 ページ「ChunkInfo」と 24 ページ「サイズの大きい応答の処理」を参照してください。

getChangedCIs

getChangedCIs メソッドは、指定した CI に関連するすべての CI の変更データを返します。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
ids	関連 CI の変更の有無がチェックされるルート CI の ID のリスト。 このコレクションでは、実際の UCMDb ID だけが有効です。
fromDate	CI が変更されたかどうかをチェックする期間の開始点。
toDate	CI が変更されたかどうかをチェックする期間の終了点。

出力

パラメータ	コメント
changeDataInfo	ChangedDataInfo 要素のゼロ個以上のコレクション。

 **getCI Neighbours**

getCI Neighbours メソッドは、指定した CI の隣接項目を返します。

たとえば、クエリが CIA の隣接項目を対象としており、CIA に、CIC を使う CIB が含まれている場合、CIB は返されますが、CIC は返されません。つまり、指定したタイプの隣接項目のみが返されます。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
ID	隣接項目の取得に使う CI の ID。この値は、実際の UCMDb ID である必要があります。
neighbourType	取得する隣接項目の CIT 名。指定したタイプの隣接項目、およびそのタイプから派生したタイプの隣接項目が返されます。 詳細については、87 ページ「タイプ名」を参照してください。
CIProperties	各構成アイテム上の返されるデータ。ユーザ・インタフェースではクエリ・レイアウトと呼びます。 詳細については、27 ページ「TypedProperties : TypedProperties は、CIT ごとに異なるセットのプロパティを渡すために使用します。TypedProperties は、タイプ名と、すべてのタイプのプロパティ・セットから構成されるペアのコレクションです。各プロパティ・セットは、対応するタイプのみ適用されます。」を参照してください。
relationProperties	各関係上の返されるデータ。ユーザ・インタフェースではクエリ・レイアウトと呼びます。詳細については、27 ページ「TypedProperties : TypedProperties は、CIT ごとに異なるセットのプロパティを渡すために使用します。TypedProperties は、タイプ名と、すべてのタイプのプロパティ・セットから構成されるペアのコレクションです。各プロパティ・セットは、対応するタイプのみ適用されます。」を参照してください。

出力

パラメータ	コメント
topology	詳細については、89 ページ「Topology」を参照してください。
comments	内部使用専用。

 **getCIsByID**

getCIsByID メソッドは、UCMDB ID を使って構成アイテムを取得します。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
CIsTypedProperties	タイプ別プロパティのコレクション。詳細については、27 ページ「TypedProperties : TypedProperties は、CIT ごとに異なるセットのプロパティを渡すために使用します。TypedProperties は、タイプ名と、すべてのタイプのプロパティ・セットから構成されるペアのコレクションです。各プロパティ・セットは、対応するタイプのみにも適用されます。」を参照してください。
IDs	このコレクションでは、実際の UCMDB ID だけが有効です。

出力

パラメータ	コメント
CIs	CI 要素のコレクション。
chunkInfo	詳細については、90 ページ「ChunkInfo」と 24 ページ「サイズの大きい応答の処理」を参照してください。

getClsByType

getClsByType メソッドは、指定したタイプ、および指定したタイプから継承するすべてのタイプの構成アイテムのコレクションを返します。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
type	クラス名。詳細については、87 ページ「タイプ名」を参照してください。
properties	各構成アイテム上の返されるデータ。 詳細については、26 ページ「CustomProperties : CustomProperties は、基本的な PropertiesList とルール・ベースのプロパティ・リストの組み合わせを含むことができます。プロパティ・フィルタは、すべてのリストによって返されるすべてのプロパティを結合したものです。」を参照してください。

出力

パラメータ	コメント
Cls	CI 要素のコレクション。
chunkInfo	詳細については、90 ページ「ChunkInfo」と 24 ページ「サイズの大きい応答の処理」を参照してください。

getFilteredCIsByType

getFilteredCIsByType メソッドは、メソッドで使用する条件を満たす指定したタイプの CI を取得します。条件には、次の要素が含まれます。

- ▶ プロパティの名前を含む名前フィールド。
- ▶ 比較演算子を含む演算子フィールド。
- ▶ 値または値のリストを含むオプションの値フィールド。

これらによって、論理式を作成します。

```
<アイテム> .property.value [operator] <条件> .value
```

たとえば、条件名が `root_actualdeletionperiod` で、条件値が `40` で、演算子が **等価** の場合、論理式は次のようになります。

```
<アイテム> .root_actualdeletionperiod.value == 40
```

ほかに条件を指定しなければ、クエリによって、`root_actualdeletionperiod` が `40` のアイテムがすべて返されます。

`conditionsLogicalOperator` 引数が **AND** の場合、クエリによって、**条件**コレクションで指定した条件をすべて満たすアイテムが返されます。

`conditionsLogicalOperator` が **OR** の場合は、クエリによって、**条件**コレクションで指定した条件の少なくとも 1 つを満たすアイテムが返されます。

次の表は、比較演算子を示します。

演算子	条件の種類とコメント
ChangedDuring	<p>Date</p> <p>これにより範囲をチェックします。条件値は時間単位で指定します。date プロパティの値が、メソッドが呼び出された時点で条件値を加えた、または引いた範囲内にある場合、条件は true となります。</p> <p>たとえば、条件値が 24 の場合、date プロパティの値が昨日のこの時刻と明日のこの時刻の間であれば、条件は true となります。</p> <p>注： ChangedDuring という名前は、下位互換性を維持するために予約されています。以前のバージョンでは、この演算子は Create Time, Modify Time にのみ使用されていました。</p>
Equal	文字列および数値
EqualIgnoreCase	文字列
Greater	数値
GreaterEqual	数値
In	<p>文字列、数値、およびリスト</p> <p>条件の値はリストです。プロパティの値がリスト内の値の 1 つであれば、条件は true になります。</p>
InList	<p>リスト</p> <p>条件の値とプロパティの値はリストです。</p> <p>条件のリスト内のすべての値がアイテムのプロパティ・リストにもあれば、条件は true になります。条件の真偽に影響を与えることなく、条件で指定したより多くのプロパティ値を利用できます。</p>
IsNull	<p>文字列、数値、およびリスト</p> <p>アイテムのプロパティに値がありません。演算子 IsNull を使用すると、条件の値は無視され、場合によっては nil 扱いとなります。</p>
Less	数値
LessEqual	数値

演算子	条件の種類とコメント
Like	文字列 条件の値はプロパティ値の部分文字列です。条件の値は、パーセント記号 (%) で囲む必要があります。たとえば、%Bi% は Bismark と Bay of Biscay に一致しますが、biscuit には一致しません。
LikeIgnoreCase	文字列 Like 演算子を使用するのと同様に、LikeIgnoreCase 演算子を使用します。ただし、大文字と小文字の区別は一致条件に入りません。このため、%Bi% は biscuit に一致します。
NotEqual	文字列および数値
UnchangedDuring	Date これにより範囲をチェックします。条件値は時間単位で指定します。date プロパティの値が、メソッドが呼び出された時点で条件値を加えた、または引いた範囲内にある場合、条件は false となります。この値が範囲外にある場合、条件は true となります。 たとえば、条件値が 24 の場合、date プロパティの値が昨日のこの時刻の前、または明日のこの時刻の後であれば、条件は true となります。 注： UnchangedDuring という名前は、下位互換性を維持するために予約されています。以前のバージョンでは、この演算子は Create Time, Modify Time にのみ使用されていました。

条件設定の例：

```
FloatCondition fc = new FloatCondition();
FloatProp fp = new FloatProp();
fp.setName("attr_name");
fp.setValue(11);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.floatOperatorEnum.Equal);
```

継承したプロパティを問い合わせる例：

ターゲット CI は、**name** と **size** という 2 つの属性を持つ **sample** です。
samplell では、**level** と **grade** という 2 つの属性によって CI は拡張されます。
 この例では、名前を使って指定することによって、**sample** から継承された **samplell** のプロパティに対するクエリをセットアップします。

```
GetFilteredCIsByType request = new GetFilteredCIsByType()
request.setCmdbContext(cmdbContext)
request.setType("samplell")
CustomProperties customProperties = new CustomProperties();
PropertiesList propertiesList = new PropertiesList();
propertiesList.addPropertyName("name");
propertiesList.addPropertyName("size");
customProperties.setPropertiesList(propertiesList);
request.setProperties(customProperties)
```

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
type	クラス名。詳細については、87 ページ「タイプ名」を参照してください。タイプは、CI タイプ・マネージャを使用して定義したタイプのいずれでもかまいません。詳細については、『モデル管理』の「CI タイプ・マネージャ」を参照してください。
properties	各 CI 上の返されるデータ（ユーザ・インタフェースではクエリ・レイアウトと呼びます）。 詳細については、26 ページ「CustomProperties : CustomProperties は、基本的な PropertiesList とルール・ベースのプロパティ・リストの組み合わせを含むことができます。プロパティ・フィルタは、すべてのリストによって返されるすべてのプロパティを結合したものです。」を参照してください。

パラメータ	コメント
conditions	名前と値のペアのコレクションと、一方を他方に関連付ける演算子。たとえば、 <code>host_hostname like QA</code> などです。
conditionsLogicalOperator	<ul style="list-style-type: none"> ▶ AND : すべての条件を満たす必要があります。 ▶ OR : 少なくとも1つの条件を満たす必要があります。

出力

パラメータ	コメント
CIs	CI 要素のコレクション。
chunkInfo	詳細については、90 ページ「 ChunkInfo 」と 24 ページ「サイズの大きい応答の処理」を参照してください。

getQueryNameOfView

`getQueryNameOfView` メソッドは、指定したビューの基となる TQL の名前を取得します。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「 CmdbContext 」を参照してください。
viewName	ビューの名前。つまり UCMDB 内のクラス・モデルのサブセット。

出力

パラメータ	コメント
queryName	ビューの基となる UCMDB 内の TQL の名前。

getTopologyQueryExistingResultByName

`getTopologyQueryExistingResultByName` メソッドは、指定した TQL の最新の
実行結果を取得します。呼び出しを実行しても TQL は実行されません。前回の
実行結果が存在しない場合は、何も返しません。

入力

パラメータ	コメント
<code>cmdbContext</code>	詳細については、85 ページ「 <code>CmdbContext</code> 」を参照してください。
<code>queryName</code>	TQL の名前。
<code>queryTypedProperties</code>	特定の構成アイテム・タイプのアイテムを取得するための、 プロパティのセットのコレクション。

出力

パラメータ	コメント
<code>queryName</code>	ビューの基となる UCMDB 内の TQL の名前。

getTopologyQueryResultCountByName

`getTopologyQueryResultCountByName` メソッドは、指定したクエリに一致する
各ノードのインスタンスの数を取得します。

入力

パラメータ	コメント
<code>cmdbContext</code>	詳細については、85 ページ「 <code>CmdbContext</code> 」を参照して ください。
<code>queryName</code>	TQL の名前。
<code>countInvisible</code>	<code>true</code> の場合、クエリで非表示として定義された CI が出力 に含まれます。

出力

パラメータ	コメント
queryName	ビューの基となる UCMDDB 内の TQL の名前。

 pullTopologyMapChunks

pullTopologyMapChunks メソッドは、メソッドへの応答を含むチャンクの 1 つを取得します。

各チャンクは、応答の一部である topologyMap 要素を含みます。1 つ目のチャンクは 1 という番号が付けられているため、取得ループ・カウンタは、1 から <応答オブジェクト> .getChunkInfo().getNumberOfChunks() を反復します。

詳細については、90 ページ「ChunkInfo」と 23 ページ「UCMDDB への問い合わせ」を参照してください。

クライアント・アプリケーションは、部分的なマップを処理できる必要があります。CI コレクションの処理に関する次の例、およびチャンクをマップに結合する例 (58 ページ「クエリの例」) を参照してください。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
ChunkRequest	取得するチャンク、およびクエリ・メソッドによって返される ChunkInfo の数。

出力

パラメータ	コメント
topologyMap	詳細については、90 ページ「TopologyMap」を参照してください。
comments	内部使用専用。

チャンクの処理例

```

GetCIsByType request =
    new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
    ucmdbService.getCIsByType(request);
ChunkRequest chunkRequest = new ChunkRequest();
chunkRequest.setChunkInfo(response.getChunkInfo());
for(int j=1 ; j < response.getChunkInfo().getNumberOfChunks() ; j++) {
    chunkRequest.setChunkNumber(j);
    PullTopologyMapChunks req = new PullTopologyMapChunks(cmdbContext,
chunkRequest);
    PullTopologyMapChunksResponse res =
        ucmdbService.pullTopologyMapChunks(req);
    for(int m=0 ;
        m < res.getTopologyMap().getCINodes().sizeCINodeList() ;
        m++) {
        CIs cis =
            res.getTopologyMap().getCINodes().getCINode(m).getCIs();
        for(int i=0 ; i < cis.sizeCICollection() ; i++) {
            // CI を処理するコード
        }
    }
}
}

```

releaseChunks

releaseChunks メソッドは、クエリからのデータを含むチャンクのメモリを解放します。

ヒント：10 分後にサーバによってデータは破棄されます。読み取りが終了したらすぐにデータを破棄するためにこのメソッドを呼び出すと、サーバのリソースを節約できます。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
chunksKey	チャンクに分けられたサーバ上のデータの識別子。キーは、ChunkInfo の要素です。

UCMDB 更新メソッド

本項では、次のメソッドに関する情報を提供します。

- ▶ 47 ページ「addCIsAndRelations」
- ▶ 49 ページ「deleteCIsAndRelations」
- ▶ 49 ページ「updateCIsAndRelations」

addCIsAndRelations

addCIsAndRelations メソッドは、CI および関係を追加または更新します。

CI または関係が UCMDB に存在しない場合は、これらは追加され、それぞれのプロパティが CIsAndRelationsUpdates 引数の内容に従って設定されます。

CI または関係が UCMDB に存在する場合は、updateExisting が **true** であれば、これらは新しいデータを使って更新されます。

updateExisting が **false** の場合は、CIsAndRelationsUpdates は、既存の構成アイテムまたは関係を参照できません。updateExisting が **false** の場合に既存のアイテムを参照しようとする、例外が発生します。

updateExisting が **true** であれば、ignoreValidation の値に関係なく、CI を検証することなく追加操作または更新操作が実行されます。

updateExisting が **false** で、ignoreValidation が **true** の場合、CI を検証することなく追加操作が実行されます。

updateExisting が **false** で ignoreValidation が **false** の場合、追加操作の前に CI が検証されます。

関係は検証されません。

CreatedIDsMap は、クライアントの一時 ID を、対応する実際の UCMDB ID に結び付ける ClientIDToCmdbID タイプのマップまたはディクショナリです。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
updateExisting	true に設定すると、UCMDB に存在するアイテムが更新されます。 false に設定すると、アイテムが存在する場合に例外がスローされます。
CIsAndRelationsUpdates	更新または作成するアイテム。詳細については、27 ページ「CIsAndRelationsUpdates」を参照してください。
ignoreValidation	true の場合、UCMDB を更新する前にチェックは行われません。

出力

パラメータ	コメント
CreatedIDsMap	クライアント ID と UCMDB ID のマップ。詳細については、47 ページ「addCIsAndRelations」を参照してください。
comments	内部使用専用。

deleteCIsAndRelations

`deleteCIsAndRelations` メソッドは、指定した構成アイテムと関係を UCMDDB から削除します。

CI を削除して、CI が 1 つ以上の関係アイテムの一端にしかない場合、これらの関係アイテムも削除されます。

入力

パラメータ	コメント
<code>cmdbContext</code>	詳細については、85 ページ「 <code>CmdbContext</code> 」を参照してください。
<code>CIsAndRelationsUpdates</code>	削除するアイテム。詳細については、27 ページ「 <code>CIsAndRelationsUpdates</code> 」を参照してください。

updateCIsAndRelations

`updateCIsAndRelations` メソッドは、指定した CI と関係を更新します。

更新には、`CIsAndRelationsUpdates` 引数のプロパティ値が使用されます。CI または関係のいずれかが UCMDDB に存在しない場合、例外がスローされます。

`CreatedIDsMap` は、クライアントの一時 ID を、対応する実際の UCMDDB ID に結び付ける `ClientIDToCmdbID` タイプのマップまたはディクショナリです。

入力

パラメータ	コメント
<code>cmdbContext</code>	詳細については、85 ページ「 <code>CmdbContext</code> 」を参照してください。
<code>CIsAndRelationsUpdates</code>	更新するアイテム。詳細については、27 ページ「 <code>CIsAndRelationsUpdates</code> 」を参照してください。
<code>ignoreValidation</code>	<code>true</code> の場合、UCMDDB を更新する前にチェックは行われません。

出力

パラメータ	コメント
CreatedIDsMap	クライアント ID と UCMDDB ID のマップ。詳細については、47 ページ「addCIsAndRelations」を参照してください。

 UCMDB の影響分析メソッド

本項では、次のメソッドに関する情報を提供します。

- ▶ 50 ページ「calculateImpact」
- ▶ 51 ページ「getImpactPath」
- ▶ 52 ページ「getImpactRulesByNamePrefix」

 calculateImpact

calculateImpact メソッドは、UCMDDB に定義したルールに従って、どの CI が特定の CI の影響を受けるかを計算します。

これにより、ルールのイベント・トリガの効果がわかります。calculateImpact の identifier 出力は、getImpactPath の入力として使用します。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
impactCategory	シミュレートするルールをトリガするイベントのタイプ。
IDs	ID 要素のコレクション。
impactRulesNames	ImpactRuleName 要素のコレクション。
severity	トリガ・イベントの重要度。

出力

パラメータ	コメント
impactTopology	詳細については、89 ページ「Topology」を参照してください。
identifier	サーバ応答に対するキー。

 **getImpactPath**

getImpactPath メソッドは、影響を受ける CI と影響を与える CI の間のパスのトポロジ・グラフを取得します。

calculateImpact の **identifier** 出力は、**getImpactPath** の **identifier** 入力引数として使用します。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
identifier	calculateImpact によって返されたサーバ応答に対するキー。
relation	impactTopology 要素の calculateImpact によって返された ShallowRelation の 1 つに基づく関係。

出力

パラメータ	コメント
impactPathTopology	CIs コレクションと ImpactRelations コレクション。
コメント	内部使用専用。

ImpactRelations 要素は、ID, type, end1ID, end2ID, rule, および action から構成されます。

getImpactRulesByNamePrefix

getImpactRulesByNamePrefix メソッドは、プレフィックス・フィルタを使用してルールを取得します。

このメソッドは、適用先の内容を示すプレフィックスを名前を含む影響ルールに適用されます。たとえば、SAP_myrule, ORA_myrule などです。このメソッドは、すべての影響ルール名をフィルタして、ruleNamePrefixFilter 引数で指定したプレフィックスで始まるものを探します。

入力

パラメータ	コメント
cmdbContext	詳細については、85 ページ「CmdbContext」を参照してください。
ruleNamePrefixFilter	一致するルール名の最初の文字を含む文字列。

出力

パラメータ	コメント
impactRules	impactRules は、ゼロ個以上の impactRule から構成されます。変更の効果を指定する impactRule は、ruleName, description, queryName, および isActive から構成されます。

使用例

次の使用例では、2つのシステムを使用することを想定しています。

- ▶ HP Universal CMDB サーバ
- ▶ 構成アイテムのリポジトリを含むサードパーティ製システム

本項の内容

- ▶ 53 ページ「UCMDB へのデータ入力」
- ▶ 53 ページ「UCMDB への問い合わせ」
- ▶ 54 ページ「クラス・モデルへの問い合わせ」
- ▶ 54 ページ「変更の影響の分析」

UCMDB へのデータ入力

使用例：

- ▶ サードパーティ製のアセット管理により、アセット管理のみで利用できる情報を使って UCMDB を更新します。
- ▶ 多くのサードパーティ製のシステムは、UCMDB にデータを入力して、変更内容を追跡し影響分析を実行できる中心的な CMDB を作成します。
- ▶ サードパーティ製のシステムは、サードパーティのビジネス・ロジックに従って構成アイテムと関係を作成し、CMDB クエリ機能を活用します。

UCMDB への問い合わせ

使用例：

- ▶ サードパーティ製のシステムは、SAP TQL の結果を取得することによって、SAP システムを表す構成アイテムと関係を取得します。
- ▶ サードパーティ製のシステムは、過去 5 時間以内に追加または変更された Oracle サーバのリストを取得します。
- ▶ サードパーティ製のシステムは、ホスト名に部分文字列 **lab** が含まれるサーバのリストを取得します。
- ▶ サードパーティ製のシステムは、隣接項目を取得することによって、特定の CI に関係する要素を検出します。

クラス・モデルへの問い合わせ

使用例：

- ▶ サードパーティ製のシステムでは、ユーザは UCMDDB から取得するデータのセットを指定できます。ユーザ・インタフェースはクラス・モデル上に構築し、ユーザに利用可能なプロパティを表示して、必要なデータを求めることができます。ユーザは、取得する情報を選択できます。
- ▶ サードパーティ製のシステムは、ユーザが UCMDDB ユーザ・インタフェースにアクセスできないときに、クラス・モデルを探索します。

変更の影響の分析


使用例：

サードパーティ製のシステムは、指定したホストに対する変更の影響を受けるビジネス・サービスのリストを出力します。

例

本項の内容

- ▶ 55 ページ「基本クラスの例」
- ▶ 58 ページ「クエリの例」
- ▶ 75 ページ「更新の例」
- ▶ 79 ページ「クラス・モデルの例」
- ▶ 81 ページ「影響分析の例」

 基本クラスの例

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.services.UcmdbService;
import com.hp.ucmdb.generated.services.UcmdbServiceStub;
import com.hp.ucmdb.generated.types.CmdbContext;
import org.apache.axis2.AxisFault;
import org.apache.axis2.transport.http.HTTPConstants;

import org.apache.axis2.transport.http.HttpTransportProperties;

import java.net.MalformedURLException;
import java.net.URL;

/**
 * User:hbarkai
 * Date:Jul 12, 2007
 */
abstract class Demo {

    UcmdbService stub;
    CmdbContext context;

    public void initDemo() {
        try {
            setStub(createUcmdbService("admin", "admin"));
            setContext();
        } catch (Exception e) {
            // 例外の処理
        }
    }

    public UcmdbService getStub() {
        return stub;
    }
}
```

```
public void setStub(UcmdbService stub) {
    this.stub = stub;
}
```

```
public CmdbContext getContext() {
    return context;
}
```

```
public void setContext() {
    CmdbContext context = new CmdbContext();
    context.setCallerApplication("demo");
    this.context = context;
}
```

```
// サービスへの接続 - axis2/jibx クライアント用

private static final String PROTOCOL = "http";
private static final String HOST_NAME = "host_name";
private static final int PORT = 8080;
private static final String FILE = "/axis2/services/UcmdbService";

protected UcmdbService createUcmdbService
(String username, String password) throws Exception{
    URL url;
    UcmdbServiceStub serviceStub;
```

```
try {
    url = new URL
        (Demo.PROTOCOL, Demo.HOST_NAME,
        Demo.PORT, Demo.FILE);
    serviceStub = new UcmdbServiceStub(url.toString());
    HttpTransportProperties.Authenticator auth =
        new HttpTransportProperties.Authenticator();
    auth.setUsername(username);
    auth.setPassword(password);
    serviceStub._getServiceClient().getOptions().setProperty
        (HTTPConstants.AUTHENTICATE, auth);
```



```
} catch (AxisFault axisFault) {  
    throw new Exception  
        ("Failed to create SOAP adapter for "  
        + Demo.HOST_NAME , axisFault);
```

```
    } catch (MalformedURLException e) {  
  
        throw new Exception  
            ("Failed to create SOAP adapter for "  
            + Demo.HOST_NAME, e);  
    }  
    return serviceStub;  
}  
}
```

クエリの例

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.query.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.services.UcmdbService;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.props.*;

import java.rmi.RemoteException;
```

```
public class QueryDemo extends Demo{

    UcmdbService stub;
    CmdbContext context;
```

```
public void getClsByTypeDemo() {
    GetClsByType request = new GetClsByType();
    // cmdbcontext の設定
    CmdbContext cmdbContext = getContext();
    request.setCmdbContext(cmdbContext);
    // CI タイプの設定
    request.setType("anyType");
    // 取得する CI プロパティの設定
    CustomProperties customProperties = new CustomProperties();
    PredefinedProperties predefinedProperties =
        new PredefinedProperties();
    SimplePredefinedProperty simplePredefinedProperty =
        new SimplePredefinedProperty();
    simplePredefinedProperty.setName
        (SimplePredefinedProperty.nameEnum.DERIVED);
    SimplePredefinedPropertyCollection
        simplePredefinedPropertyCollection =
        new SimplePredefinedPropertyCollection();
```

```

simplePredefinedPropertyCollection.addSimplePredefinedProperty
    (simplePredefinedProperty);
predefinedProperties.setSimplePredefinedProperties
    (simplePredefinedPropertyCollection);
customProperties.setPredefinedProperties(predefinedProperties);
request.setProperties(customProperties);
try {
    GetCIsByTypeResponse response =
        getStub().getCIsByType(request);
    TopologyMap map =
        getTopologyMapResultFromCIs
            (response.getCIs(), response.getChunkInfo());
} catch (RemoteException e) {
    // 例外の処理
} catch (UcmdbFaultException e) {
    // 例外の処理
}
}

```

```

public void getCIsByIdDemo() {
    GetCIsById request = new GetCIsById();
    CmdbContext cmdbContext = getContext();
    // cmdbcontext の設定
    request.setCmdbContext(cmdbContext);
    // ID の設定
    ID id1 = new ID();
    id1.setBase("cmdbobjectidCIT1");
    ID id2 = new ID();
    id2.setBase("cmdbobjectidCIT2");
    IDs ids = new IDs();
    ids.addID(id1);
    ids.addID(id2);
    request.setIDs(ids);
    // 取得する CI プロパティの設定
    TypedPropertiesCollection properties =
        new TypedPropertiesCollection();

```

```

TypedProperties typedProperties1 =
    new TypedProperties();
typedProperties1.setType("CIT1");

```

```
CustomTypedProperties customProperties1 =
    new CustomTypedProperties();
PredefinedTypedProperties predefinedProperties1 =
    new PredefinedTypedProperties();
SimpleTypedPredefinedProperty simplePredefinedProperty1 =
    new SimpleTypedPredefinedProperty();
simplePredefinedProperty1.setName
    (SimpleTypedPredefinedProperty.nameEnum.CONCRETE);
SimpleTypedPredefinedPropertyCollection
    simplePredefinedPropertyCollection1 =
        new SimpleTypedPredefinedPropertyCollection();
simplePredefinedPropertyCollection1
    .addSimpleTypedPredefinedProperty
        (simplePredefinedProperty1);
```

```
predefinedProperties1.
    setSimpleTypedPredefinedProperties
        (simplePredefinedPropertyCollection1);
customProperties1.
    setPredefinedTypedProperties
        (predefinedProperties1);
typedProperties1.setProperties(customProperties1);
properties.addTypedProperties(typedProperties1);
```

```
TypedProperties typedProperties2 =
    new TypedProperties();
typedProperties2.setType("CIT2");
CustomTypedProperties customProperties2 =
    new CustomTypedProperties();
PredefinedTypedProperties predefinedProperties2 =
    new PredefinedTypedProperties();
SimpleTypedPredefinedProperty simplePredefinedProperty2 =
    new SimpleTypedPredefinedProperty();
simplePredefinedProperty2.setName
    (SimpleTypedPredefinedProperty.nameEnum.NAMING);
SimpleTypedPredefinedPropertyCollection
    simplePredefinedPropertyCollection2 =
        new SimpleTypedPredefinedPropertyCollection();
```

```
simplePredefinedPropertyCollection2.  
    addSimpleTypedPredefinedProperty  
        (simplePredefinedProperty2);
```

```
predefinedProperties2.setSimpleTypedPredefinedProperties  
    (simplePredefinedPropertyCollection2);  
customProperties2.setPredefinedTypedProperties  
    (predefinedProperties2);  
typedProperties2.setProperties(customProperties2);  
properties.addTypedProperties(typedProperties2);
```

```
request.setClsTypedProperties(properties);  
try {  
    GetClsByIdResponse response =  
        getStub().getClsById(request);  
    Cls cis = response.getCls();  
} catch (RemoteException e) {  
    // 例外の処理  
} catch (UcmdbFaultException e) {  
    // 例外の処理  
}  
}
```

```
public void getFilteredClsByTypeDemo() {  
    GetFilteredClsByType request = new GetFilteredClsByType();  
    CmdbContext cmdbContext = getContext();  
    // cmdbcontext の設定  
    request.setCmdbContext(cmdbContext);  
    // CI タイプの設定  
    request.setType("anyType");  
    // フィルタ条件の設定  
    Conditions conditions = new Conditions();  
    IntConditions intConditions = new IntConditions();  
    IntCondition intCondition = new IntCondition();  
    IntProp intProp = new IntProp();  
    intProp.setName("int_attr1");
```

```
intProp.setValue(100);
intCondition.setCondition(intProp);
intCondition.setIntOperator
    (IntCondition.intOperatorEnum.Greater);
intConditions.addIntCondition(intCondition);
```

```
conditions.setIntConditions(intConditions);
request.setConditions(conditions);
// 条件の論理演算子の設定
request.setConditionsLogicalOperator
    (GetFilteredCIsByType.conditionsLogicalOperatorEnum.AND);
// 取得する CI プロパティの設定
CustomProperties customProperties =
    new CustomProperties();
PredefinedProperties predefinedProperties =
    new PredefinedProperties();
SimplePredefinedProperty simplePredefinedProperty =
    new SimplePredefinedProperty();
simplePredefinedProperty.setName
    (SimplePredefinedProperty.nameEnum.NAMING);
```

```
SimplePredefinedPropertyCollection
    simplePredefinedPropertyCollection =
        new SimplePredefinedPropertyCollection();
simplePredefinedPropertyCollection.
    addSimplePredefinedProperty
        (simplePredefinedProperty);
predefinedProperties.setSimplePredefinedProperties
    (simplePredefinedPropertyCollection);
customProperties.setPredefinedProperties
    (predefinedProperties);
```

```
request.setProperties(customProperties);
try {
    GetFilteredCIsByTypeResponse response =
        getStub().getFilteredCIsByType(request);
    TopologyMap map =
        getTopologyMapResultFromCIs
            (response.getCIs(), response.getChunkInfo());
```

```
    } catch (RemoteException e) {  
        // 例外の処理  
    } catch (UcmdbFaultException e) {  
        // 例外の処理  
    }  
}
```

```
public void executeTopologyQueryByNameDemo() {  
    ExecuteTopologyQueryByName request = new  
ExecuteTopologyQueryByName();  
    CmdbContext cmdbContext = getContext();  
    // cmdbcontext の設定  
    request.setCmdbContext(cmdbContext);  
    // クエリ名の設定  
    request.setQueryName("queryName");  
}
```

```
try {  
    ExecuteTopologyQueryByNameResponse response =  
        getStub().executeTopologyQueryByName(request);  
    TopologyMap map =  
        getTopologyMapResult  
            (response.getTopologyMap(), response.getChunkInfo());  
} catch (RemoteException e) {  
    // 例外の処理  
} catch (UcmdbFaultException e) {  
    // 例外の処理  
}  
}
```

```
// 次のクエリが UCMDB で定義されていると想定
// クエリ名 : exampleQuery
// クエリの概略図 :
//
//                               ホスト
//                               / \
//                               ip ディスク
// クエリのパラメータ :
//   ホスト -
//     host_os (like)
//   ディスク -
//     disk_failures (equal)
```

```
public void executeTopologyQueryByNameWithParametersDemo() {
    ExecuteTopologyQueryByNameWithParameters request =
        new ExecuteTopologyQueryByNameWithParameters();
    CmdbContext cmdbContext = getContext();
    // cmdbcontext の設定
    request.setCmdbContext(cmdbContext);
    // クエリ名の設定
    request.setQueryName("queryName");
    // パラメータの設定
    ParameterizedNode hostParameterizedNode =
        new ParameterizedNode();
    hostParameterizedNode.setNodeLabel("Host");
    CIProperties parameters = new CIProperties();
    StrProps strProps = new StrProps();
    StrProp strProp = new StrProp();
    strProp.setName("host_os");
    strProp.setValue("%2000%");
    strProps.addStrProp(strProp);
    parameters.setStrProps(strProps);
    hostParameterizedNode.setParameters(parameters);
    request.addParameterizedNodes(hostParameterizedNode);
    ParameterizedNode diskParameterizedNode =
        new ParameterizedNode();
```

```
    diskParameterizedNode.setNodeLabel("Disk");
    CIProperties parameters1 = new CIProperties();
    IntProps intProps = new IntProps();
```



```

IntProp intProp = new IntProp();
intProp.setName("disk_failures");
intProp.setValue(30);
intProps.addIntProp(intProp);
parameters1.setIntProps(intProps);
diskParametrizedNode.setParameters(parameters1);

```

```

request.addParameterizedNodes(diskParametrizedNode);
try {
    ExecuteTopologyQueryByNameWithParametersResponse
        response =
            getStub().executeTopologyQueryByNameWithParameters
                (request);
    TopologyMap map =
        getTopologyMapResult
            (response.getTopologyMap(), response.getChunkInfo());
} catch (RemoteException e) {
    // 例外の処理
} catch (UcldbFaultException e) {
    // 例外の処理
}
}

```

```

/// 次のクエリが UCMDDB で定義されていると想定
// クエリ名 : exampleQuery
// クエリの概略図 :
//
//                               ホスト
//                               / \
//                               ip ディスク
// クエリのパラメータ :
//   ホスト -
//     host_os (like)
//   ディスク -
//     disk_failures (equal)

```

```
public void executeTopologyQueryWithParametersDemo() {
    ExecuteTopologyQueryWithParameters request =
        new ExecuteTopologyQueryWithParameters();
    CmdbContext cmdbContext = getContext();
    // cmdbcontext の設定
    request.setCmdbContext(cmdbContext);
    // クエリ定義の設定
    String queryXml = "<xml that represents the query above>";
    request.setQueryXml(queryXml);
    // パラメータの設定
    ParameterizedNode hostParameterizedNode =
        new ParameterizedNode();
```

```
    hostParameterizedNode.setNodeLabel("Host");
    CIProperties parameters = new CIProperties();
    StrProps strProps = new StrProps();
    StrProp strProp = new StrProp();
    strProp.setName("host_os");
    strProp.setValue("%2000%");
    strProps.addStrProp(strProp);
    parameters.setStrProps(strProps);
    hostParameterizedNode.setParameters(parameters);
    request.addParameterizedNodes(hostParameterizedNode);
    ParameterizedNode diskParameterizedNode =
        new ParameterizedNode();
    diskParameterizedNode.setNodeLabel("Disk");
    CIProperties parameters1 = new CIProperties();
    IntProps intProps = new IntProps();
    IntProp intProp = new IntProp();
    intProp.setName("disk_failures");
    intProp.setValue(30);
    intProps.addIntProp(intProp);
    parameters1.setIntProps(intProps);
    diskParameterizedNode.setParameters(parameters1);
    request.addParameterizedNodes(diskParameterizedNode);
```

```

try {
    ExecuteTopologyQueryWithParametersResponse
    response = getStub().executeTopologyQueryWithParameters
        (request);
    TopologyMap map =
        getTopologyMapResult
            (response.getTopologyMap(), response.getChunkInfo());

```

```

    } catch (RemoteException e) {
        // 例外の処理
    } catch (UcmdbFaultException e) {
        // 例外の処理
    }
}

```

```

public void getCI NeighboursDemo() {
    GetCI Neighbours request = new GetCI Neighbours();
    // cmdbcontext の設定
    CmdbContext cmdbContext = getContext();
    request.setCmdbContext(cmdbContext);
    // CI の ID 設定
    ID id = new ID();
    id.setBase("cmdbobjectidCIT1");
    request.setID(id);
    // 隣接項目のタイプの設定
    request.setNeighbourType("neighbourType");
    // 取得する隣接 CI のプロパティの設定
    TypedPropertiesCollection properties =
        new TypedPropertiesCollection();
    TypedProperties typedProperties1 = new TypedProperties();
    typedProperties1.setType("neighbourType");
    CustomTypedProperties customProperties1 =
        new CustomTypedProperties();
    PredefinedTypedProperties predefinedProperties1 =
        new PredefinedTypedProperties();

```

```
QualifierProperties qualifierProperties =  
    new QualifierProperties();  
qualifierProperties.addQualifierName("ID_ATTRIBUTE");  
predefinedProperties1.setQualifierProperties(qualifierProperties);  
customProperties1.setPredefinedTypedProperties  
    (predefinedProperties1);  
typedProperties1.setProperties(customProperties1);  
properties.addTypedProperties(typedProperties1);  
request.setCIPProperties(properties);
```

```
TypedPropertiesCollection relationsProperties =  
    new TypedPropertiesCollection();  
TypedProperties typedProperties2 = new TypedProperties();  
typedProperties2.setType("relationType");  
CustomTypedProperties customProperties2 =  
    new CustomTypedProperties();
```

```
PredefinedTypedProperties predefinedProperties2 =  
    new PredefinedTypedProperties();  
SimpleTypedPredefinedProperty simplePredefinedProperty2 =  
    new SimpleTypedPredefinedProperty();  
simplePredefinedProperty2.setName
```

```
(SimpleTypedPredefinedProperty.nameEnum.CONCRETE);  
SimpleTypedPredefinedPropertyCollection  
    simplePredefinedPropertyCollection2 =  
    new SimpleTypedPredefinedPropertyCollection();  
simplePredefinedPropertyCollection2.  
    addSimpleTypedPredefinedProperty  
        (simplePredefinedProperty2);  
predefinedProperties2.  
    setSimpleTypedPredefinedProperties  
        (simplePredefinedPropertyCollection2);  
customProperties2.setPredefinedTypedProperties  
    (predefinedProperties2);  
typedProperties2.setProperties(customProperties2);  
relationsProperties.addTypedProperties(typedProperties2);  
request.setRelationProperties(relationsProperties);
```

```

    try {
        GetCI NeighboursResponse response =
            getStub().getCI Neighbours(request);
        Topology topology = response.getTopology();
    } catch (RemoteException e) {
        // 例外の処理
    } catch (UcldbFaultException e) {
        // 例外の処理
    }
}

```

// チャンク化された結果 / チャンク化されていない結果のトポロジ・マップの取得

```

private TopologyMap getTopologyMapResult(TopologyMap topologyMap,
ChunkInfo chunkInfo) {
    if(chunkInfo.getNumberOfChunks() == 0) {
        return topologyMap;
    } else {

```

```

        topologyMap = new TopologyMap();
        for(int i=1 ; i <= chunkInfo.getNumberOfChunks() ; i++) {
            ChunkRequest chunkRequest = new ChunkRequest();
            chunkRequest.setChunkInfo(chunkInfo);
            chunkRequest.setChunkNumber(i);
            PullTopologyMapChunks req =
                new PullTopologyMapChunks();
            req.setChunkRequest(chunkRequest);
            req.setCmdbContext(getContext());
            PullTopologyMapChunksResponse res = null;

```

```

        try {
            res = getStub().pullTopologyMapChunks(req);
            TopologyMap map = res.getTopologyMap();
            topologyMap = mergeMaps(topologyMap, map);
        } catch (RemoteException e) {
            // 例外の処理
        } catch (UcmdbFaultException e) {
            // 例外の処理
        }
    }
}
return topologyMap;
}

```

```

private TopologyMap getTopologyMapResultFromCIs(CIs cis, ChunkInfo
chunkInfo) {
    TopologyMap topologyMap = new TopologyMap();
    if(chunkInfo.getNumberOfChunks() == 0) {
        CINode ciNode = new CINode();
        ciNode.setLabel("");
        ciNode.setCIs(cis);
        CINodes ciNodes = new CINodes();
        ciNodes.addCINode(ciNode);
        topologyMap.setCINodes(ciNodes);
    } else {

```

```

        for(int i=1 ; i <= chunkInfo.getNumberOfChunks() ; i++) {
            ChunkRequest chunkRequest =
                new ChunkRequest();
            chunkRequest.setChunkInfo(chunkInfo);
            chunkRequest.setChunkNumber(i);
            PullTopologyMapChunks req =
                new PullTopologyMapChunks();
            req.setChunkRequest(chunkRequest);
            req.setCmdContext(getContext());
            PullTopologyMapChunksResponse res = null;

```

```

try {
    res = getStub().pullTopologyMapChunks(req);
} catch (RemoteException e) {
    // 例外の処理
} catch (UcmdbFaultException e) {
    // 例外の処理
}
TopologyMap map = res.getTopologyMap();
topologyMap = mergeMaps(topologyMap, map);
}

```

```

// チャンクの解放
ReleaseChunks req = new ReleaseChunks();
req.setChunksKey(chunkInfo.getChunksKey());
req.setCmdbContext(getContext());

```

```

try {
    getStub().releaseChunks(req);
} catch (RemoteException e) {
    // 例外の処理
} catch (UcmdbFaultException e) {
    // 例外の処理
}
}
return topologyMap;
}

```

```

//=====================================================
/* 警告：結合は、各ノードに一意の名前が付けられている場合のみ正しく行われ
ます。これは、CI と関係の両ノードに当てはまります。*/
//=====================================================
private TopologyMap mergeMaps(TopologyMap topologyMap, TopologyMap
newMap) {
    for(int i=0 ; i < newMap.getCINodes().sizeCINodeList() ; i++ ) {
        CINode ciNode = newMap.getCINodes().getCINode(i);
        boolean alreadyExist = false;
        if(topologyMap.getCINodes() == null) {
            topologyMap.setCINodes(new CINodes());
        }
    }
}

```

```
for(int j=0 ; j < topologyMap.getCINodes().sizeCINodeList() ; j++) {  
    CNode ciNode2 = topologyMap.getCINodes().getCINode(j);  
    if(ciNode2.getLabel().equals(ciNode.getLabel())){
```

```
        CIs cisTOAdd = ciNode.getCIs();  
        CIs cis =  
            mergeCIsGroups  
            (topologyMap.getCINodes().getCINode(j).getCIs(),  
             cisTOAdd);  
        topologyMap.getCINodes().getCINode(j).setCIs(cis);  
        alreadyExist = true;  
    }  
}  
if(!alreadyExist) {  
    topologyMap.getCINodes().addCINode(ciNode);  
}  
}
```

```
for(int i=0 ; i < newMap.getRelationNodes().sizeRelationNodeList() ; i++ ) {  
    RelationNode relationNode =  
        newMap.getRelationNodes().getRelationNode(i);  
    boolean alreadyExist = false;  
    if(topologyMap.getRelationNodes() == null) {  
        topologyMap.setRelationNodes(new RelationNodes());  
    }  
}
```



```

for(int j=0 ;
    j < topologyMap.getRelationNodes().sizeRelationNodeList() ;
    j++) {
    RelationNode relationNode2 =
        topologyMap.getRelationNodes().getRelationNode(j);
    if(relationNode2.getLabel().equals(relationNode.getLabel())){
        Relations relationsTOAdd = relationNode.getRelations();
        Relations relations =
            mergeRelationsGroups
                (topologyMap.getRelationNodes().
                    getRelationNode(j).getRelations(),
                    relationsTOAdd);
        topologyMap.getRelationNodes().
            getRelationNode(j).setRelations(relations);
        alreadyExist = true;
    }
}

```

```

    if(!alreadyExist) {
        topologyMap.getRelationNodes().addRelationNode(relationNode);
    }
}

return topologyMap;
}

```

```

private Relations mergeRelationsGroups(Relations relations1, Relations
relations2) {
    for(int i=0 ; i < relations2.sizeRelationList() ; i++) {
        relations1.addRelation(relations2.getRelation(i));
    }
    return relations2;
}

```

```
private Cls mergeClsGroups(Cls cis1, Cls cis2) {  
    for(int i=0 ; i < cis2.sizeClList() ; i++) {  
        cis1.addCl(cis2.getCl(i));  
    }  
    return cis1;  
}  
  
}
```

 更新の例

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.update.AddCIsAndRelations;
import com.hp.ucmdb.generated.params.update.AddCIsAndRelationsResponse;
import com.hp.ucmdb.generated.params.update.UpdateCIsAndRelations;
import com.hp.ucmdb.generated.params.update.DeleteCIsAndRelations;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.update.CIsAndRelationsUpdates;
import com.hp.ucmdb.generated.types.update.ClientIDToCmdbID;

import java.rmi.RemoteException;
```

```
public class UpdateDemo extends Demo{
```

```
    public void getAddCIsAndRelationsDemo() {
        AddCIsAndRelations request = new AddCIsAndRelations();
        request.setCmdbContext(getContext());
        request.setUpdateExisting(true);
        CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
        CIs cis = new CIs();
        CI ci = new CI();
        ID id = new ID();
        id.setBase("temp1");
        id.setTemp(true);
```

```
        ci.setID(id);
        ci.setType("host");
```

```
        CIProperties props = new CIProperties();
        StrProps strProps = new StrProps();
        StrProp strProp = new StrProp();
        strProp.setName("host_key");
        String value = "blabla";
        strProp.setValue(value);
```

```
strProps.addStrProp(strProp);
props.setStrProps(strProps);
ci.setProps(props);
cis.addCI(ci);
updates.setCIsForUpdate(cis);
request.setCIsAndRelationsUpdates(updates);
```

```
try {
    AddCIsAndRelationsResponse response =
        getStub().addCIsAndRelations(request);
    for(int i = 0 ; i < response.sizeCreatedIDsMapList() ; i++) {
        ClientIDToCmdbID idsMap = response.getCreatedIDsMap(i);
        // 何らかの処理を実行
    }
} catch (RemoteException e) {
    // 例外の処理
} catch (UcmdbFaultException e) {
    // 例外の処理
}
}
```

```
public void getUpdateCIsAndRelationsDemo() {
    UpdateCIsAndRelations request = new UpdateCIsAndRelations();
    request.setCmdbContext(getContext());
```

```
CIsAndRelationsUpdates updates =
    new CIsAndRelationsUpdates();
CIs cis = new CIs();
CI ci = new CI();
ID id = new ID();
```

```
id.setBase("temp1");
id.setTemp(true);
ci.setID(id);
ci.setType("host");
CIProperties props = new CIProperties();
StrProps strProps = new StrProps();
```

```
StrProp hostKeyProp = new StrProp();
hostKeyProp.setName("host_key");
String hostKeyValue = "blabla";
hostKeyProp.setValue(hostKeyValue);
strProps.addStrProp(hostKeyProp);
```

```
StrProp hostOSProp = new StrProp();
hostOSProp.setName("host_os");
String hostOSValue = "winXP";
hostOSProp.setValue(hostOSValue);
strProps.addStrProp(hostOSProp);
```

```
StrProp hostDNSProp = new StrProp();
hostDNSProp.setName("host_dnsname");
String hostDNSValue = "dnsname";
hostDNSProp.setValue(hostDNSValue);
strProps.addStrProp(hostDNSProp);
```

```
props.setStrProps(strProps);
ci.setProps(props);
cis.addCI(ci);
updates.setCIsForUpdate(cis);
request.setCIsAndRelationsUpdates(updates);
```


```
try {
    getStub().updateCIsAndRelations(request);
} catch (RemoteException e) {
    // 例外の処理
} catch (UcmdbFaultException e) {
    // 例外の処理
}
}
```

```
public void getDeleteCIsAndRelationsDemo() {
    DeleteCIsAndRelations request =
        new DeleteCIsAndRelations();
    request.setCmdbContext(getContext());
    CIsAndRelationsUpdates updates =
        new CIsAndRelationsUpdates();
    CIs cis = new CIs();
    CI ci = new CI();
    ID id = new ID();
    id.setBase("stam");
    id.setTemp(true);
    ci.setID(id);
    ci.setType("host");
```

```
    CIProperties props = new CIProperties();
    StrProps strProps = new StrProps();
    StrProp strProp1 = new StrProp();
    strProp1.setName("host_key");
    String value1 = "for_delete";
    strProp1.setValue(value1);
    strProps.addStrProp(strProp1);
    props.setStrProps(strProps);
    ci.setProps(props);
    cis.addCI(ci);
    updates.setCIsForUpdate(cis);
    request.setCIsAndRelationsUpdates(updates);
```

```
    try {
        getStub().deleteCIsAndRelations(request);
    } catch (RemoteException e) {
        // 例外の処理
    } catch (UcmdbFaultException e) {
        // 例外の処理
    }
}

}
```

 クラス・モデルの例

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.classmodel.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.classmodel.UcmdbClassModelHierarchy;
import com.hp.ucmdb.generated.types.classmodel.UcmdbClass;

import java.rmi.RemoteException;

public class ClassmodelDemo extends Demo{
```

```
    public void getClassAncestorsDemo() {
        GetClassAncestors request =
            new GetClassAncestors();
        request.setCmdbContext(getContext());
        request.setClassName("className");
```

```
        try {
            GetClassAncestorsResponse response =
                getStub().getClassAncestors(request);
            UcmdbClassModelHierarchy hierarchy =
                response.getClassHierarchy();
        } catch (RemoteException e) {
            // 例外の処理
        } catch (UcmdbFaultException e) {
            // 例外の処理
        }
    }
}
```

```
public void getAllClassesHierarchyDemo() {
    GetAllClassesHierarchy request =
        new GetAllClassesHierarchy();
    request.setCmdbContext(getContext());
    try {
        GetAllClassesHierarchyResponse response =
            getStub().getAllClassesHierarchy(request);
        UcmdbClassModelHierarchy hierarchy =
            response.getClassesHierarchy();
    } catch (RemoteException e) {
        // 例外の処理
    } catch (UcmdbFaultException e) {
        // 例外の処理
    }
}
```

```
public void getCmdbClassDefinitionDemo() {
    GetCmdbClassDefinition request =
        new GetCmdbClassDefinition();
    request.setCmdbContext(getContext());
    request.setClassName("className");
```

```
    try {
        GetCmdbClassDefinitionResponse response =
            getStub().getCmdbClassDefinition(request);
        UcmdbClass ucmdbClass = response.getUcmdbClass();
    } catch (RemoteException e) {
        // 例外の処理
    } catch (UcmdbFaultException e) {
        // 例外の処理
    }
}

}
```


 影響分析の例

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.impact.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.impact.*;

import java.rmi.RemoteException;
```

```
/**
 * User:hbarkai
 * Date:Jul 17, 2007
 */
public class ImpactDemo extends Demo{

    // 影響ルール名 : impactExample
    // 影響のクエリ :
    //     ネットワーク
    //     |
    //     ホスト
    //     |
    //     IP
    // 影響の作用 : IP へのネットワークの影響 : 重大度 100%, カテゴリ : change
    //
    public void calculateImpactAndGetImpactPathDemo() {
        CalculateImpact request = new CalculateImpact();
        request.setCmdbContext(getContext());
        // ルート・コース ID の設定
        IDs ids = new IDs();
        ID id = new ID();
        id.setBase("rootCauseCmdbID");
        ids.addID(id);
```

```
request.setIDs(ids);
// 影響のカテゴリ設定
request.setImpactCategory("change");
// ルール名の設定
ImpactRuleNames impactRuleNames = new ImpactRuleNames();
ImpactRuleName impactRuleName = new ImpactRuleName();
impactRuleName.setBase("impactExample");
impactRuleNames.addImpactRuleName(impactRuleName);
request.setImpactRuleNames(impactRuleNames);
// 重大度の設定
request.setSeverity(100);
CalculateImpactResponse response =
    new CalculateImpactResponse();
```

```
request.setIDs(ids);
// 影響のカテゴリ設定
request.setImpactCategory("change");
// ルール名の設定
ImpactRuleNames impactRuleNames = new ImpactRuleNames();
ImpactRuleName impactRuleName = new ImpactRuleName();
impactRuleName.setBase("impactExample");
impactRuleNames.addImpactRuleName(impactRuleName);
request.setImpactRuleNames(impactRuleNames);
// 重大度の設定
request.setSeverity(100);
CalculateImpactResponse response =
    new CalculateImpactResponse();
```

```
try {
    response = getStub().calculateImpact(request);
} catch (RemoteException e) {
    // 例外の処理
```

```

} catch (UcmdbFaultException e) {
    // 例外の処理
}
Identifier identifier= response.getIdentfier();
Topology topology = response.getImpactTopology();
Relation relation = topology.getRelations().getRelation(0);
GetImpactPath request2 = new GetImpactPath();
// cmdb コンテキストの設定
request2.setCmdbContext(getContext());
// 影響識別子の設定
request2.setIdentifier(identifier);
// shallowRelation の設定
ShallowRelation shallowRelation = new ShallowRelation();
shallowRelation.setID(relation.getID());
shallowRelation.setEnd1ID(relation.getEnd1ID());
shallowRelation.setEnd2ID(relation.getEnd2ID());
shallowRelation.setType(relation.getType());
request2.setRelation(shallowRelation);

```

```

try {
    GetImpactPathResponse response2 =
        getStub().getImpactPath(request2);
    ImpactTopology impactTopology =
        response2.getImpactPathTopology();
} catch (RemoteException e) {
    // catch ステートメントの内容を変更するには
    // ファイル | 設定 | File Templates を使用
    e.printStackTrace();
} catch (UcmdbFaultException e) {
    // catch ステートメントの内容を変更するには
    // ファイル | 設定 | File Templates を使用
    e.printStackTrace();
}

}

```

```
public void getImpactRulesByGroupName() {
    GetImpactRulesByGroupName request =
        new GetImpactRulesByGroupName();
    // cmdb コンテキストの設定
    request.setCmdbContext(getContext());
    // グループ名リストの設定
    request.addRuleGroupNameFilter("groupName1");
    request.addRuleGroupNameFilter("groupName2");
```

```
    try {
        GetImpactRulesByGroupNameResponse response =
            getStub().getImpactRulesByGroupName(request);
        ImpactRules impactRules = response.getImpactRules();
    } catch (RemoteException e) {
        // 例外の処理
    } catch (UcmdbFaultException e) {
        // 例外の処理
    }
}
```

```
public void getImpactRulesByNamePrefix() {
    GetImpactRulesByNamePrefix request =
        new GetImpactRulesByNamePrefix();
    // cmdb コンテキストの設定
    request.setCmdbContext(getContext());
    // プレフィックス・リストの設定
    request.addRuleNamePrefixFilter("prefix1");
```

```

try {
    GetImpactRulesByNamePrefixResponse response =
        getStub().getImpactRulesByNamePrefix(request);
    ImpactRules impactRules = response.getImpactRules();
} catch (RemoteException e) {
    // 例外の処理
} catch (UcmdbFaultException e) {
    // 例外の処理
}
}
}

```

UCMDB の一般的なパラメータ

本項では、サービスのメソッドの最も一般的なパラメータについて説明します。詳細については、スキーマのドキュメントを参照してください。

本項の内容

- ▶ 85 ページ 「CmdbContext」
- ▶ 86 ページ 「ID」
- ▶ 86 ページ 「キー属性」
- ▶ 86 ページ 「ID のタイプ」
- ▶ 87 ページ 「CIProperties」
- ▶ 87 ページ 「タイプ名」
- ▶ 88 ページ 「構成アイテム (CI)」
- ▶ 88 ページ 「関係」

CmdbContext

すべての UCMDB Web Service API サービスの呼び出しには、**CmdbContext** 引数が必要です。**CmdbContext** は、サービスを呼び出すアプリケーションを識別する **callerApplication** 文字列です。**CmdbContext** は、ログの記録とトラブルシューティングに使用します。

ID

すべての CI と関係には ID フィールドがあります。このフィールドは、大文字と小文字が区別される ID 文字列と、ID が一時かどうかを示すオプションの temp フラグから構成されています。

キー属性

状況によっては、CI または関係を識別する際に、キー属性を UCMDB ID の代わりに使用できます。キー属性は、クラス定義で設定した ID_ATTRIBUTE を持つ属性です。

ユーザ・インタフェースの構成アイテム・タイプ属性のリストにおいて、キー属性の横にはキー・アイコンが表示されます。詳細については、『モデル管理』の「[属性の追加/属性の編集] ダイアログ・ボックス」を参照してください。API クライアント・アプリケーション内からのキー属性の識別の詳細については、30 ページ「getCmdbClassDefinition」を参照してください。

ID のタイプ

ID 要素には、実際の ID と一時 ID があり、空の場合もあります。

実際の ID は、UCMDB によって割り当てられた文字列で、データベース内のエンティティを識別します。一時 ID は、現在の要求において一意である任意の文字列です。空の ID は、値が割り当てられていないことを意味します。

一時 ID はクライアントによって割り当てられ、多くの場合、クライアントによって保存されている CI の ID を表します。この ID は、必ずしも UCMDB にすでに作成されているエンティティを表す必要はありません。一時 ID がクライアントによって渡されると、CI のキー・プロパティを使用して UCMDB によって既存のデータ構成アイテムが識別できる場合には、実際の ID を使って識別されたのと同じように、その CI は状況に適したものとして使用されます。

CI の実際の ID は、CI のタイプとキー・プロパティの組み合わせに基づいて UCMDB によって計算されます。関係の実際の ID は、関係のタイプ、関係に属する 2 つの CI の ID、関係のキー・プロパティに基づいています。このため、キー属性値は CI または関係の作成時に設定する必要があります。CI の作成時にキー・プロパティ値が指定されていない場合には、次に示す 2 つの可能性が考えられます。

- ▶ CIT に RANDOM_GENERATED_ID 修飾子が含まれている場合、サーバによって一意の ID が生成されます。

- ▶ CIT に RANDOM_GENERATED_ID 修飾子が含まれていない場合、例外がスローされます。

詳細については、『モデル管理』の「CI タイプ・マネージャ」を参照してください。

CIProperties

CIProperties 要素はコレクションから構成され、それぞれにコレクション名によって示されるタイプのプロパティを指定する、一連の名前と値の要素が含まれます。これらは必須のコレクションではないため、CIProperties 要素は任意の組み合わせのコレクションを含むことができます。

CIProperties は、CI 要素および 関係要素によって使用されます。詳細については、88 ページ「構成アイテム (CI)」と 88 ページ「関係」を参照してください。

プロパティのコレクションを次に示します。

- ▶ dateProps : DateProp 要素のコレクション
- ▶ doubleProps : DoubleProp 要素のコレクション
- ▶ floatProps : FloatProp 要素のコレクション
- ▶ intListProps : intListProp 要素のコレクション
- ▶ intProps : IntProp 要素のコレクション
- ▶ strProps : StrProp 要素のコレクション
- ▶ strListProps : StrListProp 要素のコレクション
- ▶ longProps : LongProp 要素のコレクション
- ▶ bytesProps : BytesProp 要素のコレクション
- ▶ xmlProps : XmlProp 要素のコレクション

タイプ名

タイプ名は、構成アイテム・タイプまたは関係タイプのクラス名です。タイプ名は、クラスを参照するためにコード内で使用します。表示名と間違えないように注意してください。表示名はクラスが示されるユーザ・インタフェースに表示されますが、コード内では意味を持ちません。

構成アイテム (CI)

CI 要素は、ID、type、および props コレクションから構成されます。

UCMDB 更新メソッドを使用して CI を更新する場合、ID 要素には、実際の UCMDB ID またはクライアントによって割り当てられた一時 ID を含めることができます。一時 ID を使用する場合は、temp フラグを true に設定します。アイテムを削除する場合、ID は空でもかまいません。UCMDB クエリ・メソッドは、実際の ID を入力パラメータとして取り、実際の ID をクエリ結果に返します。

type は、CI タイプ・マネージャで定義した任意のタイプ名を指定できます。詳細については、『モデル管理』の「CI タイプ・マネージャ」を参照してください。

props 要素は、CIProperties コレクションです。詳細については、87 ページ「CIProperties」を参照してください。

関係

関係は、2 つの構成アイテムをリンクするエンティティです。関係要素は、ID、タイプ、リンク対象の 2 つのアイテムの識別子 (end1ID と end2ID)、および props コレクションから構成されます。

UCMDB 更新メソッドを使用して関係を更新する場合、関係の ID の値には、実際の UCMDB ID または一時 ID を使用できます。アイテムを削除する場合、ID は空でもかまいません。UCMDB クエリ・メソッドは、実際の ID を入力パラメータとして取り、実際の ID をクエリ結果に返します。

関係タイプは、関係のインスタンスが作成される HP UCMDB クラスのタイプ名です。タイプは、UCMDB に定義した関係タイプのいずれでもかまいません。クラスまたはタイプの詳細については、29 ページ「UCMDB クラス・モデルの問い合わせ」を参照してください。

詳細については、『モデル管理』の「CI タイプ・マネージャ」を参照してください。

関係の 2 つの終了 ID は、現在の関係の ID を作成するのに使用されるため空の ID を指定することはできません。しかし、これらの終了 ID には、クライアントによって割り当てられた一時 ID を使用することができます。

props 要素は、CIProperties コレクションです。詳細については、87 ページ「CIProperties」を参照してください。

UCMDB 出力パラメータ

本項では、サービス・メソッドの最も一般的な出力パラメータについて説明します。詳細については、スキーマのドキュメントを参照してください。

本項の内容

- ▶ 89 ページ 「CIs」
- ▶ 89 ページ 「ShallowRelation」
- ▶ 89 ページ 「Topology」
- ▶ 89 ページ 「CINode」
- ▶ 90 ページ 「RelationNode」
- ▶ 90 ページ 「TopologyMap」
- ▶ 90 ページ 「ChunkInfo」

CIs

CIs は、CI 要素のコレクションです。

ShallowRelation

ShallowRelation は、2つの構成アイテムをリンクするエンティティで、ID、タイプ、およびリンク対象の2つのアイテムの識別子（end1ID と end2ID）から構成されます。関係タイプは、関係のインスタンスが作成される UCMDB クラスのタイプ名です。タイプは、UCMDB に定義した関係タイプのいずれでもかまいません。

Topology

Topology は、CI 要素と関係のグラフです。Topology は、CIs コレクション、および1つ以上の関係要素を含む Relations コレクションから構成されています。

CINode

CINode は、ラベルを持つ CIs コレクションから構成されています。CINode のラベルは、クエリで使用する TQL のノードで定義したラベルです。

RelationNode

RelationNode は、ラベルを持つ **Relation** コレクションのセットです。
RelationNode のラベルは、クエリで使用する TQL のノードで定義したラベルです。

TopologyMap

TopologyMap は、TQL クエリに一致するクエリ計算を出力したものです。
TopologyMap のラベルは、クエリで使用する TQL で定義したノード・ラベルです。

TopologyMap のデータは、次の形式で返されます。

- ▶ **CINodes** : 1 つ以上の **CINode** です (89 ページ「**CINode**」を参照してください)。
- ▶ **relationNodes** : 1 つ以上の **RelationNode** です (90 ページ「**RelationNode**」を参照してください)。

これら 2 つの構造内のラベルによって、構成アイテムと関係のリストが配列されます。

ChunkInfo

クエリによって大量のデータが返されると、サーバはデータをチャンクというセグメントに分割して保存します。チャンクに分割したデータを取得するためにクライアントが使用する情報は、クエリによって返される **ChunkInfo** 構造に配置されます。**ChunkInfo** は、取得する必要がある **numberOfChunks** と **chunksKey** から構成されます。**chunksKey** は、この特定のクエリ呼び出しに対するサーバ上のデータの一意の識別子です。

詳細については、24 ページ「サイズの大きい応答の処理」を参照してください。

第 3 章

HP Universal CMDB Java API

本章では、サードパーティ製ツールまたはカスタム・ツールで HP Universal CMDB Java API を使用して、データや計算を抽出したり UCMDB (Universal Configuration Management database) にデータを書き込む方法について説明します。

本章は、オンラインのドキュメント・ライブラリで使用できる API Javadoc と併せて使用してください。

本章の内容

概念

- ▶ 規則 (92 ページ)
- ▶ HP Universal CMDB Java API の使用 (92 ページ)
- ▶ アプリケーションの一般的な構造 (94 ページ)

タスク

- ▶ API Jar ファイルの取得 (95 ページ)
- ▶ インテグレーション・ユーザの作成 (95 ページ)

参照先

- ▶ HP Universal CMDB Java API リファレンス (96 ページ)
- ▶ 使用例 (96 ページ)
- ▶ 例 (98 ページ)

規則

本章では、次の表記規則を使用します。

- ▶ **UCMDB** は、Universal Configuration Management database 自体を指します。
HP Universal CMDB はアプリケーションを意味します。
- ▶ UCMDB の要素およびメソッド引数は、インタフェース内で指定される場合に記述します。

HP Universal CMDB Java API の使用

HP Universal CMDB Java API は、アプリケーションを Universal CMDB (UCMDB) に統合するために使用します。この API により、次を実施するメソッドが提供されます。

- ▶ CMDB での CI と関係の追加、削除、および更新
- ▶ クラス・モデルに関する情報の取得
- ▶ what-if シナリオの実行
- ▶ 構成アイテムおよび関係に関する情報の取得

構成アイテムと関係に関する情報を取得するメソッドでは、一般的にトポロジ・クエリ言語 (TQL) を使用します。詳細については、『**モデル管理**』の「トポロジ・クエリ言語」を参照してください。

HP Universal CMDB Java API のユーザは、次のことを十分理解している必要があります。

- ▶ Java プログラミング言語
- ▶ HP Universal CMDB

本項の内容

- ▶ 93 ページ「API の使用」
- ▶ 93 ページ「権限」

API の使用

API を使用すると、多くのビジネス要件を満たすことができます。たとえば次のような場合です。

- ▶ サードパーティ製のシステムは、利用できる構成アイテム (CI) に関する情報をクラス・モデルに問い合わせることができます。
- ▶ サードパーティ製のアセット管理ツールは、そのツールのみで利用できる情報を使って UCMDb を更新できるため、アセット管理ツールのデータを HP アプリケーションで収集したデータと統一できます。
- ▶ 多くのサードパーティ製のシステムは、UCMDb にデータを入力して、変更内容を追跡し影響分析を実行できる中心的な UCMDb を作成できます。
- ▶ サードパーティ製のシステムは、ビジネス・ロジックに従ってエンティティと関係を作成し、データを UCMDb に書き込んで UCMDb のクエリ機能を活用できます。
- ▶ ほかのシステムは、変更分析向けの影響分析メソッドを使用できます。

権限

管理者により、API に接続するためのログイン資格情報が提供されます。API クライアントには、UCMDb で定義されているインテグレーション・ユーザのユーザ名とパスワードが必要です。これらのユーザは UCMDb の実際のユーザを表すものではなく、UCMDb に接続するアプリケーションを表します。

詳細については、95 ページ「インテグレーション・ユーザの作成」を参照してください。

アプリケーションの一般的な構造

静的ファクトリ「UcmdbServiceFactory」のみが存在します。このファクトリは、アプリケーションのエントリ・ポイントです。UcmdbServiceFactory は getServiceProvider メソッドを公開します。これらのメソッドは UcmdbServiceProvider インタフェースのインスタンスを返します。クライアントは HTTP を通じてサーバと通信します。

クライアントは、インタフェース・メソッドを使用してほかのオブジェクトを作成します。たとえば、新しいクエリ定義を作成するには、クライアントは次の手順を実行します。

- 1 メイン UCMDDB サービス・オブジェクトからクエリ・サービスを取得します。
- 2 サービス・オブジェクトからクエリ・ファクトリ・オブジェクトを取得します。
- 3 ファクトリから新しいクエリ定義を取得します。

```
UcmdbServiceProvider provider =
    UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
UcmdbService = provider.connect(provider.createCredentials(USERNAME,
    PASSWORD), provider.createClientContext("Test"));
TopologyQueryService queryService = ucmdbService.getTopologyQueryService();
TopologyQueryFactory factory = queryService.getFactory();
QueryDefinition queryDefinition = factory.createQueryDefinition("Test Query");
queryDefinition.addNode("Node").ofType("host");
Topology topology = queryService.executeQuery(queryDefinition);
System.out.println("There are " + topology.getAllCIs().size() + " hosts in uCMDB");
```

UcmdbService から使用できるサービスは次のとおりです。

サービス・メソッド	用途
getClassModelService	CI のタイプと関係に関する情報
getImpactAnalysisService	IT ユニバース内の変更の影響の分析
getTopologyQueryService	IT ユニバースに関する情報の取得
getTopologyUpdateService	IT ユニバース内の情報の変更

API Jar ファイルの取得

BAC または UCMDDB サーバ・インストールから ucmdb-api.jar を取得します。サーバ上の AppServer¥webapps ディレクトリにある ucmdb-api.war から jar ファイルを取り出します。war アーカイブ内では、ucmdb-api.jar は WEB-INF¥lib ディレクトリにあります。

クラスパス内の ucmdb-api.jar を使用して、アプリケーションをコンパイルして実行します。

インテグレーション・ユーザの作成

この API を使用して書き込まれたアプリケーションにインテグレーション・ユーザでログオンするよう設定します。

インテグレーション・ユーザを作成するには、次の手順を実行します。

- 1 JMX Agent コンソールにログオンします。
 - a URL は `http:// < UCMDDB ホスト > :8080/jmx-console` です。
 - b 標準設定のユーザとパスワードは両方とも「admin」です。
- 2 [JMX Agent View] ページで、「MAM」というラベルのセクションを見つけます。
- 3 **service=MAM Security Services** をクリックします。
- 4 [JMX MBean View] で、「`java.lang.String createIntegrationUser()`」を見つけます。
 - a [userName] および [password] フィールドを入力します。
 - b [Invoke] をクリックします。
- 5 [Back to MBean View] をクリックしてさらにユーザを作成するか、JMX Agent コンソールを閉じます。
- 6 管理者として UCMDDB にログオンします。
- 7 [設定] タブで、**パッケージ・マネージャ**を実行します。
- 8 [新規作成] アイコンをクリックします。
- 9 新しいパッケージ名を入力して、[次へ] をクリックします。
- 10 [設定] の下の [リソースの選択] タブで、[インテグレーション ユーザ] をクリックします。

- 11 JMX Agent コンソールを使用して作成したユーザを選択します。
- 12 [次へ] をクリックし、次に [終了] をクリックします。新しいパッケージが、パッケージ・マネージャの [パッケージ名] の一覧に表示されます。
- 13 API アプリケーションを実行するユーザにパッケージをデプロイします。
詳細については、「パッケージのデプロイ」を参照してください。

HP Universal CMDB Java API リファレンス

使用できる API の完全なドキュメントについては、『HP UCMDB Java API リファレンス』を参照してください。ファイルは次のフォルダにあります。

```
¥¥ < HP Universal CMDB のルート・ディレクトリ > ¥UCMDBServer¥j2¥  
AppServer¥webapps¥site.war¥amdocs¥eng¥doc_lib¥Integrations¥UCMDB_JavaAPI¥  
index.html
```

使用例

次の使用例は2つのシステムを想定しています。

- ▶ HP Universal CMDB サーバ
- ▶ 構成アイテムのリポジトリを含むサードパーティ製のシステム

本項の内容

- ▶ 97 ページ「UCMDB の作成」
- ▶ 97 ページ「UCMDB への問い合わせ」
- ▶ 97 ページ「クラス・モデルへの問い合わせ」
- ▶ 98 ページ「変更の影響の分析」

UCMDB の作成

使用例：

- ▶ サードパーティ製のアセット管理は、アセット管理でのみ使用できる情報で UCMDB を更新します。
- ▶ 多くのサードパーティ製のシステムは、UCMDB にデータを入力して、変更内容を追跡し影響分析を実行できる中心的な CMDB を作成します。
- ▶ サードパーティ製のシステムは、サードパーティのビジネス・ロジックに従って構成アイテムと関係を作成し、CMDB クエリ機能を活用します。

UCMDB への問い合わせ

使用例：

- ▶ サードパーティ製のシステムは、SAP TQL の結果を取得することによって、SAP システムを表す構成アイテムと関係を取得します。
- ▶ サードパーティ製のシステムは、過去 5 時間以内に追加または変更された Oracle サーバのリストを取得します。
- ▶ サードパーティ製のシステムは、ホスト名に部分文字列 **lab** が含まれるサーバのリストを取得します。
- ▶ サードパーティ製のシステムは、隣接項目を取得することによって、特定の CI に関係する要素を検出します。

クラス・モデルへの問い合わせ

使用例：

- ▶ サードパーティ製のシステムでは、ユーザは UCMDB から取得するデータのセットを指定できます。ユーザ・インタフェースはクラス・モデル上に構築し、ユーザに利用可能なプロパティを表示して、必要なデータを求めることができます。ユーザは、取得する情報を選択できます。
- ▶ サードパーティ製のシステムは、ユーザが UCMDB ユーザ・インタフェースにアクセスできないときに、クラス・モデルを探索します。

変更の影響の分析

使用例：

サードパーティ製のシステムは、指定したホストに対する変更の影響を受けるビジネス・サービスのリストを出力します。

例

本項の内容

- ▶ 98 ページ「エントリ・ポイントの例」
- ▶ 99 ページ「クエリの例」
- ▶ 100 ページ「トポロジ・クエリの例」
- ▶ 101 ページ「トポロジ更新の例」
- ▶ 101 ページ「影響分析の例」

エントリ・ポイントの例

```
final String HOST_NAME = "localhost";
final int PORT = 8080;
UcmdbServiceProvider provider =
    UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
final String USERNAME = "integration_user";
final String PASSWORD = "integration_password";
Credentials credentials =
    provider.createCredentials(USERNAME, PASSWORD),
ClientContext clientContext = provider.createClientContext("Example");
UcmdbService ucmdbService = provider.connect(credentials, clientContext);
```

クエリの例

次の例は、単一クラス定義の取得と、すべての CIT 定義とその属性のリストの取得を示しています。

クラス定義の取得

```
ClassModelService classModelService
= ucmdbService.getClassModelService();
String typeName = "disk";
ClassDefinition def =
  classModelService.getClassDefinition(typeName);
System.out.println("Type " + typeName + " is derived from type "
+ def.getParentClassName());
System.out.println("Has " + def.getChildClasses().size() +
" derived types");
System.out.println("Defined and inherited attributes:");
for (Attribute attr : def.getAllAttributes().values()) {
  System.out.println("Attribute " + attr.getName() +
" of type " + attr.getType());
}
```

CIT 定義と属性のリストを取得

この例では、1つの CIT の属性のクエリを実行し、その名前とタイプを印刷します。

```
ClassModelService classModelService =
  ucmdbService.getClassModelService();
for (ClassDefinition def : classModelService.getAllClasses()) {
  System.out.println("Type " + def.getName() +
  "(" + def.getDisplayName() + ") is derived from type "
+ def.getParentClassName());
  System.out.println
    ("Has " + def.getChildClasses().size() + " derived types");
  System.out.println
    ("Defined and inherited attributes:");
  for (Attribute attr : def.getAllAttributes().values()) {
    System.out.println
      ("Attribute " + attr.getName() +
      " of type " + attr.getType());
  }
}
```

 トポロジ・クエリの例

```
TopologyQueryService queryService =
ucmdbService.getTopologyQueryService();
TopologyQueryFactory queryFactory =
queryService.getFactory();
QueryDefinition queryDefinition =
queryFactory.createQueryDefinition
("Get hosts with more than one network interface");
String hostNodeName = "Host";
QueryNode hostNode =
queryDefinition.addNode(hostNodeName).ofType("host").queryProperty("display_
label");
QueryNode ipNode =
queryDefinition.addNode("IP").ofType("ip").queryProperty("ip_address");
hostNode.linkedTo(ipNode).withLinkOfType("contained").atLeast(2);
Topology topology = queryService.executeQuery(queryDefinition);
Collection<TopologyCI> hosts = topology.getCIsByName(hostNodeName);
for (TopologyCI host : hosts) {
System.out.println("Host " + host.getPropertyValue("display_label"));
for (TopologyRelation relation : host.getOutgoingRelations()) {
System.out.println
(" has IP " + relation.getEnd2CI().getPropertyValue("ip_address"));
}
}
```

トポロジ更新の例

```
TopologyUpdateService topologyUpdateService =
    ucmdbService.getTopologyUpdateService();
TopologyUpdateFactory topologyUpdateFactory =
    topologyUpdateService.getFactory();
TopologyModificationData topologyModificationData =
    topologyUpdateFactory.createTopologyModificationData();
CI host = topologyModificationData.addCI("host");
host.setPropertyValue("host_key", "test1");
CI ip = topologyModificationData.addCI("ip");
ip.setPropertyValue("ip_address", "127.0.0.10");
ip.setPropertyValue("ip_domain", "DefaultDomain");
topologyModificationData.addRelation("contained", host, ip);
topologyUpdateService.create
    (topologyModificationData, CreateMode.IGNORE_EXISTING);
```

影響分析の例

```
ImpactAnalysisService impactAnalysisService =
    ucmdbService.getImpactAnalysisService();
ImpactAnalysisFactory impactFactory =
    impactAnalysisService.getFactory();
ImpactAnalysisDefinition definition =
    impactFactory.createImpactAnalysisDefinition();
definition.addTriggerCI(disk).withSeverity
    (impactFactory.getSeverityByName("Warning(2)"));
definition.useAllRules();
ImpactAnalysisResult impactResult =
    impactAnalysisService.analyze(definition);
AffectedTopology affectedCIs =
    impactResult.getAffectedCIs();
for (AffectedCI affectedCI : affectedCIs.getAllCIs()) {
    System.out.println("Affected " +
        affectedCI.getType() + " " + affectedCI.getId() +
        " - severity " + affectedCI.getSeverity());
}
```


第 4 章

ディスカバリおよび依存関係マップ Web サービス API

詳細については、『ディスカバリおよび依存関係マップ』の「HP Discovery and Dependency Mapping Web Service API」を参照してください。

第II部

フェデレーションと調整

第 5 章

フェデレート CMDB の概要

本章では、フェデレート CMDB の機能に関する情報を提供します。

本章の内容

概念

- ▶ フェデレート CMDB – 概要 (108 ページ)
- ▶ アダプタ (108 ページ)
- ▶ 複数のデータ・ストアからのデータの取得 (109 ページ)
- ▶ 外部データ・ストアからの属性の取得 (111 ページ)
- ▶ マッピング情報 (112 ページ)

タスク

- ▶ フェデレート・データを使った作業 – ワークフロー (112 ページ)
- ▶ フェデレート・アダプタの暗号化パスワードの変更 (114 ページ)

参照先

- ▶ フェデレート CMDB のユーザ・インタフェース (115 ページ)

フェデレート CMDB — 概要

CMDB 実装には、しばしばフェデレーションが含まれます。フェデレーションでは、ほかのソースから CMDB にデータを取り込みますが、データの制御はデータ・ソースが引き続き行います。

フェデレート CMDB を使用して、次の点を明確にします。

- ▶ 指定した期間ごとに指定した回数以上変更された（SAP などの）特定のアプリケーションのホスト。
- ▶ 指定した時間ごとに指定した回数以上変更されたアプリケーションの名前。
- ▶ 指定した時間ごとに指定した回数以上変更されたモデル内のホスト。

アダプタ

HP Universal CMDB API を使用して、アダプタをセットアップし、CMDB のさまざまなソースを元にフェデレートするデータを使用するようにします。詳細については、『[参照情報](#)』の「[HP Universal CMDB Web サービス API](#)」を参照してください。

次の事前定義されているアダプタを使用して、CMDB のさまざまなソースをフェデレートできます。

- ▶ **CmdbChangesAdapter** : 変更があるかどうか UCMDB のクエリを行うソース・アダプタを定義します。アダプタはフェデレート TQL および UCMDB 履歴データベースまたはスキーマで動作します（詳細については、272 ページ「[CmdbChanges アダプタ](#)」を参照してください。）
- ▶ **CmdbHistoryAdapter** : UCMDB 履歴データベースからデータを取得するアダプタを定義します。注：履歴アダプタはレプリケーションをサポートしていません。つまり、このアダプタを使用してレプリケーション・ジョブ（あるデータ・ストアから別のデータ・ストアにデータをコピーする）を定義することはできません。
- ▶ **CmdbRmiAdapter** : RMI API を使用して外部 CMDB データ・ストアからデータを取得するアダプタを定義します。注：標準設定では、このアダプタはフェデレート・レプリケーションのみをサポートします。フェデレート・クエリをサポートするには、HP ソフトウェア・サポートにお問い合わせください。

- ▶ **CmdbSoapAdapter** : SOAP API を使用して外部 CMDB データ・ストアからデータを取得するアダプタを定義します。注：このアダプタはフェデレート・クエリをサポートしません。フェデレート・レプリケーションでターゲットとしてのみ使用できます。
- ▶ **ServiceDeskAdapter** : HP ServiceCenter および HP Service Manager からのデータの取得をサポートするアダプタを定義します。このアダプタは、Web サービス API を使用して ServiceCenter または Service Manager に接続し、データを取得します。このアダプタを使った作業の詳細については、第8章「HP ServiceCenter/Service Manager Adapter」を参照してください。
- ▶ **GenericDBAdapter** : あらゆるタイプのリレーショナル・データベースと統合し、データベースに対してフェデレート TQL を実行できるようにするデータベース・アダプタを設定するためのプラットフォームです。このアダプタを使った作業の詳細については、第6章「汎用データベース・アダプタ」を参照してください。

Federation Framework SDK は、HP Universal CMDB とアダプタ間のメディアエータとして、また、フェデレート・アダプタのコンテナとして機能します。詳細については、209 ページ「Federation Framework SDK」を参照してください。

データ・ストア作成時のアダプタの選択の詳細については、116 ページ「[データストア] タブ」を参照してください。

複数のデータ・ストアからのデータの取得

FTQL 計算中、複数のデータ・ストアから同じ CIT のデータを取得できます。ローカル UCMDB や外部データ・ストア、または複数の外部データ・ストアからデータを取得できます。

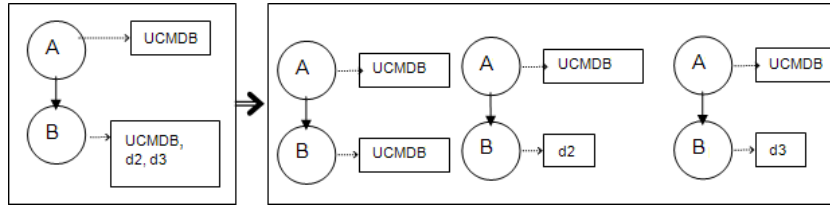
同一のアダプタを使用して複数の場所から、または複数のアダプタを使用して複数の場所からデータを取得できます。

外部データ・ストアから取得した各 CI には、CI の取得元のデータ・ストアを示す属性（[Created By]）が含まれています。

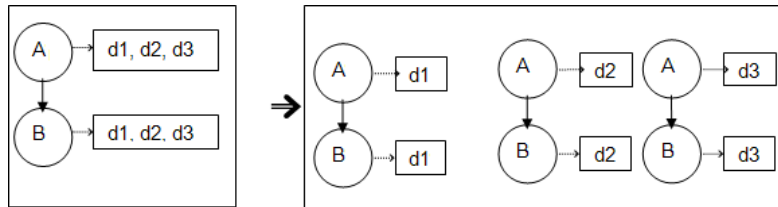
マッピング情報の詳細については、112 ページ「マッピング情報」を参照してください。

制限事項

- ▶ 2つのデータ・ストア間に仮想リンクがある場合、HP Universal CMDB は次の場合にのみマッピングをサポートします。
 - ▶ リンクの一方向の端に UCMDB データ・ストアがあり、もう一方の端に複数のデータ・ストアがあります。A のデータ・ストア (UCMDB) と B のデータ・ストア (UCMDB, d2, d3) についてデカルト積が計算されます。



- ▶ リンクの両方の端に同じデータ・ストアがあります。リンクは各データ・ストアの内部リンクで、マッピングは必要ありません。



- ▶ 複数のデータ・ストアから取得されたデータは調整されません。2つのデータ・ストアに同じ CIT がある場合、HP Universal CMDB ではこの CIT が2回示されます。2つのデータ・ストアに同じ属性がある場合、1つの属性のみが UCMDB とフェデレートします。

外部データ・ストアからの属性の取得

- ▶ コア CI データが UCMDDB に格納されている場合、外部データ・ストアから CI の属性を取得できます。
- ▶ コア・データ・ストアは UCMDDB でなければなりません。
- ▶ CIT は属性を定義するデータ・ストアになければなりません。
- ▶ 複数のデータ・ストアから同じ属性を取得できます。

取得オプションの詳細については、121 ページ「[サポートされる CI タイプ] タブ」の「CI タイプ取得モード」フィールドを参照してください。

- ▶ CIT を外部 CIT や外部 CIT 属性に割り当てることはできません。
- ▶ (CIT データをフェデレートする) アダプタが CIT に対するマッピング情報 (調整) をサポートする場合、この CIT は外部属性をサポートできます。

使用例

- ▶ システム内の SMS または Altiris デスクトップを検出する必要があります。デスクトップ CIT はコア CIT で、すでに UCMDDB と同期しています。ただし、UCMDDB にすべてのデスクトップ・データを格納しないでください。これは非効率で不要です。名前や MAC アドレスなどのコア属性を UCMDDB に格納し、デスクトップのその他の詳細は外部属性として SMS および Altiris の 2 つのデータ・ストアに定義するだけで十分です。
- ▶ VMware は、ハードウェア・リソースを動的かつ透過的に割り当てる仮想マシン・モニタ (hypervisor) を含む仮想マシンを作成します。単一の物理コンピュータ上で複数のオペレーティング・システムを同時に実行できます。リソース (メモリなど) は動的に割り当てられるので、DDM はこれらのリソースを検出できません (DDM は 24 時間おきに実行されますが、リソース・データは 1 時間ごとに変わる可能性があります)。HP Universal CMDB を常にリアルタイム・データで更新できるようにするには、データを 2 つに分割します。1 つは仮想ホストのコア・データで、UCMDDB に置かれ、検出されます。もう 1 つはリソース属性で、外部ソースから取得されます。この使用例では、これらの属性のデータは、UCMDDB と VMware の 2 つのデータ・ストアから取得されます。

マッピング情報

フェデレート・クエリは、マッピング・エンジンを使用して、コア・データ・ストアの表面的な CI（CI のコア属性）と外部データ・ストアの属性を結合します。

マッピング・エンジンの詳細については、214 ページ「FTQL 用の Federation Framework フロー」を参照してください。

フェデレーションに含める属性の選択の詳細については、121 ページ「[サポートされる CI タイプ] タブ」を参照してください。

フェデレート・データを使った作業 – ワークフロー

本項では、HP Universal CMDB API を使用して、アダプタをセットアップし、CMDB のさまざまなソースを元に、フェデレート・データを使用する方法について説明します。

本項の内容

- ▶ 112 ページ「前提条件」
- ▶ 113 ページ「データ・ストアの作成」
- ▶ 113 ページ「データ・ストアの複製」
- ▶ 113 ページ「ビューの作成」
- ▶ 113 ページ「ビュー・インスタンスを IT ユニバース・マネージャに表示」
- ▶ 113 ページ「レポートの表示」

1 前提条件

アダプタをセットアップします。詳細については、231 ページ「新しい外部データ・ストア用のアダプタの追加」を参照してください。既存のアダプタの詳細については、108 ページ「アダプタ」を参照してください。

2 データ・ストアの作成



[管理] > [設定] > [フェデレート CMDB] にアクセスします。[新規データストア] ボタンをクリックして、[新規データストア] ダイアログ・ボックスを開きます。詳細については、118 ページ「[新規データストア] ウィザード」を参照してください。

3 データ・ストアの複製

レプリケーション向けのデータ・ストアを使用します。[管理] > [設定] > [フェデレート CMDB] > [レプリケーション ジョブ] タブにアクセスします。[新規レプリケーション ジョブ] ボタンをクリックして、[レプリケーション ジョブ] ダイアログ・ボックスを開きます。詳細については、124 ページ「[レプリケーション ジョブ] ダイアログ・ボックス」を参照してください。

4 ビューの作成

詳細については、『モデル管理』の「ビュー・マネージャの概要」を参照してください。

5 ビュー・インスタンスを IT ユニバース・マネージャに表示

詳細については、『モデル管理』の「IT ユニバース・マネージャの概要」を参照してください。

6 レポートの表示

詳細については、『モデル管理』の「トポロジ・レポート・マネージャ」を参照してください。

フェデレート・アダプタの暗号化パスワードの変更

新しいデータ・ストア・アダプタを定義する場合、パスワードを入力する必要があります。標準設定では、このパスワードは暗号化されており、暗号化はユーザに対して完全に透過的に実行されます。

次の手順で、新しい鍵ファイルの生成と、既存の鍵ファイルの変更方法について説明します。HP Universal CMDB には標準設定の鍵が含まれているため、この手順は任意です。

注：パスワードは、暗号化されていないバージョンからのアップグレード・プロセスで暗号化されます。

暗号鍵は `C:\hp\UCMDB\UCMDBServer\j2f\fcmdb\key.bin` にあります。

新しい鍵ファイルを生成するには、次の手順を実行します。

- 1 Web ブラウザを起動して次のアドレスを入力します。

```
http:// <マシン名または IP アドレス> . <ドメイン名> :8080/jmx-console
```

<マシン名または IP アドレス>には、HP Universal CMDB がインストールされているマシンを指定します。ユーザ名とパスワードでログインしなければならない場合もあります。

- 2 [Topaz] > [service=FCmdb Config Services] リンクをクリックします。
- 3 [JMX MBEAN View] ページで次の演算を見つけます。 `java.lang.String generateNewKeyFile()`
- 4 [customerID] フィールドに「1」を入力します。
- 5 [Invoke] をクリックします。新しい鍵ファイルが生成されてアプリケーションにロードされたことを示すメッセージが表示されます。

鍵ファイルを変更するには、次の手順を実行します。

- 1 鍵ファイルの準備を整え、場所を書き留めておきます。
- 2 [JMX MBEAN View] ページで次の演算を見つけます。 `java.lang.String importKeyFile()`

- 3 [customerId] フィールドに「1」を入力し、[newKeyFileLocation] フィールドに新しい鍵へのフル・パスを入力します。
- 4 [Invoke] をクリックします。メッセージが表示されます。

フェデレート CMDB のユーザ・インタフェース






本項の内容

- ▶ 116 ページ 「[データ ストア] タブ」
- ▶ 117 ページ 「[フェデレート CMDB] ウィンドウ」
- ▶ 118 ページ 「[新規データ ストア] ウィザード」
- ▶ 124 ページ 「[レプリケーション ジョブ] ダイアログ・ボックス」
- ▶ 125 ページ 「[レプリケーション ジョブ統計] ウィンドウ」
- ▶ 127 ページ 「[レプリケーション ジョブ] タブ」

[データストア] タブ

<p>説明</p>	<p>既存のデータ・ストアの編集，または新しいデータ・ストアの作成ができます。</p> <p>利用方法：[管理] > [設定] > [フェデレート CMDB] > [データストア] タブを選択します。</p>
<p>重要情報</p>	<p>[データストア] には次のタブがあります。</p> <ul style="list-style-type: none"> ▶ [プロパティ]：外部データ・ソースにアクセスするアダプタを設定できます。詳細については，119 ページ「[プロパティ] タブ」を参照してください。 ▶ [サポートされる CI タイプ]：データ・ストアがサポートする CIT を選択できます。たとえば，フェデレート TQL (FTQL) に特定の CIT を示すノードが含まれている場合，CIT インスタスはこの外部データ・ストアに受け入れられます。詳細については，121 ページ「[サポートされる CI タイプ] タブ」を参照してください。 ▶ [サポートされるクエリ]：データのレプリケーション向けにこのデータ・ストアがサポートするクエリを選択できます。詳細については，123 ページ「[サポートされるクエリ] タブ」を参照してください。

含まれている要素は次のとおりです（ラベルのない GUI 要素は山括弧で囲んで示します）。

GUI 要素	説明
	<p>データ・ストアを追加します。詳細については，118 ページ「[新規データストア] ウィザード」を参照してください。</p>
	<p>データ・ストアを削除します。</p>
	<p>変更内容を保存します。</p>
 更新	<p>ページを更新します。</p>
	<p>アダプタに変更が加えられた場合，再ロードします。</p>

GUI 要素	説明
<定義されたアダプタのリスト>	<p>[名前] : データ・ストアに付けた名前です。</p> <p>[アダプタ] : このデータ・ストア用のアダプタのタイプです。</p>
[ビュー]	<p>詳細については、次を参照してください。</p> <ul style="list-style-type: none"> ▶ 116 ページ「[データ ストア] タブ」 ▶ 121 ページ「[サポートされる CI タイプ] タブ」

[フェデレート CMDB] ウィンドウ

説明	<p>外部データ・ストアの定義およびレプリケーション・ジョブの作成ができます。</p> <p>利用方法 : [管理] > [設定] > [フェデレート CMDB] を選択します。</p>
重要情報	<p>データ・ストアは [新規データ ストア] ウィザードで作成します。詳細については、118 ページ「[新規データ ストア] ウィザード」を参照してください。</p>
ほかのタスク	<p>112 ページ「フェデレート・データを使った作業 - ワークフロー」</p>
参考リンク	<ul style="list-style-type: none"> ▶ 116 ページ「[データ ストア] タブ」 ▶ 127 ページ「[レプリケーションジョブ] タブ」 ▶ 108 ページ「アダプタ」

[新規データストア] ウィザード

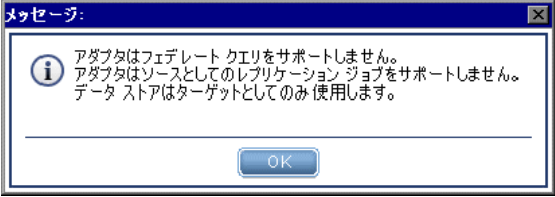
<p>説明</p>	<p>外部データ・ソースにアクセスするアダプタを使用するデータ・ストアを定義できます。</p> <p>利用方法：[管理] > [設定] > [フェデレート CMDB] > [データストア] タブを選択し，[新規データストア] ボタンをクリックします。</p>
<p>重要情報</p>	<p>アダプタの機能に応じて，次のいずれかのアダプタを使用してデータ・ストアを定義します。</p> <ul style="list-style-type: none"> ▶ CmdbChangesAdapter：フェデレート・レプリケーションのみをサポートし，ソースとして使用できます。したがって，[プロパティ] および [サポートされるクエリ] タブの入力が必要です。 ▶ CmdbHistoryAdapter：フェデレート TQL クエリのみをサポートします。したがって，[プロパティ] および [サポートされる CI タイプ] タブの入力が必要です。 ▶ CmdbRmiAdapter：フェデレート・レプリケーションとフェデレート・クエリの両方をサポートします。したがって，[プロパティ]，[サポートされる CI タイプ]，および [サポートされるクエリ] タブの入力が必要です（標準設定では，このアダプタはレプリケーションのみをサポートします）。 ▶ CmdbSoapAdapter：フェデレート・レプリケーションのみをサポートし，ターゲットとして使用できます。したがって，[プロパティ] タブの入力が必要です。 ▶ ServiceDeskAdapter：フェデレート TQL クエリのみをサポートします。したがって，[プロパティ] および [サポートされる CI タイプ] タブの入力が必要です。 ▶ GenericDBAdapter
<p>ほかのタスク</p>	<p>112 ページ「フェデレート・データを使った作業 – ワークフロー」</p>
<p>参考リンク</p>	<ul style="list-style-type: none"> ▶ 109 ページ「複数のデータ・ストアからのデータの取得」 ▶ 111 ページ「外部データ・ストアからの属性の取得」 ▶ 108 ページ「アダプタ」

[プロパティ] タブ

説明	アダプタのタイプを選択できます。また、定義する外部データ・ストアの接続情報を定義できます。 利用方法 : [管理] > [設定] > [フェデレート CMDB] > [データストア] タブを選択します。[新規データストア] ボタンをクリックします。
重要情報	有効なフィールドはすべて必須フィールドです。
参考リンク	108 ページ「アダプタ」

含まれている要素は次のとおりです（ラベルのない GUI 要素は山括弧で囲んで示します）。



GUI 要素	説明
[アダプタ]	外部データ・ストアから外部データを取得するために、アダプタを選択します。
[テスト接続] ボタン	入力した有効な情報をテストします。
[顧客 ID]	HP Universal CMDB の場合は、「1」を入力します。 HP Software-as-a-Service の場合は、カスタマ ID 番号を入力します。
[ホスト]	アダプタが接続するマシン名を入力します。
[名前]	データ・ストアを識別するための名前を入力します。名前は HP Universal CMDB のバージョンで一意である必要があります。

GUI 要素	説明
<p>[次へ] ボタン</p>	<p>アダプタの定義を続けます。</p> <p>注：接続をテストしていない場合，[次へ] をクリックすると自動的にテストされます。</p> <p>アダプタがソースとしてフェデレート・クエリおよびフェデレート・レプリケーションをサポートしない場合（たとえば SOAP アダプタのように，アダプタはターゲットとしてレプリケーションのみをサポート），次のメッセージが表示されます。</p>  <p>[OK] をクリックし，次に [完了] をクリックします。</p>
<p>[パスワード]</p>	<p>接続する外部データ・ストアへのアクセスに必要なパスワードを入力します。システムを暗号化されたパスワードを受け入れるように設定した場合，このフィールドに入力したパスワードは暗号化されてデータベースに保存されます。</p> <p>暗号化されたパスワードの詳細については，114 ページ「フェデレート・アダプタの暗号化パスワードの変更」を参照してください。</p>
<p>[ポート]</p>	<p>外部データ・ストアへのアクセスに使用するポートを入力します（必要な場合）。</p>
<p>[URL]</p>	<p>外部データ・ストアに接続するための特定の URL を定義します。</p>
<p>[ユーザ]</p>	<p>接続する外部データ・ストアへのアクセスに必要なユーザ名を入力します。</p>

[サポートされる CI タイプ] タブ

説明	<p>データ・ストアがサポートする CIT または属性を選択できません。たとえば、フェデレート TQL (FTQL) に特定の CIT を表すノードが含まれている場合、CIT インスタスはこの外部データ・ストアに受け入れられます。</p> <p>利用方法 : [管理] > [設定] > [フェデレート CMDB] > [データ ストア] タブを選択します。[サポートされる CI タイプ] タブをクリックします。</p>
重要情報	このページは、選択したアダプタがフェデレート・クエリをサポートする場合に表示されます。
ほかのタスク	112 ページ「フェデレート・データを使った作業 - ワークフロー」
参考リンク	111 ページ「外部データ・ストアからの属性の取得」

含まれている要素は次のとおりです (ラベルのない GUI 要素は山括弧で囲んで示します)。


GUI 要素	説明
	インスタンスのフェデレーションを示します。つまり、CIT はフェデレートしています。
	属性のフェデレーションを示します。つまり、CIT の属性のレベルでフェデレートが行われています。
< CIT のリスト >	<p>このリストには、アダプタがサポートするすべての CIT が含まれます。</p> <p>FTQL でクエリする場合、ここで選択した CIT は外部データ・ストアからデータを取得するように設定されます。</p> <p>データ・ストアがサポートする CIT を選択します。</p>

GUI 要素	説明
[CI タイプ取得モード]	<p>▶ [選択した CI タイプの CI を取得]：その属性を含むすべての CI データは、データ ストアから取得されます。 CI タイプの CI を UCMDB から取得します。</p> <p>▶ [選択した属性の取得]：選択した属性がデータ・ストアから取得されます。CI がすでに UCMDB に存在している必要があります。 属性を UCMDB から取得します。</p> <p>注：</p> <p>▶ データ・ストア定義に含まれるすべての CIT（およびその子 CIT）は、同じ取得モードを使用する必要があります。</p> <p>▶ 同じデータ・ストアの CIT と属性の両方を選択することはできません。</p>
[CI タイプ]	<p>CIT の階層です。</p> <p>同じデータ・ストア・タイプの新しいデータ・ストアを定義するには、CIT を右クリックします。</p>
[属性の選択]	<p>フェデレーションに含める外部 CIT の属性を定義できます。</p> <p>▶ [CI タイプ取得モード] 表示枠で、[選択した属性の取得] を選択します。</p> <p>▶ [属性の選択] リストで、フェデレーションに含める属性を選択します。</p> <p>▶ 変更を保存します。</p> <p>注：属性は CIT マネージャで定義されます。詳細については、『モデル管理』の「[属性の追加 / 属性の編集] ダイアログ・ボックス」を参照してください。</p>

[サポートされるクエリ] タブ

説明	<p>アダプタがサポートするクエリのリストが表示されます。データのレプリケーション向けにこのデータ・ストアがサポートするクエリを選択できます。</p> <p>利用方法 : [管理] > [設定] > [フェデレート CMDB] > [データ ストア] タブを選択します。[サポートされるクエリ] タブをクリックします。</p>
重要情報	<p>このページは、フェデレート・レプリケーションをサポートし、ソースとして使用できるアダプタを選択した場合に表示されます。</p>
ほかのタスク	<p>112 ページ「フェデレート・データを使った作業 - ワークフロー」</p>

含まれている要素は次のとおりです（ラベルのない GUI 要素は山括弧で囲んで示します）。

GUI 要素	説明
	<ul style="list-style-type: none"> ▶ [すべて選択] または [すべて選択解除] : クリックするとリストのすべてのクエリを選択し、再度クリックするとすべてのクエリがクリアされます。 ▶ [Switch Selection] : クリックすると、選択されていないすべてのクエリが選択されるか、選択したすべてのクエリがクリアされます。 ▶ [選択したクエリを表示] : 選択したすべてのクエリのリストが表示されます。
<p><クエリ名のリスト></p>	<p>このリストには、アダプタがサポートするすべてのクエリが含まれます。ここで選択したクエリは、レプリケーション・ジョブの構成中に表示されます。</p> <p>設定したデータ・ストアに含めるクエリを選択します。</p> <p>[可視] : 現在のデータ・ストアの [レプリケーション ジョブ] ダイアログ・ボックスにクエリが表示されます。</p>

[レプリケーション ジョブ] ダイアログ・ボックス

説明	2つのデータ・ストア間でデータを複製できます。 利用方法 : [管理] > [設定] > [フェデレート CMDB] > [レプリケーション ジョブ] にアクセスします。[新規レプリケーション ジョブ] ボタンをクリックします。
重要情報	<ul style="list-style-type: none"> ▶ レプリケーションを成功させるために、レプリケーションに CMDB アダプタを使用する場合 (SOAP または RMI), ターゲット (データがインポートされ、フェデレートされるマシン) の CIT クラス・モデルはソース (データのエクスポート元のマシン) のクラス・モデルと同一でなければなりません。 ▶ レプリケーション・ジョブ中に特定の CI を TQL からのみ追加するには、CI に関する必要な条件を設定する必要があります。次に、CI にリンクされている TQL 関係に同じ条件を設定する必要があります。
ほかのタスク	112 ページ「フェデレート・データを使った作業 - ワークフロー」

含まれている要素は次のとおりです (ラベルのない GUI 要素は山括弧で囲んで示します)。

GUI 要素	説明
[名前]	レプリケーション・ジョブを識別するための名前を入力します。
[レプリケーション ジョブ クエリ]	<ul style="list-style-type: none"> ▶ [Active] : 選択すると、レプリケーション・ジョブでこのクエリを使用します。 ▶ [名前] : クエリの名前です。 ▶ [説明] : クエリの説明です。 ▶ [ターゲットでの削除を許可] : CI および関係がソース・クエリ結果で削除された場合、ターゲット・マシンでのこれらの削除を許可します。

GUI 要素	説明
[ソース データストア]	<p>ターゲット・マシンに引き渡されて複製されるデータのデータ・ストアです。</p> <p>ここでは、レプリケーションをサポートし、ソースとして使用できるアダプタを持つデータ・ストアのみが表示されます。</p> <p>データ・ストアに関する情報を表示するには、[詳細] をクリックします。</p>
[ターゲットデータストア]	<p>ソース・マシンから取り込まれたデータが複製されるデータ・ストアです。</p> <p>ここでは、レプリケーションをサポートし、ターゲットとして使用できるアダプタを持つデータ・ストアのみが表示されます。</p>

[レプリケーション ジョブ統計] ウィンドウ

説明	<p>選択したレプリケーション・ジョブの統計を表示して、レプリケーションが成功したかどうかを確認できます。</p> <p>利用方法 : [管理] > [設定] > [フェデレート CMDB] > [レプリケーション ジョブ] タブにアクセスします。レプリケーション・ジョブを選択し、ツールバーの [統計] アイコンをクリックします。</p>
重要情報	<p>個別のレプリケーション・ジョブのエラー・メッセージを表示するには、行をクリックします。</p>

含まれている要素は次のとおりです (ラベルのない GUI 要素は山括弧で囲んで示します)。






GUI 要素	説明
[アドホック]	<p>[Yes] : スケジューラ、または [レプリケーション ジョブ] タブの [アドホック] アイコンのクリックによってレプリケーション・ジョブが実行されました。</p>
[完全レプリケーション]	<p>[Yes] : 最後に実行したレプリケーション・ジョブを考慮に入れず、ソース内のすべての該当データをターゲットに取得します。</p>




GUI 要素	説明
[ジョブ名]	レプリケーション・ジョブに付けた名前です。
[クエリ当たりのレプリケーションジョブ統計]	<ul style="list-style-type: none"> ▶ [クエリ名] : レプリケーションで使用されるクエリの名前です。 ▶ [Status] : SUCCEEDED または FAILED です。ステータスが「FAILED」の場合、クエリをクリックして [レプリケーションジョブエラー] ダイアログ・ボックスを表示します。失敗の理由が表示されます。詳細については、次のフォルダにある <code>fcmdb.synchronizer.log</code> ファイルを参照してください。 < HP Universal CMDB のルート・ディレクトリ > <code>¥UCMDBServer¥j2¥fcmdb</code> ▶ [Replication Start Time] : レプリケーション・ジョブが現在のクエリのレプリケーションの実行を開始した時刻です。 ▶ [Replication Stop Time] : レプリケーション・ジョブが現在のクエリのレプリケーションの実行を停止した時刻です。 ▶ [更新済み] : 最後のレプリケーション・ジョブ中にターゲット・データ・ストアで更新された CI および関係の数です。 ▶ [追加済み] : 最後のレプリケーション・ジョブ中にターゲット・データ・ストアで追加された CI および関係の数です。 ▶ [削除済み] : 最後のレプリケーション・ジョブ中にターゲット・データ・ストアで削除された CI および関係の数です。注 : CI および関係は、レプリケーション・ジョブ定義で [ターゲットでの削除を許可] が選択されている場合にのみターゲットから削除されます。 ▶ [Has Error] : [Yes] : [レプリケーションジョブエラー] ダイアログ・ボックスが開きます。失敗の理由が表示されます。
[ソース データストア]	ソース・データ・ストアの ID です。
[Status]	SUCCEEDED または FAILED です。
[ターゲット データストア]	ターゲット・データ・ストアの ID です。

[レプリケーション ジョブ] タブ

説明	<p>ソース・データ・ストア、ターゲット・データ・ストア、および結果を複製するクエリを含むレプリケーション・ジョブを定義できます。</p> <p>利用方法 : [管理] > [設定] > [フェデレート CMDB] > [レプリケーション ジョブ] タブにアクセスします。</p>
重要情報	<p>スケジュールをセットアップしてレプリケーション・ジョブを実行できます。詳細については、『モデル管理』の「スケジューラのアクション」を参照してください。</p>

含まれている要素は次のとおりです（ラベルのない GUI 要素は山括弧で囲んで示します）。

GUI 要素	説明
 [新規レプリケーションジョブ]	<p>レプリケーション・ジョブを追加します。詳細については、124 ページ「[レプリケーションジョブ] ダイアログ・ボックス」を参照してください。</p>
 [選択した項目の削除]	<p>レプリケーション・ジョブを削除します。</p>
 [新規データストア]	<p>[新規データストア] ダイアログ・ボックスが開きます。詳細については、118 ページ「[新規データストア] ウィザード」を参照してください。</p>
 [保存]	<p>変更内容を保存します。</p>
 [更新]	<p>ページを更新します。</p>

GUI 要素	説明
	<p>レプリケーションが正常に行われたことを次のようにテストします。</p> <ul style="list-style-type: none"> ▶ 【一時的な差分レプリケーション】：ソース・データ・ストアの結果と、最後に実行したレプリケーション・ジョブの結果が比較されます。変更分だけがターゲット・データ・ストアに送信されます。 ▶ 【一時的なフルレプリケーション】：最後に実行したレプリケーション・ジョブを考慮に入れず、ソース内のすべての該当データをターゲットに取り込みます。 <p>注：この機能は、レプリケーション・ジョブの開発時にテストする場合に便利です。実運用では、レプリケーション・ジョブの実行にはスケジューラを使用してください。詳細については、129 ページ「[スケジュール] タブ」を参照してください。</p>
 【統計情報】	<p>選択したレプリケーション・ジョブに関するレプリケーション・ジョブ統計が表示されます。詳細については、125 ページ「[レプリケーションジョブ統計] ウィンドウ」を参照してください。</p>
	<p>タスクのリストをフィルタ処理するには、リストから選択し、[by] ボックスに最初の文字を入力します。</p>
<p><右クリック・メニュー></p>	<p>詳細については、この表のボタンの説明を参照してください。</p> <p>【重複レプリケーション ジョブ】：必要に応じて重複ジョブの名前を入力し、変更を加えます。ジョブを保存します。</p>
<p>【タスク テーブル】</p>	<p>【名前】：レプリケーション・ジョブに付けた名前です。</p> <p>【ソース データ ストア】：ソース・データ・ストアの ID です。</p> <p>【ターゲット データ ストア】：ターゲット・データ・ストアの ID です。</p> <p>【クエリ】：レプリケーション・ジョブでレプリケーションに使用するクエリです。</p>




[プロパティ] タブ

説明	レプリケーション・ジョブを編集できます。 利用方法 : [管理] > [設定] > [フェデレート CMDB] > [レプリケーション ジョブ] タブにアクセスします。[プロパティ] タブをクリックします。
重要情報	詳細については、124 ページ「[レプリケーション ジョブ] ダイアログ・ボックス」を参照してください。

[スケジュール] タブ

説明	スケジュールを設定してレプリケーション・ジョブを実行できます。 利用方法 : [管理] > [設定] > [フェデレート CMDB] > [レプリケーション ジョブ] タブにアクセスします。[スケジュール] タブをクリックします。
参考リンク	『モデル管理』の第15章「スケジューラ」

含まれている要素は次のとおりです（ラベルのない GUI 要素は山括弧で囲んで示します）。

GUI 要素	説明
	スケジュールを追加します。詳細については、『モデル管理』の「[ジョブ定義] ダイアログ・ボックス」を参照してください。レプリケーション・ジョブ・アクションの設定の詳細については、『モデル管理』の「[アクション定義] ダイアログ・ボックス」を参照してください。
	既存のスケジュールを編集します。詳細については、『モデル管理』の「[ジョブ定義] ダイアログ・ボックス」を参照してください。
	スケジュールを削除します。
[Active]	定義された時間にスケジュールを実行します。
[ジョブ定義]	ジョブの説明です。
[最終実行時間]	前回スケジュールが実行された時刻です。

GUI 要素	説明
[名前]	スケジュールの名前です。
[次の実行時間]	スケジュールが次に実行される日時です。
[スケジュール]	スケジュールの説明です。

第 6 章

汎用データベース・アダプタ

本章では、汎用データベース・アダプタの機能、使用法、および HP Universal CMDB へのデプロイメントについて説明します。

本章の内容

概念

- ▶ データベース・アダプタの概要 (132 ページ)
- ▶ サポートしていない TQL クエリ (133 ページ)
- ▶ 調整 (134 ページ)
- ▶ JPA プロバイダとしての Hibernate (136 ページ)

タスク

- ▶ データベース・アダプタのデプロイ - 最小メソッド (138 ページ)
- ▶ データベース・アダプタのデプロイ - 詳細メソッド (144 ページ)

参照先

- ▶ フェデレート・データベース構成ファイル (168 ページ)
- ▶ adapter.conf ファイル (169 ページ)
- ▶ simplifiedConfiguration.xml ファイル (170 ページ)
- ▶ orm.xml ファイル (177 ページ)
- ▶ reconciliation_rules.txt ファイル (182 ページ)
- ▶ transformations.txt ファイル (182 ページ)
- ▶ persistence.xml ファイル (183 ページ)
- ▶ discriminator.properties ファイル (185 ページ)
- ▶ replication_config.txt ファイル (186 ページ)

- ▶ fixed_values.txt ファイル (186 ページ)
- ▶ 用意済みのコンバータ (186 ページ)
- ▶ プラグイン (190 ページ)
- ▶ 設定例 (190 ページ)
- ▶ フェデレート・データベース・ログ・ファイル (203 ページ)
- ▶ 外部参照 (205 ページ)
- ▶ **トラブルシューティングと制限事項 (206 ページ)**

データベース・アダプタの概要

このアダプタの目的は、どんな種類のリレーショナル・データベース管理システム (RDBMS) とでも統合できるようにして、データベースに対するフェデレート TQL クエリとレプリケーション・ジョブを実行することです。

このバージョンのデータベース・アダプタ実装は、JPA (Java Persistence API) と、パーシステンス・プロバイダとしての Hibernate ORM ライブラリに基づいています。

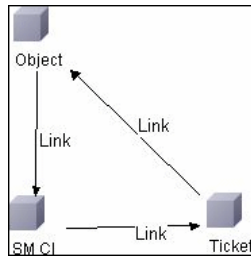
汎用データベース・アダプタでサポートしているデータベースは、Oracle, Microsoft SQL Server, および MySQL です。

サポートしていない TQL クエリ

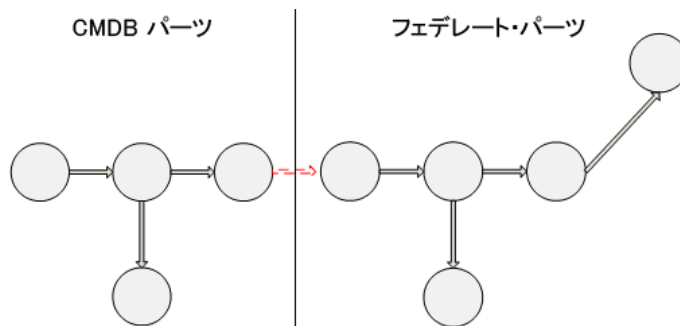
フェデレート CMDB には次の制限事項があります。

- ▶ サブグラフがサポートされていない
- ▶ 複合関係がサポートされていない
- ▶ サイクルまたはサイクル・パーツがサポートされていない

次の TQL はサイクルの一例です。



- ▶ 関数レイアウトがサポートされていない。
- ▶ 0.0 カーディナリティがサポートされていない。
- ▶ Joinf 関係がサポートされていない。
- ▶ 修飾子条件がサポートされていない。
- ▶ 2つの CI を接続するには、テーブルまたは外部キー形式の関係が外部データベース・ソースに存在する必要がある。



調整

調整はアダプタ側で TQL 計算の一部として実行されます。調整を行うには、CMDB 側を `multinode` というフェデレート・エンティティにマップします。

マッピング : UCMDB の各属性がデータ・ソースのカラムにマップされます。

マッピングは直接実行されますが、マッピング・データの変換関数もサポートされています。新しい関数は Java コードで追加できます (`lowercase`, `uppercase` など)。これらの関数の目的は、値を変換できるようにすることです (ある形式で CMDB に保管された値と別の形式でフェデレート・データベースに保管された値)。

注 :

- ▶ UCMDB と外部データベース・ソースに接続するには、データベースに適切な関連付けがなければなりません。詳細については、145 ページ「前提条件」を参照してください。
- ▶ CMDB id による調整もサポートされています。

調整ルールは OR および AND 条件の形を取ります。 これらのルールはさまざまなノードで定義できます (たとえば、ホストは `host_name from host AND/OR ip_address from ip` で識別されます)。

次のオプションで一致するものが見つかります。

- ▶ **順序一致** : 調整式は左から右へ読みます。2つの OR 副次式が値を持ち、それらが等しい場合は、同じであると見なされます。2つの OR 副次式が値を持ち、それらが等しくない場合は、同じではないと見なされます。その他の場合は決まりがなく、次の OR 副次式で相等性がテストされます。

host_name OR ip_address : UCMDB とデータ・ソースに `host_name` が含まれていて、それらが同じであれば、ホストが同じであると見なされます。両方に `host_name` があっても同じでなければ、ホストは同じでないと見なされ、`ip_address` はテストされません。UCMDB またはデータ・ソースに `host_name` がなければ、`ip_address` がチェックされます。

- ▶ **正規表現一致** : OR 副次式のいずれかに同じものがあれば、UCMDB とデータ・ソースは同じであると見なされます。

host_name OR ip_address : `host_name` が一致しない場合は、`ip_address` で相等性がチェックされます。

複雑な調整の場合は、調整エンティティがクラス・モデルで関係のある複数の CIT (`host` など) としてモデル化され、スーパーセット・ノードのマッピングに、モデル化されたすべての CIT の関連属性がすべて含まれます。

注 : 結果として、データ・ソースの調整属性はすべて、同じプライマリ・キーを共有するテーブルになければならないという制限があります。

別の制限によって、調整 TQL には2つのノードしか持てません。たとえば、`host > ticket TQL` は、UCMDB にホストがあり、データ・ソースにチケットがあります。

結果を調整するには、ホストから `host_name` を取得するか、IP アドレスから `ip_address` を取得する必要があります。このフェデレート `multinode` ホストからデータベース・ホスト・テーブルへ、そしてチケットからデータベース・チケットへ新しいマッピングが作成されます。この場合は、マルチノードが調整に必要なすべての属性のスーパーセットになります (`host_name + ip_address`)。

CMDB の `host_name` が `*.m.com` の形式である場合は、コンバータを使用して CMDB からフェデレート・データベースに、また逆の場合も同様にこれらの値を変換できます。

データベース・チケット・テーブルの `host_id` カラムを使用して、2つのエンティティを接続します (関連付けの定義もホスト・テーブルで作成できます)。

DB ホスト	
PK	<code>host_id</code>
	<code>ex_host_name</code> <code>ex_ip_address</code>

DB チケット	
PK	<code>ticket_id</code>
	<code>host_id</code>

注 : 2つのテーブルは CMDB データベースではなく、フェデレート RDBMS ソースの一部でなければなりません。

🍷 JPA プロバイダとしての Hibernate

Hibernate はオブジェクト関連 (OR) マッピング・ツールで、数種類のリレーショナル・データベース (Oracle や Microsoft SQL Server など) 上のテーブルに Java クラスをマッピングできます。詳細については、206 ページ「機能上の制限事項」を参照してください。

基本マッピングでは、各 Java クラスが単一テーブルにマップされます。詳細なマッピングでは、継承マッピングができます (CMDB データベースで行うことができます)。

サポートされているほかの機能としては、複数テーブルへのクラスのマッピング、コレクションのサポート、1対1、1対多、および多対1タイプの関連付けなどがあります。詳細については、137 ページ「関連付け」を参照してください。

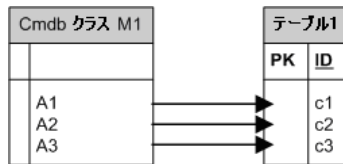
そのために、Java クラスを作成する必要はありません。CMDB クラス・モデル CIT からデータベース・テーブルへのマッピングが定義されます。

オブジェクト関連マッピングの例

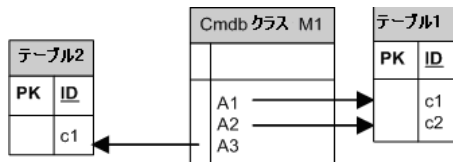
次の例で、オブジェクト関連マッピングについて説明します。

1つのデータベース・テーブルにマップされた1つのCMDBクラスの例

クラス M1 と属性 A1, A2, および A3 がテーブル1のカラム c1, c2, および c3 にマップされています。つまり、どの M1 インスタンスも、テーブル1に一致する行があります。

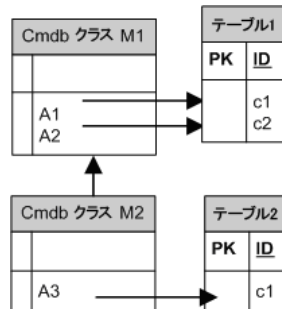


2つのデータベース・テーブルにマップされた1つのCMDBクラスの例



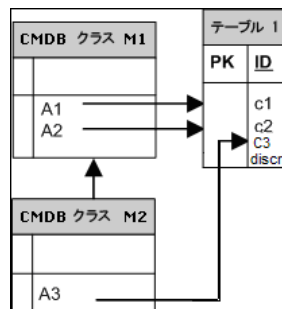
継承の例

このケースは CMDB で使用され、各クラスにそれぞれのデータベース・テーブルがあります。



識別子による単一テーブル継承の例

クラスの階層全体が単一データベース・テーブルにマップされ、そのカラムはマップされたクラスの全属性のスーパーセットで構成されます。このテーブルには追加カラム (Discriminator) も含まれて、その値は該当エントリにマップされる特定のクラスを示します。



関連付け

関連付けには、1対多、多対1、および多対多の3つのタイプがあります。異なるデータベース・オブジェクトを接続するには、外部キー・カラム (1対多の場合) またはマッピング・テーブル (多対多の場合) を使って、これらの関連付けのいずれかを定義する必要があります。

ユーザビリティ

JPA スキーマは非常に広範囲に及ぶので、定義を簡単にするために簡素化された XML ファイルが用意されています。

この XML ファイルを使用する事例は次のとおりです。フェデレート・データは1つのフェデレート・クラスにモデル化されます。このクラスは非フェデレート CMDB クラスと多対1の関係になります。また、フェデレート・クラスと非フェデレート・クラス間には、考えられる関係タイプが1つしかありません。

データベース・アダプタのデプロイ — 最小メソッド

注：アダプタを初めて作成するときは、このメソッドを使ってください。このメソッドを実行した結果として自動的に生成される **orm.xml** ファイルは、後で高度なメソッドを操作するときに利用できる好例です。

次の手順では、UCMDB のクラス・モデルを RDBMS にマッピングするメソッドについて説明します。この最小メソッドを使用するのは、次の必要がある場合です。

- ▶ ホスト属性などの単一ノードをフェデレートする。
- ▶ 汎用データベース・アダプタの機能を示す。
このメソッドでは、次のものをサポートしています。
- ▶ 1ノード・フェデレーションのみをサポートする
- ▶ 多対1の関係のみをサポートする

このタスクには次の手順が含まれています。

- ▶ 139 ページ「前提条件」
- ▶ 139 ページ「データベース・アダプタ構成ファイルの抽出」
- ▶ 139 ページ「アダプタ・パッケージのデプロイ」
- ▶ 139 ページ「アダプタのデプロイ」
- ▶ 139 ページ「CI タイプの作成」
- ▶ 139 ページ「関係の作成」
- ▶ 140 ページ「adapter.conf ファイルの設定」
- ▶ 140 ページ「simplifiedConfiguration.xml ファイルの設定」
- ▶ 143 ページ「手順の続行」

1 前提条件

詳細については、144 ページ「データベース・アダプタのデプロイ - 詳細メソッド」の前提条件を参照してください。

2 データベース・アダプタ構成ファイルの抽出

詳細については、148 ページ「データベース・アダプタ構成ファイルの抽出」を参照してください。

3 アダプタ・パッケージのデプロイ

詳細については、151 ページ「アダプタ・パッケージのデプロイ」を参照してください。

4 アダプタのデプロイ

詳細については、152 ページ「アダプタのデプロイ」を参照してください。

5 CI タイプの作成

詳細については、153 ページ「CI タイプの作成」を参照してください。

6 関係の作成

詳細については、155 ページ「関係の作成」を参照してください。

7 adapter.conf ファイルの設定

このステップでは、データを自動的にフェデレートできるように、adapter.conf ファイルの設定を変更します。

- a 次のファイルをテキスト・エディタで開きます。
＜ HP Universal CMDB ルート・ディレクトリ＞ ¥j2f¥fcmdb¥Codebase¥
MyAdapter¥META-INF¥adapter.conf
- b 次の行を見つけます。
`use.simplified.xml.config=<true/false>`
- c `use.simplified.xml.config=true` に変更します。

8 simplifiedConfiguration.xml ファイルの設定

このステップでは、UCMDB の CIT と RDBMS テーブルのフィールド間でマッピングして、simplifiedConfiguration.xml ファイルを設定します。

- a 次のファイルをテキスト・エディタで開きます。
＜ HP Universal CMDB ルート・ディレクトリ＞ ¥j2f¥fcmdb¥Codebase¥
MyAdapter¥META-INF¥simplifiedConfiguration.xml

- b このファイルには、マップする各エンティティに使用するテンプレートが含まれています。

```

<cmdb-class cmdb-class-name="host" default-table-name="[table_name]">
  <primary-key column-name="[column_name]"/>
  <reconciliation-by-two-nodes connected-node-cmdb-class-name="ip" cmdb-
link-type="contained">
    <or is-ordered="true">
      <attribute cmdb-attribute-name="host_hostname" column-
name="[column_name]" ignore-case="true"/>
      <connected-node-attribute cmdb-attribute-name="ip_address"
column-name="[column_name]"/>
    </or>
  </reconciliation-by-two-nodes>
</cmdb-class>

<class cmdb-class-name="[cmdb_class_name]" default-table-
name="[default_table_name]" connected-cmdb-class-name="host" link-class-
name="container_f">
  <foreign-primary-key column-name="[column_name]" cmdb-class-primary-
key-column="[column_name]"/>
  <primary-key column-name="[column_name]"/>
  <attribute cmdb-attribute-name="[cmdb_attribute_name]" column-
name="[column_name]" from-cmdb-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.Generi
cEnumTransformer(generic-enum-transformer-example.xml)" to-cmdb-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.Generi
cEnumTransformer(generic-enum-transformer-example.xml)"/>
  <attribute cmdb-attribute-name="[cmdb_attribute_name]" column-
name="[column_name]"/>
  <attribute cmdb-attribute-name="[cmdb_attribute_name]" column-
name="[column_name]"/>
</class>

```

- c 属性を次のように変更します。

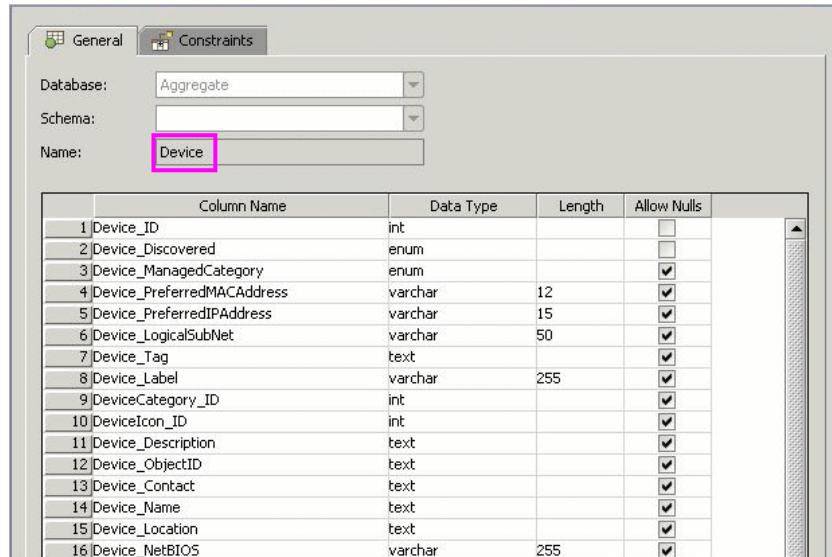
- ▶ Universal CMDB の CIT 名 (cmdb-class-name) と RDBMS の対応するテー
ブル名 (default-table-name) :

```
<cmdb-class cmdb-class-name="host" default-table-name="Device">
```

cmdb-class-name 属性はホスト CIT から取り出されます。



default-table-name 属性はデバイス・テーブルから取り出されます。



- ▶ RDBMS の一意の識別子 :

```
<primary-key column-name="Device_ID"/>
```

- ▶ 調整ルール (reconciliation-by-two-nodes) :

```
<reconciliation-by-two-nodes connected-node-cmdb-class-name="ip" cmdb-link-type="contained">
```

- ▶ Universal CMDB (cmdb-attribute-name) および RDBMS (column-name) の調整属性 :

```
<connected-node-attribute cmdb-attribute-name="ip_address" column-name="[column_name]"/>
```

- ▶ CIT の名前 (cmdb-class-name) と RDBMS で対応するテーブルの名前 (default-table-name)。また, CMDB 関係 (connected-cmdb-class-name) と CIT 関係 (link-class-name) :

```
<class cmdb-class-name="sw_sub_component" default-table-name="SWSubComponent" connected-cmdb-class-name="host" link-class-name="container_f">
```

- ▶ プライマリ・キーと外部キー :

```
<foreign-primary-key column-name="Device_ID" cmdb-class-primary-key-column="Device_ID"/>
```

- ▶ RDBMS の一意の識別子 :

```
<primary-key column-name="Device_ID"/>
```

- ▶ Universal CMDB 属性 (cmdb-attribute-name) と RDBMS のカラム名 (column-name) 間のマッピング :

```
<attribute cmdb-attribute-name="last_access_time" column-name="SWSubComponent_LastAccess TimeStamp"/>
```

d ファイルを保存します。

9 手順の続行

残りの手順については、次の項を参照してください。

- ▶ 162 ページ「アダプタのロード」
- ▶ 163 ページ「データ・ストアの作成」
- ▶ 163 ページ「ビューの作成」
- ▶ 165 ページ「結果の計算」
- ▶ 165 ページ「結果の表示」
- ▶ 167 ページ「レポートの表示」
- ▶ 167 ページ「ログ・ファイルの有効化」

データベース・アダプタのデプロイ — 詳細メソッド

次の手順では、UCMDB のクラス・モデルを RDBMS にマッピングする完全なメソッドについて説明します。

このタスクには次の手順が含まれています。

- ▶ 145 ページ「前提条件」
- ▶ 148 ページ「データベース・アダプタ構成ファイルの抽出」
- ▶ 151 ページ「アダプタ・パッケージのデプロイ」
- ▶ 152 ページ「アダプタのデプロイ」
- ▶ 153 ページ「CI タイプの作成」
- ▶ 155 ページ「関係の作成」
- ▶ 157 ページ「orm.xml ファイルの設定」
- ▶ 159 ページ「関係のマップ」
- ▶ 162 ページ「reconciliation_rules.txt ファイルの設定」
- ▶ 162 ページ「アダプタのロード」
- ▶ 163 ページ「データ・ストアの作成」
- ▶ 163 ページ「ビューの作成」
- ▶ 165 ページ「結果の計算」
- ▶ 165 ページ「結果の表示」
- ▶ 167 ページ「レポートの表示」
- ▶ 167 ページ「ログ・ファイルの有効化」

1 前提条件

データベースでデータベース・アダプタを使用できるか検証するには、次の点をチェックします。

- ▶ 調整クラスとその属性（**multinode** ともいう）がデータベースにあるかどうか。たとえば、調整をホスト名で実行する場合は、ホスト名の記入されたカラムが含まれているテーブルがあるか確認します。調整をホストの **cmdb_id** に従って実行する場合は、**CMDB** 内のホストの **CMDB ID** に一致する **CMDB ID** を持つカラムがあるか確認します。調整の詳細については、134 ページ「調整」を参照してください。

ID	HOST_NAME	IP_ADDRESS
31	BABA	16.59.33.60
33	ext3.devlab.ad	16.59.59.116
46	LABM1MAM15	16.59.58.188
72	cert-3-j2ee	16.59.57.100
102	labm1sun03.devlab.ad	16.59.58.45
114	LABM2PCOE73	16.59.66.79
116	CUT	16.59.41.214
117	labm1hp4.devlab.ad	16.59.60.182

- ▶ 2つの **CIT** を関係と関連付けるには、**CIT** テーブル間に相関データがなければなりません。相関関係は外部キー・カラムまたはマッピング・テーブルによるものになります。たとえば、ホストとチケットを関連付けるには、ホスト **ID** の含まれたチケット・テーブルのカラム、接続するチケット **ID** の含まれたホスト・テーブルのカラム、または **end1** がホスト **ID** で、**end2** がチケット **ID** のマッピング・テーブルがなければなりません。相関データの詳細については、136 ページ「JPA プロバイダとしての **Hibernate**」を参照してください。

次の表に、外部キーの HOST_ID カラムを示します。

HOST_ID	CARD_ID	CARD_TYPE	CARD_NAME
2015	1	シリアル・バス・ コントローラ	Intel® 82801EB USB ユニバーサル・ ホスト・コントローラ
3581	2	システム	Intel® 631xESB/6321ESB/3100 チップ セット LPC
3581	3	ディスプレイ	ATI ES1000
3581	4	基本システム・ ペリフェラル	HP ProLiant iLO 2 レガシー・サポー ト機能

- ▶ それぞれの CIT は 1 つ以上のテーブルにマップできます。1 つの CIT を複数のテーブルにマップするには、プライマリ・キーがほかのテーブルに存在するプライマリ・テーブルがあり、一意の値カラムがあるかチェックします。

たとえば、チケットは 2 つのテーブル、**ticket1** と **ticket2** にマップされます。最初のテーブルにはカラム **c1** と **c2** があり、もう 1 つのテーブルにはカラム **c3** と **c4** があります。これらを 1 つのテーブルと見なせるようにするには、両方に同じプライマリ・キーを設定する必要があります。あるいは、最初のテーブルのプライマリ・キーをもう 1 つのテーブルのカラムにします。

次の例では、テーブルが **CARD_ID** という同じプライマリ・キーを共有しています。

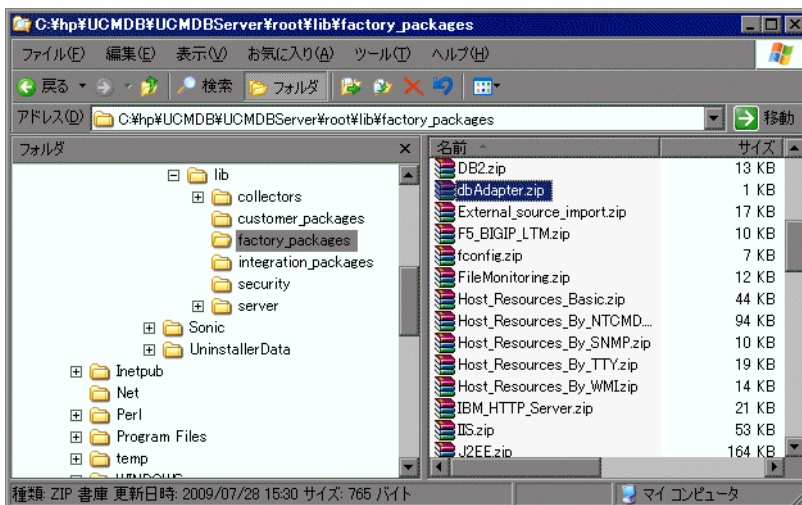
CARD_ID	CARD_TYPE	CARD_NAME
1	シリアル・バス・ コントローラ	Intel® 82801EB USB ユニバーサル・ホスト・ コントローラ
2	システム	Intel® 631xESB/6321ESB/3100 チップセット LPC
3	ディスプレイ	ATI ES1000
4	基本システム・ ペリフェラル	HP ProLiant iLO 2 レガシー・サポート機能

CARD_ID	CARD_VENDOR
1	Hewlett-Packard Company
2	(標準 USB ホスト・コントローラ)
3	Hewlett-Packard Company
4	(標準システム・デバイス)
5	Hewlett-Packard Company

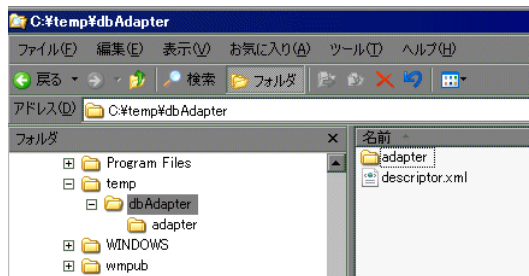
2 データベース・アダプタ構成ファイルの抽出

このステップでは、汎用データベース・アダプタ・パッケージを見つけて、そのコピーを作成します。

- a < HP Universal CMDB ルート・ディレクトリ > %UCMDBServer%root%lib%factory_packages で dbAdapter パッケージを見つめます。



- b このパッケージをローカルの一時ディレクトリに抽出します。



- c adapter\db_adapter.xml ファイルをテキスト・エディタで開きます。

このファイルには、データ・ストアに接続するのに必要なフィールドがあります。このファイルをテンプレート（推奨）として使用するか、新しいXML ファイルをゼロから作成できます。

d **adapter-id** 属性を見つけて、名前を置き換えます。

```
<adapter-config adapter-id="MyAdapter">
  <class-name>com.mercury.topaz.fcldb.adapters.dbAdapter.DBAdapter
</class-name>
  <adapter-capabilities>
    <support-federated-query>
      <supported-classes>
      </supported-classes>
      <topology>
        <pattern-topology>
        </pattern-topology>
      </topology>
    </support-federated-query>
  </adapter-capabilities>
  <fields-to-connect>
    <field>host</field>
    <field>customerId</field>
    <field>port</field>
    <field>url</field>
  </fields-to-connect>
  <default-mapping-engine>
com.mercury.topaz.fcldb.adapters.dbAdapter.reconciliation.mapping_engine.
DBMappingEngine</default-mapping-engine>
</adapter-config>
```

名前には、大文字 / 小文字や特殊文字などの制限はありません。ここで入力する名前は、HP Universal CMDB の [データストア] 表示枠のアダプタ・リストに表示されます。

プロパティ サポートされる CI タイプ サポートされるクエリ

プロパティ test1

アダプタ: MyAdapter

接続プロパティ

名前: test1

顧客 ID: 1

ホスト: localhost

ポート:

ユーザ:

パスワード:

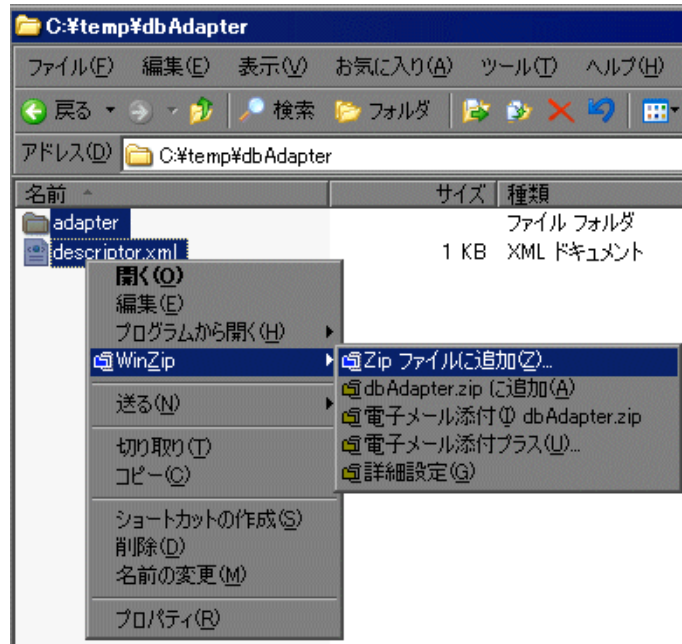
URL:

テスト接続

詳細については、116 ページ「[データストア] タブ」を参照してください。

重要：一貫性のために、また使いやすくするために、ファイルを保存するときやアダプタを定義するときに、この名前を使います。

- e *.zip ファイルを作成します。



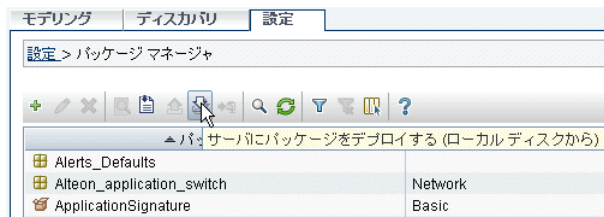
- f この zip ファイルには、手順 149 ページ、手順 d で述べているように、**attribute-id** 属性に付けたのと同じ名前を付けます。

注： descriptor xml ファイルは、すべてのパッケージに存在する標準設定ファイルです。

3 アダプタ・パッケージのデプロイ

このステップでは、パッケージ・マネージャにパッケージをデプロイします。

- a 前のステップで作成した新しいパッケージを次のフォルダに保存します。
< HP Universal CMDB ルート・ディレクトリ > ¥UCMDBServer¥root¥lib¥customer_packages
- b HP Universal CMDB で、パッケージ・マネージャにアクセスします。詳細については、「[パッケージマネージャ] ウィンドウ」を参照してください。
- c アダプタをデプロイします。[サーバにパッケージをデプロイする] アイコンをクリックします。



[追加] をクリックして、アダプタ・パッケージを参照します。[開く]、次に [OK] をクリックして、パッケージ・マネージャにパッケージを導入します。

- d XML ファイルの内容がパッケージ・マネージャに認識されているか確認します。リストでパッケージを選択して、[パッケージ リソースの表示] をクリックします。



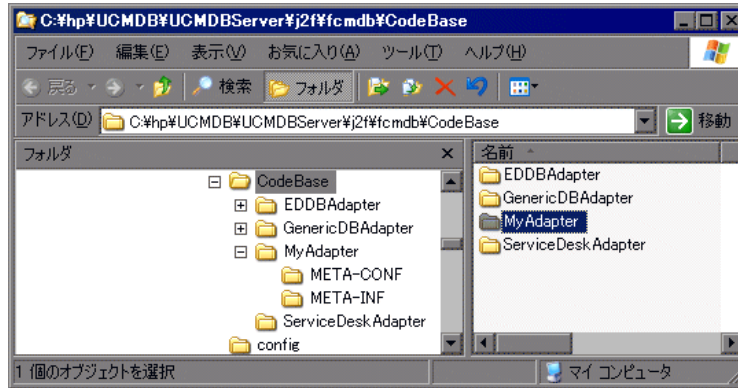
このパッケージにはアダプタ XML ファイルのみが含まれています。

4 アダプタのデプロイ

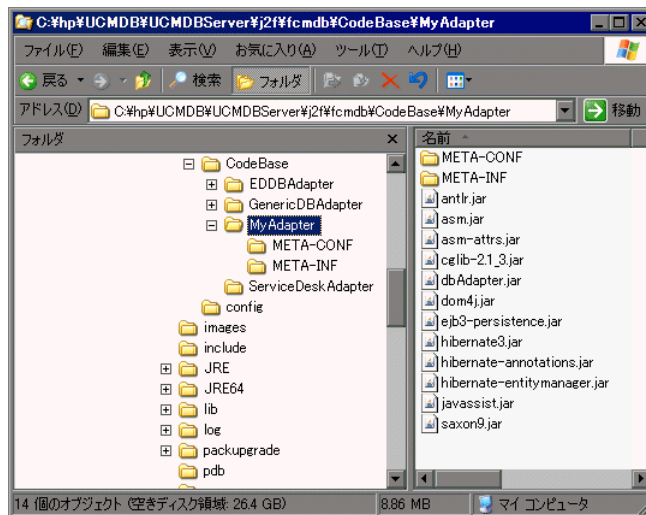
このステップでは、アダプタのロジックと定義間の接続を作成します。

a < HP Universal CMDB ルート・ディレクトリ > %j2f%cmdb%Codebase

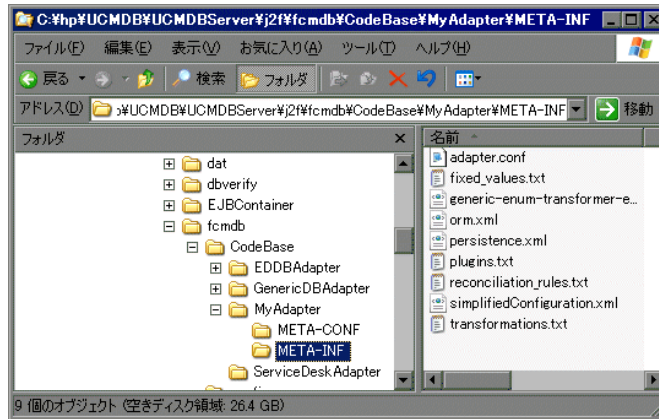
フォルダに汎用データベース・アダプタ・フォルダをコピーし、その名前をアダプタの名前（ステップ 149 ページ、手順 d で変更した adapter-id フィールドに付けたのと同じ名前）に変更します。



このフォルダには、アダプタ名、Universal CMDB のクエリとクラス、およびアダプタがサポートする RDBMS のフィールドなど、フェデレーション・ロジックを実行する jar ファイルが含まれています。



- b 新しいアダプタ・フォルダを開いて、META-INF フォルダにドリル・ダウンします。



このフォルダには、アダプタを実行するファイルが含まれています。
orm.xml ファイルによって、Universal CMDB クラス・モデルが RDBMS の
実際のカラムとテーブルにマップされます。

5 CI タイプの作成

このステップでは、RDBMS（外部データ・ソース）のデータにマップされる
フェデレート CIT を作成します。

- a HP Universal CMDB で、CI タイプ・マネージャにアクセスして、新規の CI
タイプを作成します。
- b CIT に必要な属性（最終アクセス日時、ベンダなど）を追加します。これら
は、アダプタによって外部データ・ソースから取得され、HP Universal
CMDB ビューに取り込まれる属性です。

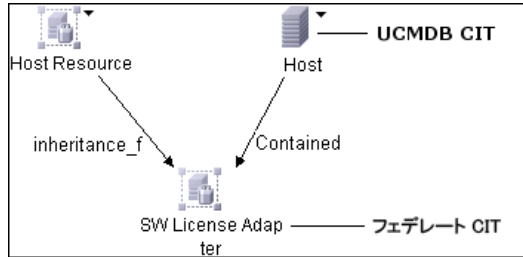
host_card CIT の作成例

```
<?xml version="1.0" encoding="UTF-8"?>
<Class class-name="host_card" display-name="Host Card" description="">
  <Class-Qualifiers/>
  <Class-Type>OBJECT</Class-Type>
  <Derived-From class-name="hostresource"/>
  <Attributes>
    <Attribute name="card_class" display-name="Card Class" description=""
type="string">
      <Attribute-Qualifiers/>
    </Attribute>
    <Attribute name="card_vendor" display-name="Card Vendor" description=""
type="string">
      <Attribute-Qualifiers/>
    </Attribute>
    <Attribute name="card_name" display-name="Card Name" description=""
type="string">
      <Attribute-Qualifiers/>
    </Attribute>
  </Attributes>
  <Attribute-Overrides>
    <Attribute-Override name="display_label" is-partially-override="true">
      <Attribute-Qualifiers>
        <Attribute-Qualifier name="CALCULATED_ATTRIBUTE">
          <Data-Items>
            <Data-Item name="FUNCTION"
type="string">card_name</Data-Item>
          </Data-Items>
        </Attribute-Qualifier>
      </Attribute-Qualifiers>
    </Attribute-Override>
  </Attribute-Overrides>
</Class>
```

6 関係の作成

このステップでは、HP Universal CMDB CIT と外部データ・ソースからフェデレートするデータを示す新しい CIT との関係を追加します。

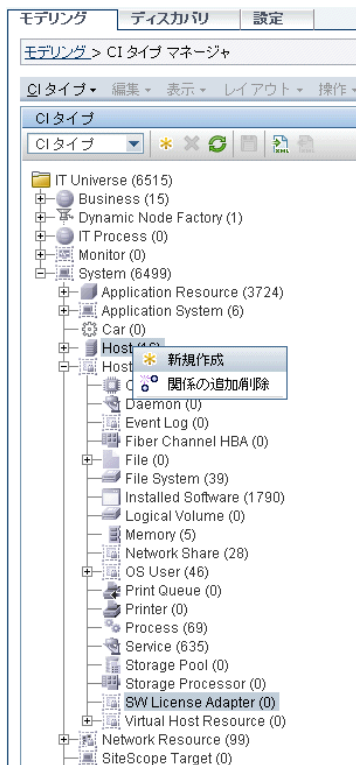
新しい CIT に適切で有効な関係を追加します。詳細については、「[関係の追加/削除] ダイアログ・ボックス」を参照してください。



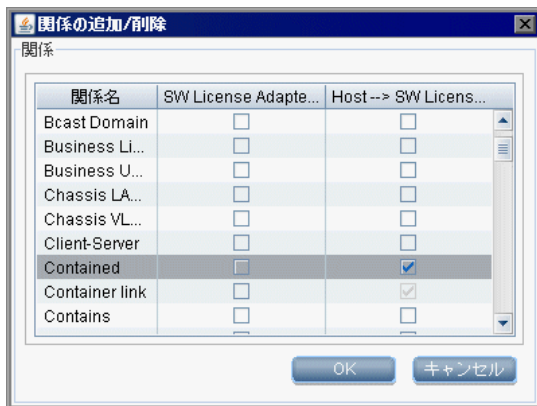
注：この段階では、データを取り込むメソッドを定義していないので、フェデレート・データをまだ表示できません。

Contained 関係の作成例

a CIT マネージャで、2つの CIT を選択します。



b 2つの CIT 間で **Contained** 関係を作成します。



host_card とホスト間の Contained 関係の例

```
<?xml version="1.0" encoding="UTF-8"?>
<Valid-Links>
  <Valid-Link>
    <Class-Ref class-name="contained"/>
    <End1 class-name="host"/>
    <End2 class-name="host_card"/>
  </Valid-Link>
</Valid-Links>
```

7 orm.xml ファイルの設定

このステップでは、Universal CMDB の CIT を RDBMS のテーブルにマップします。

ヒント：アダプタを初めて作成するときは、簡易メソッドの使用をお勧めします。このメソッドを実行した結果として自動的に生成される **orm.xml** ファイルは、高度なメソッドを操作するときに利用できる好例です。詳細については、138 ページ「データベース・アダプタのデプロイ - 最小メソッド」を参照してください。

- a XML またはテキスト・エディタで **orm.xml** を開きます。標準設定では、このファイルには、フェデレーションに必要な数の CIT とテーブルをマップするのに使用するテンプレートが含まれています。
- b マップされるデータ・エンティティに従って、ファイルに変更を加えます。詳細については、次の例を参照してください。

命名規則の詳細については、181 ページ「命名規則」を参照してください。

クラス・モデルと RDBMS 間のエンティティ・マッピングの例

注：設定する必要がない属性は、次の例から除外しています。

- ▶ Universal CMDB CIT の名前とクラス。

```
<entity name="host" class="generic_db_adapter.host">
```

- ▶ RDBMS にあるテーブルの名前。

```
<table name="Device"/>
```

- ▶ RDBMS テーブルにある一意の識別子のカラム名。

```
<column name="Device ID"/>
```

- ▶ Universal CMDB CIT にある属性の名前。

```
<basic name="host_hostname">
```

- ▶ 外部データ・ソースにあるテーブル・フィールドの名前。

```
<column name="Device_Name"/>
```

- ▶ 153 ページ「CI タイプの作成」で作成した新規 CIT の名前。

```
<entity name="MyAdapter" class="generic_db_adapter.MyAdapter">
```

- ▶ RDBMS で対応するテーブルの名前。

```
<table name="SW_License"/>
```

- ▶ 2つのプライマリ・キー・ノード。

```
<id class="generic_db_adapter.IDClass2PK_SW_Adapter">
```

- ▶ RDBMS の一意の識別子。

```
<id name="id1">
  <column updatable="false" insertable="false" name="Device_ID"/>
  <generated-value strategy="TABLE"/>
</id>
<id name="id2">
  <column updatable="false" insertable="false" name="Version_ID"/>
  <generated-value strategy="TABLE"/>
</id>
```

- ▶ Universal CMDB CIT にある属性名と、RDBMS で対応する属性の名前。

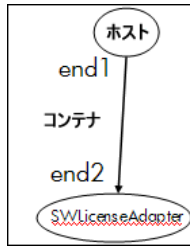
```
<basic name="license_required">
  <column updatable="false" insertable="false"
name="MyAdapter_LicenseRequired"/>
```

8 関係のマップ

このステップでは、Universal CMDB にある関係を RDBMS にある関係にマップします。関係の詳細については、161 ページ「クラス・モデルと RDBMS 間の関係マッピングの例」を参照してください。

マップされる関係に従って、`orm.xml` ファイルに変更を加えます。使用例に従って、RDBMS に存在するのと同じ関係を Universal CMDB で定義します。次の関係をマップできます。

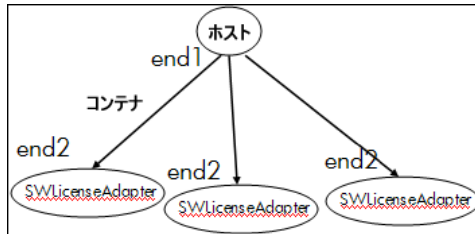
▶ 1対1:



このタイプの関係を記述するコードは次のようになります。

```
<one-to-one name="end1" target-entity="host">
  <join-column name="Device_ID" />
</one-to-one>
<one-to-one name="end2" target-entity="sw_sub_component">
  <join-column name="Device_ID" />
  <join-column name="Version_ID" />
</one-to-one>
```

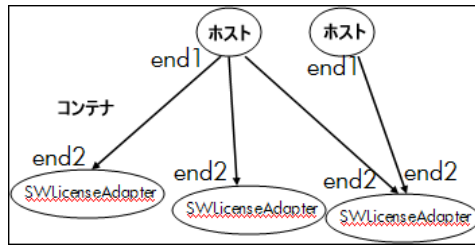
▶ 多対1:



このタイプの関係を記述するコードは次のようになります。

```
<many-to-one name="end1" target-entity="host">
  <join-column name="Device_ID" />
</many-to-one>
<one-to-one name="end2" target-entity="sw_sub_component">
  <join-column name="Device_ID" />
  <join-column name="Version_ID" />
</one-to-one>
```


- ▶ 多対多：



このタイプの関係を記述するコードは次のようになります。

```
<many-to-one name="end1" target-entity="host">
  <join-column name="Device_ID" />
</many-to-one>
<many-to-one name="end2" target-entity="sw_sub_component">
  <join-column name="Device_ID" />
  <join-column name="Version_ID" />
</many-to-one>
```

クラス・モデルと RDBMS 間の関係マッピングの例

- ▶ Universal CMDB 関係の名前とクラス：

```
<entity name="host_contained_MyAdapter"
class="generic_db_adapter.host_contained_MyAdapter">
```

- ▶ 関係が実行される RDBMS テーブルの名前。

```
<table name="MyAdapter"/>
```

- ▶ RDBMS にある一意の ID：

```
<id name="id1">
  <column updatable="false" insertable="false" name="Device_ID"/>
  <generated-value strategy="TABLE"/>
</id>
<id name="id2">
  <column updatable="false" insertable="false" name="Version_ID"/>
  <generated-value strategy="TABLE"/>
</id>
```

- ▶ 関係タイプと Universal CMDB CIT :

```
<many-to-one target-entity="host" name="end1">
```

- ▶ RDBMS にあるプライマリ・キーおよび外部キー・フィールド :

```
<join-column updatable="false" insertable="false" referenced-column-  
name="[column_name]" name="Device_ID"/>
```

9 reconciliation_rules.txt ファイルの設定

このステップでは、アダプタで Universal CMDB と RDBMS を調整するルールを定義します。

- テキスト・エディタで **META-INF¥reconciliation_rules.txt** を開きます。
- マッピングする CIT に従って、ファイルに変更を加えます。たとえば、ホスト CIT をマップするには、次の式を使います。

```
multinode[host] ordered expression[^host_hostname]
```

注 :

- ▶ データベースのデータで大文字 / 小文字を区別する場合は、制御文字 (^) を削除しないでください。
 - ▶ それぞれの左角括弧に対応する右角括弧があるかチェックしてください。
-

10 アダプタのロード

このステップでは、HP Universal CMDB マシンにアダプタをロードします。

注 : アダプタに変更を加えるたびに、JMX コンソールを使ってアダプタを再デプロイする必要があります。

- a HP Universal CMDB サーバ・マシンで Web ブラウザを起動し、次のアドレスを入力します。

```
http:// <マシン名または IP アドレス> . < domain_name > :8080/jmx-console
```

<マシン名または IP アドレス> には、HP Universal CMDB がインストールされているマシンを指定します。ユーザ名とパスワードでログインしなければなりません。

- b Topaz セクションの下で **service=Fcmdb Config Services** リンクをクリックします。
- c [JMX MBEAN View] ページで、**loadOrReloadCodeBaseForAdaptorId()** 演算を見つけます。
- d [customerID] フィールドに **1** を入力します。
- e [adaptorId] フィールドに **MyAdapter** を入力します（これはアダプタに付けた名前です）。
- f [Invoke] をクリックします。

11 データ・ストアの作成

このステップでは、フェデレーションが機能しているか、すなわち、接続が有効で、XML ファイルが有効であるかチェックします。ただし、このチェックでは、XML が RDBMS の正しいフィールドにマッピングされるかは確認されません。

- a HP Universal CMDB で、[フェデレート CMDB] ページにアクセスします（[設定] > [フェデレート CMDB]）。
- b データ・ストアを作成します。詳細については、116 ページ「[データストア] タブ」を参照してください。

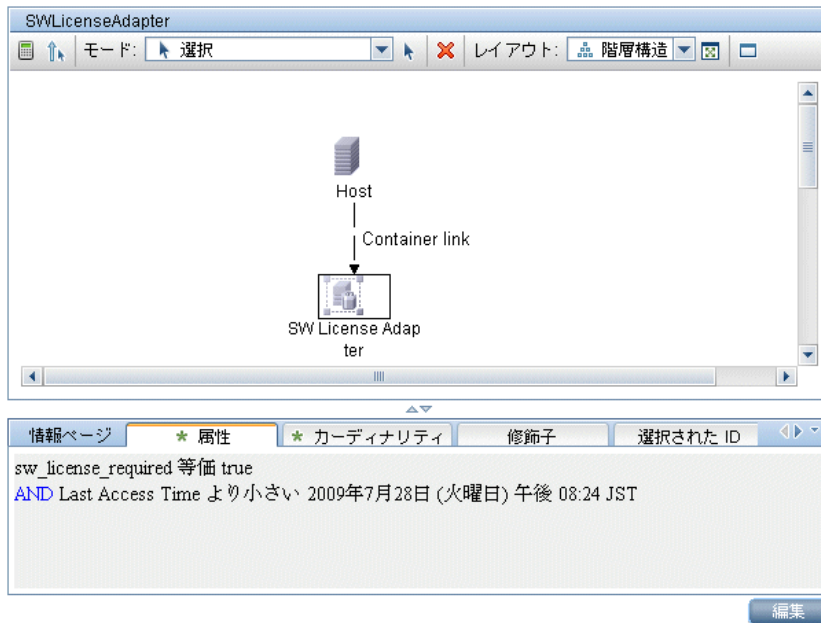
[データストア] ダイアログ・ボックスには、フェデレーションをサポートしている CIT がすべて表示されます。

12 ビューの作成

このステップでは、CIT のインスタンスを表示できるビューを作成します。

- a HP Universal CMDB で、ビュー・マネージャにアクセスします（[管理] > [モデリング] > [ビュー マネージャ]）。

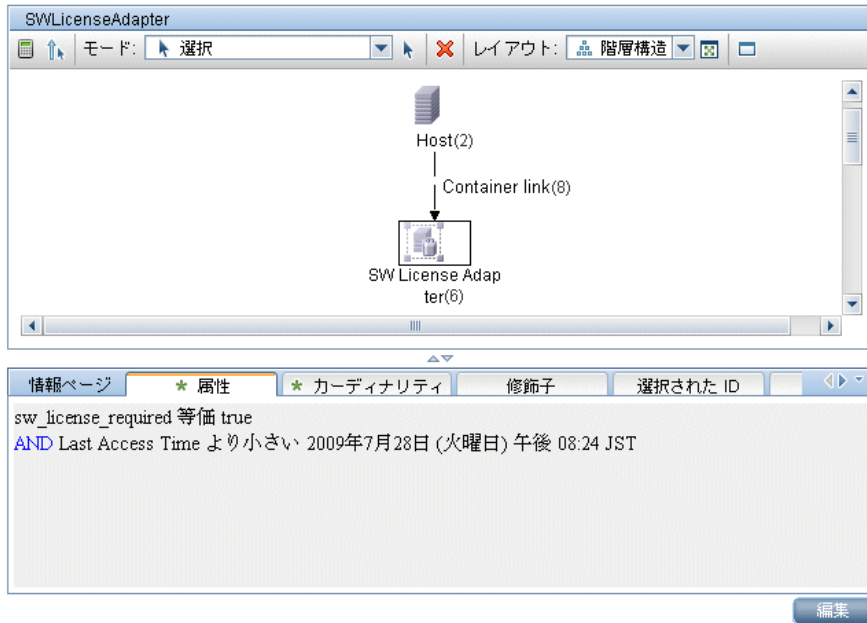
- b ビューを作成します。詳細については、『モデル管理』の「テンプレート・ベース・ビュー」を参照してください。
- c TQL に条件（最終アクセス日時が6か月を超えるなど）を追加できます。



13 結果の計算

このステップでは、結果をチェックします。

- a ビュー・マネージャから、結果を計算します。[TQL 結果数を計算する] ボタンをクリックします。



- b [プレビュー] ボタンをクリックすると、CIT の CI が表示されます。

14 結果の表示

このステップでは、結果を表示し、手順の問題をデバッグします。たとえば、ビューに何も表示されない場合は、`orm.xml` ファイルで定義をチェックし、関係属性を削除して、アダプタを再ロードします。

- a HP Universal CMDB で、トポロジ・ビューにアクセスします ([アプリケーション] > [トポロジ ビュー])。

b [プロパティ] タブには、フェデレーションの結果が表示されます。

The screenshot displays a software management application. The top window, titled "Topology View", shows a hierarchical diagram of components. At the top is a component labeled "acid", which is contained by a parent component. Below "acid", there are five "sw_sub_component" nodes, each connected to "acid" by a "Contained" relationship. The interface includes a toolbar with various icons and a status bar at the bottom of the topology view that reads "追加された CI △ 変更された CI 削除の候補".

Below the topology view is a "情報ページ" (Information Page) section with three tabs: "プロパティ" (Properties), "履歴" (History), and "ディスカバリ" (Discovery). The "プロパティ" tab is active, showing the following information:

- Name: sw_sub_component
- ID: sw_component%0A1%0AExternal-ID0%3DSTRING%3D75%0A
- CI Type: sw_sub_component

Below this information is a table of properties:

User Label	
root_iconproperties	
sw_last_access_date	2007-06-12 00:00:00.0
sw_license_required	True
sw_vendor	Corel Corporation

15 レポートの表示

このステップでは、トポロジ・レポートを表示します。詳細については、『モデル管理』の第28章「トポロジ・レポート」を参照してください。

sw_componet_report - sw_component_report - Required License software

ビュー固有レポートを設定(任意): [\[クリア\]](#) 履歴 DB に保存する

localization (Host)

Host Name: localization

Symantec AntiVirus (Installed Software) (ノード: localization (Host))

Last Access Time: 2009年8月25日 (火曜日) 午前 11:28 **Software Vendor:** Symantec Corporation
Name: Symantec AntiVirus

Windows Server 2003 セキュリティの更新: サポート技術情報 (KB) 926122 (Installed Software) (ノード: localization (Host))

Last Access Time: 2009年8月25日 (火曜日) 午前 11:28 **Software Vendor:** Microsoft Corporation
Name: Windows Server 2003 セキュリティの更新: サポート技術情報 (KB) 926122

SoundMAX (Installed Software) (ノード: localization (Host))

Last Access Time: 2009年8月25日 (火曜日) 午前 11:28 **Software Vendor:** Analog Devices
Name: SoundMAX

Windows Server 2003 セキュリティの更新: サポート技術情報 (KB) 932168 (Installed Software) (ノード: localization (Host))

Last Access Time: 2009年8月25日 (火曜日) 午前 11:28 **Software Vendor:** Microsoft Corporation

16 ログ・ファイルの有効化

計算フロー、アダプタ・ライフサイクルを理解したり、デバッグ情報を表示するには、ログ・ファイルを参照します。詳細については、203 ページ「フェデレート・データベース・ログ・ファイル」を参照してください。

フェデレート・データベース構成ファイル

本項で述べるファイルは、< HP Universal CMDB ルート・ディレクトリ > ¥ UCMDBServer¥j2f¥fcmdb¥CodeBase¥GenericDBAdapter¥META-INF フォルダにあります。

本項の内容

- ▶ 168 ページ「一般的な設定」
- ▶ 168 ページ「詳細な設定」
- ▶ 168 ページ「Hibernate 設定」
- ▶ 169 ページ「単純構成」

一般的な設定

- ▶ **adapter.conf** : アダプタ設定ファイルです。詳細については、169 ページ「adapter.conf ファイル」を参照してください。

詳細な設定

- ▶ **orm.xml** : CMDB CIT とデータベース・テーブル間にマップするオブジェクト関連マッピング・ファイル。詳細については、177 ページ「orm.xml ファイル」を参照してください。
- ▶ **reconciliation_rules.txt** : 調整ルールが含まれています。詳細については、182 ページ「reconciliation_rules.txt ファイル」を参照してください。
- ▶ **transformations.txt** : CMDB 値をデータベース値に、またその逆に変換するのに適用するコンバータを指定する変換ファイルです。詳細については、182 ページ「transformations.txt ファイル」を参照してください。

Hibernate 設定

- ▶ **persistence.xml** : 用意済みの Hibernate 設定をオーバーライドするのに使います。詳細については、183 ページ「persistence.xml ファイル」を参照してください。

単純構成

- ▶ **simplifiedConfiguration.xml** : orm.xml, transformations.txt, および reconciliation_rules.txt を機能の少ないものに置き換える設定ファイルです。詳細については、170 ページ「simplifiedConfiguration.xml ファイル」を参照してください。

adapter.conf ファイル

このファイルには次の設定が含まれています。

- ▶ **use.simplified.xml.config=false. true** : simplifiedConfiguration.xml を使用します。

注 : このファイルを使用することは、orm.xml, transformations.txt, および reconciliation_rules.txt を機能の少ないものに置き換えるということです。

- ▶ **dal.ids.chunk.size=300** : この値は変更しないでください。
- ▶ **dal.use.persistence.xml=false. true** : アダプタが persistence.xml から Hibernate 設定を読み込みます。

注 : Hibernate 設定を上書きするのはお勧めしません。

simplifiedConfiguration.xml ファイル

このファイルは、CMDB クラスをデータベース・テーブルに単純にマッピングするのに使用されます。このファイルを編集するテンプレートは、**< HP Universal CMDB ルート・ディレクトリ > ¥UCMDBServer¥2f¥fcmdb¥CodeBase¥GenericDBAdapter¥META-INF** フォルダにあります。

本項の内容

- ▶ 170 ページ「XSD ファイルの例」
- ▶ 173 ページ「テンプレート」
- ▶ 177 ページ「制限事項」

XSD ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel.3 U (http://www.xmlspy.com) by Nimrod (Mercury Interactive) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="generic-DB-adapter-config">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="CMDB-class" maxOccurs="unbounded"/>
        <xs:element ref="class" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="class">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="foreign-primary-key" maxOccurs="unbounded"/>
        <xs:element ref="primary-key" maxOccurs="unbounded"/>
        <xs:element ref="attribute" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="CMDB-class-name" type="xs:string" use="required"/>
      <xs:attribute name="default-table-name" type="xs:string" use="required"/>
      <xs:attribute name="connected-CMDB-class-name" type="xs:string" use="required"/>
      <xs:attribute name="link-class-name" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="reconciliation-by-single-node">
    <xs:complexType>
      <xs:choice>
        <xs:element ref="attribute"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```

<xs:element name="or">
  <xs:complexType>
    <xs:choice minOccurs="2" maxOccurs="unbounded">
      <xs:element name="and">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="attribute" minOccurs="2"
maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element ref="attribute"/>
    </xs:choice>
    <xs:attribute name="is-ordered" type="xs:boolean" use="optional"
default="false"/>
  </xs:complexType>
</xs:element>
<xs:element name="and">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="attribute" minOccurs="2" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
<xs:element name="primary-key">
  <xs:complexType>
    <xs:attribute name="column-name" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="attribute">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="attribute-type"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:complexType name="attribute-type">
  <xs:attribute name="table-name" type="xs:string" use="optional"/>
  <xs:attribute name="column-name" type="xs:string" use="required"/>
  <xs:attribute name="CMDB-attribute-name" type="xs:string" use="required"/>
  <xs:attribute name="from-CMDB-converter" type="xs:string" use="optional"/>
  <xs:attribute name="to-CMDB-converter" type="xs:string" use="optional"/>
  <xs:attribute name="ignore-case" type="xs:boolean" use="optional" default="false"/>

```

```

</xs:complexType>
<xs:complexType name="class-type"/>
<xs:element name="or">
  <xs:complexType>
    <xs:choice minOccurs="2" maxOccurs="unbounded">
      <xs:element ref="and"/>
      <xs:element ref="attribute"/>
      <xs:element ref="connected-node-attribute"/>
    </xs:choice>
    <xs:attribute name="is-ordered" type="xs:boolean" use="optional" default="false"/>
  </xs:complexType>
</xs:element>
<xs:element name="and">
  <xs:complexType>
    <xs:choice minOccurs="2" maxOccurs="unbounded">
      <xs:element ref="attribute"/>
      <xs:element ref="connected-node-attribute"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
<xs:element name="reconciliation-by-two-nodes">
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element ref="attribute"/>
        <xs:element ref="connected-node-attribute"/>
        <xs:element ref="or"/>
        <xs:element ref="and"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="connected-node-CMDB-class-name" type="xs:string" use="required"/>
    <xs:attribute name="CMDB-link-type" type="xs:string" use="required"/>
    <xs:attribute name="link-direction" use="optional" default="main-to-connected">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="main-to-connected"/>
          <xs:enumeration value="connected-to-main"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="connected-node-attribute" type="attribute-type"/>
<xs:element name="CMDB-class">
  <xs:complexType>
    <xs:sequence>

```

```

<xs:element ref="primary-key" maxOccurs="unbounded"/>
<xs:choice>
  <xs:element ref="reconciliation-by-single-node"/>
  <xs:element ref="reconciliation-by-two-nodes"/>
</xs:choice>
</xs:sequence>
<xs:attribute name="CMDB-class-name" type="xs:string" use="required"/>
<xs:attribute name="default-table-name" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="foreign-primary-key">
  <xs:complexType>
    <xs:attribute name="CMDB-class-primary-key-column" type="xs:string" use="required"/>
    <xs:attribute name="column-name" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
</xs:schema>

```

テンプレート

CMDB-class-name プロパティは multinode タイプ（TQL でフェデレート CIT が接続するノード）です。

```

<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../META-CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="host" default-table-name="[table_name]">
    <primary-key column-name="[column_name]"/>

```

reconciliation-by-two-nodes : 調整を行うには、1 つまたは 2 つのノードを使用します。この例では、調整で 2 つのノードを使用しています。

connected-node-CMDB-class-name : 調整 TQL で必要とされる第 2 クラス・タイプ。

CMDB-link-type : 調整 TQL で必要とされる関係タイプ。

link-direction : 調整 TQL における関係の方向（host から ip または ip から host） :

```

<reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-
link-type="contained" link-direction="main-to-connected">

```

調整式は OR の形式で、それぞれの OR には AND が含まれます。

is-ordered : 調整をオーダー形式で行うか、通常の OR 比較で行うか決定します。

```
<or is-ordered="true">
```

調整プロパティをメイン・クラスから取得する場合は (マルチノード), **attribute** を使用するか, **connected-node-attribute** タグを使用します。

ignore-case. true : Universal CMDB クラス・モデルのデータを RDBMS のデータと比較する場合, 大文字 / 小文字は区別されません。

```

    <attribute CMDB-attribute-name="host_hostname" column-
name="[column_name]" ignore-case="true"/>
    <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="[column_name]"/>
  </or>
</reconciliation-by-two-nodes>
</CMDB-class>
<class CMDB-class-name="[CMDB_class_name]" default-table-
name="[default_table_name]" connected-CMDB-class-name="host" link-class-
name="container_f">

```

カラム名は外部キー・カラム (マルチノード・プライマリ・キー・カラムを指示する値の含まれたカラム) の名前です。

マルチノード・プライマリ・キー・カラムが複数のカラムで構成されている場合は, 各プライマリ・キー・カラムごとに 1 つ, 複数の外部キー・カラムがなければなりません。

```
<foreign-primary-key column-name="[column_name]" CMDB-class-primary-key-
column="[column_name]"/>
```

プライマリ・キー・カラムが少ない場合は, このカラムを複製します。

```
<primary-key column-name="[column_name]"/>
```

from-CMDB-converter および **to-CMDB-converter** プロパティは、次のインタフェースを実行する Java クラスです。

- ▶ `com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.FcldbDalTransformerFromExternalDB`
- ▶ `com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.FcldbDalTransformerToExternalDB`

CMDB とデータベースの値が同じでない場合は、これらのコンバータを使います。たとえば、CMDB のホスト名には、`mer.com` というサフィックスが付いています。

この例では、丸括弧内に記入された XML ファイル (**generic-enum-transformer-example.xml**) に従って列挙子を変換するのに、`GenericEnumTransformer` を使用しています。

```
<attribute CMDB-attribute-name="[CMDB_attribute_name]" column-
name="[column_name]" from-CMDB-
converter="com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.GenericE
numTransformer(generic-enum-transformer-example.xml)" to-CMDB-
converter="com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.GenericE
numTransformer(generic-enum-transformer-example.xml)"/>
  <attribute CMDB-attribute-name="[CMDB_attribute_name]" column-
name="[column_name]"/>
  <attribute CMDB-attribute-name="[CMDB_attribute_name]" column-
name="[column_name]"/>
</class>
</generic-DB-adapter-config>
```

単純マッピングの例

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="host" default-table-name="Device">
    <primary-key column-name="Device_ID"/>
    <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-link-
type="contained">
      <or>
        <attribute CMDB-attribute-name="host_hostname" column-name="Device_Name"/>
        <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PreferredIPAddress"/>
      </or>
    </reconciliation-by-two-nodes>
  </CMDB-class>
</generic-DB-adapter-config>
```

```

</CMDB-class>
<class CMDB-class-name="host_card" default-table-name="hwCards" connected-CMDB-class-
name="host" link-class-name="contained">
  <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-column="Device_ID"/>
  <primary-key column-name="Device_ID"/>
  <primary-key column-name="hwBusesSupported_Seq"/>
  <primary-key column-name="hwCards_Seq"/>
  <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
  <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVendor"/>
  <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
</class>
<class CMDB-class-name="sw_sub_component" default-table-name="SWSubComponent"
connected-CMDB-class-name="host" link-class-name="contained">
  <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-column="Device_ID"/>
  <primary-key column-name="Device_ID"/>
  <primary-key column-name="Version_ID"/>
  <attribute CMDB-attribute-name="installed_dir" column-
name="SWSubComponent_InstalledDirectory"/>
  <attribute CMDB-attribute-name="license_required" column-
name="SWSubComponent_LicenceRequired"/>
  <attribute CMDB-attribute-name="last_access_time" column-
name="SWSubComponent_LastAccessTimeStamp"/>
  <attribute CMDB-attribute-name="last_access_time_string" column-
name="SWSubComponent_LastAccessTimeStamp"/>
</class>
<class CMDB-class-name="host_scsi_device" default-table-name="hwSCSIDevices" connected-
CMDB-class-name="host" link-class-name="contained">
  <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-column="Device_ID"/>
  <primary-key column-name="Device_ID"/>
  <primary-key column-name="hwSCSIDevices_Seq"/>
  <attribute CMDB-attribute-name="scsi_device_name" column-name="hwSCSIDeviceName"/>
  <attribute CMDB-attribute-name="scsi_device_vendor" column-name="hwSCSIDeviceVendor"/>
  <attribute CMDB-attribute-name="scsi_device_type" column-name="hwSCSIDeviceType"/>
</class>
</generic-DB-adapter-config>

```


制限事項

- ▶ (データベース・ソースで) 1ノードの TQL をマップする場合にのみ使用できません。たとえば、`host > ticket` と `ticket` TQL を実行できます。データベースからノードの階層を取り出すには、詳細な `orm.xml` ファイルを使う必要があります。
- ▶ 1 対多の関係だけがサポートされています。たとえば、各ホストに 1 つ以上のチケットを持ち込むことができます。複数のホストに属するチケットは持ち込めません。
- ▶ 同じクラスを異なるタイプの CMDB CIT に接続することはできません。たとえば、`ticket` を `host` に接続すると定義すると、`application` に接続することはできません。

orm.xml ファイル

このファイルは、CMDB CIT をデータベース・テーブルにマッピングするのに使用されます。

新規ファイルを作成するのに使用するテンプレートは、

< HP Universal CMDB ルート・ディレクトリ > ¥UCMDBServer¥j2f¥fcmdb¥CodeBase¥GenericDBAdapter¥META-INF フォルダにあります。

本項の内容

- ▶ 177 ページ「テンプレート」
- ▶ 181 ページ「複数の ORM ファイル」
- ▶ 181 ページ「命名規則」

テンプレート

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd">
  <description>Generic DB adapter orm</description>
```

パッケージ名は変更しないでください。

```
<package>generic_DB_adapter</package>
```

entity : Universal CMDDB CIT 名。これは multinode エンティティです。

name は **class** と同じにして、**class** には **generic_DB_adapter**. プレフィックスが含まれるようにします。

```
<entity name="host" class="generic_DB_adapter.host">  
  <table name="[table_name]"/>
```

エンティティを複数のテーブルにマップする場合は、セカンダリ・テーブルを使用します。

```
<secondary-table name=""/>  
<attributes>
```

識別子による単一テーブル継承の場合は、次のコードを使用します。

```
<inheritance strategy="SINGLE_TABLE"/>  
<discriminator-value>host</discriminator-value>  
<discriminator-column name="[column_name]"/>
```

id タグのある属性がプライマリ・キー・カラムです。これらのプライマリ・キー・カラムの命名規則は **idX** (id1, id2 など) であり、**X** はプライマリ・キーのカラム・インデックスです。

```
<id name="id1">
```

プライマリ・キーのカラム名のみを変更します。

```
<column updatable="false" insertable="false" name="[column_name]"/>  
<generated-value strategy="TABLE"/>  
</id>
```

basic : CMDB 属性を宣言するのに使用します。**name** および **column_name** プロパティだけを編集します。この式は `reconciliation_rules.txt` ファイルにあります。

```
<basic name="host_hostname">
  <column updatable="false" insertable="false" name="[column_name]"/>
</basic>
<basic name="ip_ip_address">
  <column updatable="false" insertable="false" name="[column_name]"/>
</basic>
</attributes>
</entity>
```

識別子による単一テーブル継承の場合は、拡張クラスを次のようにマップします。

```
<entity name="[cmdb_class_name]" class="generic_DBdb_adapter.nt" name="nt">
  <discriminator-value>nt</discriminator-value>
  <attributes/>
</entity>
<entity class="generic_DB_adapter.unix" name="unix">
  <discriminator-value>unix</discriminator-value>
  <attributes/>
</entity>
<entity name="[CMDB_class_name]"
class="generic_DB_adapter.[CMDB[cmdb_class_name]]">
  <table name="[default_table_name]"/>
  <secondary-table name=""/>
  <attributes>
    <id name="id1">
      <column updatable="false" insertable="false" name="[column_name]"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id2">
      <column updatable="false" insertable="false" name="[column_name]"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id3">
      <column updatable="false" insertable="false" name="[column_name]"/>
      <generated-value strategy="TABLE"/>
    </id>
```

次の例に、プレフィックスのない CMDB 属性名を示します。

```
<basic name="[CMDB_attribute_name]">
  <column updatable="false" insertable="false" name="[column_name]"/>
</basic>
<basic name="[CMDB_attribute_name]">
  <column updatable="false" insertable="false" name="[column_name]"/>
</basic>
<basic name="[CMDB_attribute_name]">
  <column updatable="false" insertable="false" name="[column_name]"/>
</basic>
</attributes>
</entity>
```

これは関係エンティティです。命名規則は **end1Type_linkType_end2Type** です。この例では、**end1Type** は **host** で、**linkType** は **container_f** です。

```
<entity name="host_container_f_[CMDB_class_name]"
class="generic_DB_adapter.host_container_f_[CMDB_class_name]">
  <table name="[default_table_name]"/>
  <attributes>
    <id name="id1">
      <column updatable="false" insertable="false" name="[column_name]"/>
      <generated-value strategy="TABLE"/>
    </id>
```

ターゲット・エンティティは、このプロパティが指示するエンティティです。この例では、**end1** が **host** エンティティにマップされます。

many-to-one : 1つのホストに多数の関係を接続できます。

join-column : **end1** ID (ターゲット・エンティティ ID) が含まれているカラム。

referenced-column-name : join カラムに使用する ID が含まれているターゲット・エンティティ (**host**) のカラム名。

```
<many-to-one target-entity="host" name="end1">
  <join-column updatable="false" insertable="false" referenced-column-
name="[column_name]" name="[column_name]"/>
</many-to-one>
```

one-to-one : 1 つの [CMDB_class_name] に 1 つの関係を接続できます。

```
<one-to-one target-entity="[CMDB_class_name]" name="end2">
  <join-column updatable="false" insertable="false" referenced-column-
name="" name="[column_name]"/>
</one-to-one>
</attributes>
</entity>
</entity-mappings>
```

複数の ORM ファイル

バージョン 7.5.1 から、複数のマッピング・ファイルがサポートされています。各マッピング・ファイル名は、最後に **orm.xml** を付けてください。マッピング・ファイルはすべて、アダプタの META-INF フォルダに置いてください。

命名規則

- ▶ 各エンティティでは、クラス・プロパティが **generic_DB_adapter** というプレフィックスのある名前プロパティに一致する必要があります。
- ▶ プライマリ・キー・カラムは、テーブルにあるプライマリ・キーの番号に従って、**idX** 形式 (**X = 1, 2, ...**) の名前を取る必要があります。
- ▶ エンティティに複数のプライマリ・キー・カラムがある場合は、**id** クラスを別個のエンティティとして追加する必要があります。この名前は **IdClassXPK_cmdbClassName** という形式を取ります。**X** はプライマリ・キー・カラムの数に置き換えられ、**className** はこの **id** クラスが関連するエンティティ名に置き換えられます。
- ▶ エンティティ名は大文字/小文字に関しても、クラス名と一致する必要があります。
- ▶ 属性名は大文字/小文字に関しても、クラス属性名と一致する必要があります。
- ▶ この関係名は **end1Type_linkType_end2Type** という形式を取ります。
- ▶ Java の予約語でもある CMDB CIT には、**gdba_** というプレフィックスを付けてください。たとえば、CMDB CIT **goto** の場合、ORM エンティティは **gdba_goto** という名前にします。

reconciliation_rules.txt ファイル

このファイルは調整ルールを設定するのに使います。

このファイルの各行がルールを示します。次に例を示します。

```
multinode[host] expression[^host.host_hostname OR ip.ip_address] end1_type[host]
end2_type[ip] link_type[contained]
```

`multinode` にはマルチノード名（TQL のフェデレート・データベース CIT に接続する CMDB CIT）を記入します。

この式には、2つのマルチノードが同じかどうかを判断するロジックが含まれています（一方のマルチノードは UCMDDB にあり、もう一方はデータベース・ソースにあります）。

この式は OR または AND で構成されています。

式の中で属性名に関する規則の部分は `[className].[attributeName]` です。たとえば、`ip` クラスの属性 `ip_address` は `ip.ip_address` と記述されます。

順序指定一致の場合は（最初の OR 副次式によってマルチノードが同じでないという応答が返されると、2番目の OR 副次式は比較されません）、`expression` ではなく、`ordered expression` を使用します。

比較で大文字/小文字を無視するには、コントロール記号（`^`）を使います。

`end1_type`、`end2_type`、および `link_type` パラメータを使用するのは、調整 TQL が単なるマルチモードではなく、2つのノードが含まれている場合のみとなります。この場合、調整 TQL は `end1_type > (link_type) > end2_type` です。

関連レイアウトは式から取り出されるので、追加する必要はありません。

transformations.txt ファイル

このファイルには、コンバータ定義がすべて含まれています。

この形式では、各行に新しい定義が含まれます。

テンプレート

```
entity[[CMDB_class_name]] attribute[[CMDB_attribute_name]]
to_DB_class[com.mercury.topaz.fcldb.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)]
```

entity : orm.xml ファイルに表示されるエンティティ名。

attribute : orm.xml ファイルに表示される属性名。

to_DB_class : **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.**

transform.FcldbDalTransformerToExternalDB インタフェースを実装するクラスの完全な修飾名。丸括弧内の要素が、このクラス・コンストラクタに設定されます。このコンバータは、CMDB 値をデータベース値に変換する（.com というサフィックスを各ホスト名に付加するなど）のに使用します。

from_DB_class : **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.**

transform.FcldbDalTransformerFromExternalDB インタフェースを実装するクラスの完全な修飾名。丸括弧内の要素が、このクラス・コンストラクタに設定されます。このコンバータは、データベース値を CMDB 値に変換する（.com というサフィックスを各ホスト名に付加するなど）のに使用します。

詳細については、186 ページ「用意済みのコンバータ」を参照してください。

persistence.xml ファイル

このファイルは標準設定の Hibernate 設定をオーバーライドしたり、用意されていないデータベース・タイプのサポートを追加するのに使用します（OOB データベース・タイプは Oracle Server, Microsoft MSSQL Server, および MySQL です）。

新しいデータベース・タイプをサポートする必要がある場合は、接続プール・プロバイダ（標準設定は c3p0）とデータベース用の JDBC ドライバを用意します（*.jar ファイルをアダプタ・フォルダに入れます）。

変更できる Hibernate 値をすべて確認するには、**org.hibernate.cfg.Environment** クラスをチェックします。

persistence.xml ファイルの例

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
  <!-- Don't change this value -->
  <persistence-unit name="GenericDBAdapter">
    <properties>
      <!-- Don't change this value -->
      <property name="hibernate.archive.autodetection" value="class, hbm"/>
      <!--The driver class name"/-->
      <property name="hibernate.connection.driver_class"
value="com.mercury.jdbc.MercOracleDriver"/>
      <!--The connection url"/-->
      <property name="hibernate.connection.url"
value="jdbc:mercury:oracle://artist:1521;sid=cmdb2"/>
      <!--DB login credentials"/-->
      <property name="hibernate.connection.username" value="CMDB"/>
      <n property name="hibernate.connection.password" value="CMDB"/>
      <!--connection pool properties"/-->
      <property name="hibernate.c3p0.min_size" value="5"/>
      <property name="hibernate.c3p0.max_size" value="20"/>
      <property name="hibernate.c3p0.timeout" value="300"/>
      <property name="hibernate.c3p0.max_statements" value="50"/>
      <property name="hibernate.c3p0.idle_test_period" value="3000"/>
      <!--The dialect to use-->
      <property name="hibernate.dialect"
value="org.hibernate.dialect.OracleDialect"/>
    </properties>
  </persistence-unit>
</persistence>
```


discriminator.properties ファイル

このファイルによって、サポートされている各 CI タイプ（orm.xml で識別子値としても使用されます）が、識別子カラムの対応する値のカンマ区切りリストにマップされます。

識別子マッピングの例

discriminator.properties ファイルには次のコードが記述されています。

```
host=10001, 10005,10010,10011,10012
nt=10002,10003
unix=10004,10006,10008
```

orm.xml ファイルには次のコードが記述されています。

```
<entity class="generic_DB_adapter.host" name="host">
  <table name="[table_name]"/>
  ...
  <inheritance strategy="SINGLE_TABLE"/>
  <discriminator-value>host</discriminator-value>
  <discriminator-column name="[discriminator_column]"/>
  ...
</entity>
<entity class="generic_DB_adapter.nt" name="nt">
  <discriminator-value>nt</discriminator-value>
  <attributes/>
</entity>
<entity class="generic_DB_adapter.unix" name="unix">
  <discriminator-value>unix</discriminator-value>
  <attributes/>
</entity>
```

[discriminator_column] 属性は次のように計算されます。

- ▶ 対応するテーブルの識別子カラムに、特定のエントリを示す 10002 が含まれます。このエントリは **nt** CIT にマップされます。
- ▶ 対応するテーブルの識別子カラムに、特定のエントリを示す 10006 が含まれます。このエントリは **unix** CIT にマップされます。
- ▶ 対応するテーブルの識別子カラムに、特定のエントリを示す 10010 が含まれます。このエントリは **host** CIT にマップされます。

host CIT は **nt** と **unix** の親でもあることに注意してください。

replication_config.txt ファイル

このファイルには、CI と関係タイプのカンマ区切りリストが含まれていて、そのプロパティ条件はレプリケーション・プラグインでサポートされています。詳細については、190 ページ「プラグイン」を参照してください。

fixed_values.txt ファイル

このファイルでは、特定の CIT に関する個別の属性に固定値を設定できます。このような方法で、これらの各属性には、データベースに保管されていない固定値を割り当てることができます。

このファイルには、ゼロ以上のエントリが次の形式で含まれます。

```
entity[<entityName>] attribute[<attributeName>] value[<value>]
```

次に例を示します。

```
entity[ip] attribute[ip_domain] value[DefaultDomain]
```

用意済みのコンバータ

次のコンバータ（変換子）を使って、フェデレート・クエリおよびレプリケーション・ジョブをデータベースのデータに、またはその逆に変換できます。

本項の内容

- ▶ 187 ページ 「enum-transformer コンバータ」
- ▶ 189 ページ 「SuffixTransformer コンバータ」
- ▶ 189 ページ 「PrefixTransformer コンバータ」
- ▶ 189 ページ 「BytesToStringTransformer コンバータ」

enum-transformer コンバータ

このコンバータでは、入力パラメータとして与えられる XML ファイルを使用します。

XML ファイルはハードコードの CMDb 値とデータベース値 (enums) 間にマップされます。いずれかの値が存在しない場合は、同じ値を返すか、NULL を返すか、例外処理を実行するか選択できます。

各エンティティ属性ごとに1つのXMLマッピング・ファイルを使用します。

注：このコンバータは、transformations.txt ファイルの to_DB_class および from_DB_class フィールドに使用できます。

入力ファイル XSD の例

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="enum-transformer">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="value" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="DB-type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="integer"/>
            <xs:enumeration value="long"/>
            <xs:enumeration value="float"/>
            <xs:enumeration value="double"/>
            <xs:enumeration value="boolean"/>
            <xs:enumeration value="string"/>
            <xs:enumeration value="date"/>
            <xs:enumeration value="xml"/>
            <xs:enumeration value="bytes"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="CMDb-type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="integer"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```

        <xs:enumeration value="long"/>
        <xs:enumeration value="float"/>
        <xs:enumeration value="double"/>
        <xs:enumeration value="boolean"/>
        <xs:enumeration value="string"/>
        <xs:enumeration value="date"/>
        <xs:enumeration value="xml"/>
        <xs:enumeration value="bytes"/>
    </xs:restriction>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="non-existing-value-action" use="required">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="return-null"/>
            <xs:enumeration value="return-original"/>
            <xs:enumeration value="throw-exception"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
</xs:complexType>
</xs:element>
<xs:element name="value">
    <xs:complexType>
        <xs:attribute name="CMDB-value" type="xs:string" use="required"/>
        <xs:attribute name="external-DB-value" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
</xs:schema>

```

sys 値を System 値に変換する例

この例では、CMDB の **sys** 値がフェデレート・データベースの **System** 値に変換され、フェデレート・データベースの **System** 値が CMDB の **sys** 値に変換されます。

XML ファイルに値がない場合は（文字列 **demo** など）、コンバータが受信する同じ入力値を返します。

```

<enum-transformer CMDB-type="string" DB-type="string" non-existing-value-action="return-original"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../META-
CONF/generic-enum-transformer.xsd">
    <value CMDB-value="sys" external-DB-value="System"/>
</enum-transformer>

```

SuffixTransformer コンバータ

このコンバータは、CMDB またはフェデレート・データベース・ソース値にサフィックスを追加または削除するのに使います。

2つの実装があります。

- ▶ **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddSuffixTransformer** : フェデレート・データベース値から CMDB 値に変換するときにサフィックス（入力として指定）を追加し、CMDB 値からフェデレート・データベース値に変換するときにサフィックスを削除します。
- ▶ **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemoveSuffixTransformer** : フェデレート・データベース値から CMDB 値に変換するときにサフィックス（入力として指定）を削除し、CMDB 値からフェデレート・データベース値に変換するときにサフィックスを追加します。

PrefixTransformer コンバータ

このコンバータは、CMDB またはフェデレート・データベース値にプレフィックスを追加または削除するのに使います。

2つの実装があります。

- ▶ **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddPrefixTransformer** : フェデレート・データベース値から CMDB 値に変換するときにプレフィックス（入力として指定）を追加し、CMDB 値からフェデレート・データベース値に変換するときにプレフィックスを削除します。
- ▶ **com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemovePrefixTransformer** : フェデレート・データベース値から CMDB 値に変換するときにプレフィックス（入力として指定）を削除し、CMDB 値からフェデレート・データベース値に変換するときにプレフィックスを追加します。

BytesToStringTransformer コンバータ

このコンバータは、UCMDB のバイト配列をフェデレート・データベース・ソースの文字列表現に変換するのに使います。

このコンバータは

com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytesToStringTransformer です。

プラグイン

汎用データベース・アダプタでは、次のプラグインをサポートしています。

- ▶ トポロジを完全に同期化する必須プラグイン。
- ▶ トポロジの変更を同期化する必須プラグイン。
- ▶ レイアウトを同期化する必須プラグイン。
- ▶ サポートされている同期化のクエリを取得するオプションのプラグイン。このプラグインが定義されていないと、すべての TQL 名が返されます。
- ▶ TQL 定義および TQL 結果を変更する内部のオプション・プラグイン。
- ▶ レイアウト要求および CI 結果を変更する内部のオプション・プラグイン。
- ▶ レイアウト要求および関係結果を変更する内部のオプション・プラグイン。

プラグインは、アダプタの META-INF フォルダにある **plugins.txt** ファイルを使って設定します。

設定例

本項では設定例を示します。

本項の内容

- ▶ 190 ページ「使用例」
- ▶ 191 ページ「単一ノード調整」
- ▶ 193 ページ「2 ノード調整」
- ▶ 198 ページ「複数のカラムが含まれているプライマリ・キーの使い方」
- ▶ 201 ページ「変換の仕方」

使用例

使用例：TQL は次のとおりです。

```
host > (container_f) > host_card
```

各パラメータの説明：

host は UCMDB エンティティです。

host_card はフェデレート・データベース・ソース・エンティティです。

container_f はそれらの関係です。

この例は ED データベースに対して実行されます。ED hosts は Device テーブルに保管され、host_card は hwCards テーブルに保管されます。次の例では、host_card がいつも同じ方法でマップされます。

単一ノード調整

この例では、host_hostname プロパティに対して調整が実行されます。

簡単な定義

multinode はホストで、**CMDB-class** という特別なタグで強調されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..META-CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="host" default-table-name="Device">
    <primary-key column-name="Device_ID"/>
    <reconciliation-by-single-node>
      <or>
        <attribute CMDB-attribute-name="host_hostname" column-name="Device_Name"/>
      </or>
    </reconciliation-by-single-node>
  </CMDB-class>
  <class CMDB-class-name="host_card" default-table-name="hwCards" connected-CMDB-class-
name="host" link-class-name="container_f">
    <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-column="Device_ID"/>
    <primary-key column-name="hwCards_Seq"/>
    <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
    <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVendor"/>
    <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
  </class>
</generic-DB-adapter-config>
```

詳細な定義

orm.xml ファイル

関係マッピングの追加に注意してください。詳細については、177 ページ「orm.xml ファイル」の定義セクションを参照してください。

orm.xml ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd" version="1.0">
  <description>Generic DB adapter orm</description>
  <package>generic_DB_adapter</package>
  <entity class="generic_DB_adapter.host" name="host">
    <table name="Device"/>
    <attributes>
      <id name="id1">
        <column name="Device_ID" insertable="false" updatable="false"/>
        <generated-value strategy="TABLE"/>
      </id>
      <basic name="host_hostname">
        <column name="Device_Name"/>
      </basic>
    </attributes>
  </entity>
  <entity class="generic_DB_adapter.host_card" name="host_card">
    <table name="hwCards"/>
    <attributes>
      <id name="id1">
        <column name="hwCards_Seq" insertable="false" updatable="false"/>
        <generated-value strategy="TABLE"/>
      </id>
      <basic name="card_class">
        <column name="hwCardClass" insertable="false" updatable="false"/>
      </basic>
      <basic name="card_vendor">
        <column name="hwCardVendor" insertable="false" updatable="false"/>
      </basic>
      <basic name="card_name">
        <column name="hwCardName" insertable="false" updatable="false"/>
      </basic>
    </attributes>
  </entity>
```



```

<entity class="generic_DB_adapter.host_container_f_host_card"
name="host_container_f_host_card">
  <table name="hwCards"/>
  <attributes>
    <id name="id1">
      <column name="hwCards_Seq" insertable="false" updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <many-to-one name="end1" target-entity="host">
      <join-column name="Device_ID" insertable="false" updatable="false"/>
    </many-to-one>
    <one-to-one name="end2" target-entity="host_card">
      <join-column name="hwCards_Seq" referenced-column-name="hwCards_Seq"
insertable="false" updatable="false"/>
    </one-to-one>
  </attributes>
</entity>
</entity-mappings>

```

reconciliation_rules.txt ファイル

詳細については、182 ページ「reconciliation_rules.txt ファイル」を参照してください。

```
multinode[host] expression[host.host_hostname]
```

transformation.txt ファイル

この例では値を変換する必要がないので、このファイルは空のままです。

2 ノード調整

この例では、host_hostname および ip_address プロパティとさまざまなバリエーションに従って調整が計算されます。

調整 TQL は **host > (contained) > ip** です。

簡単な定義

この調整は host_hostname OR ip_address です。

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="host" default-table-name="Device">
    <primary-key column-name="Device_ID"/>
    <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-
link-type="contained">
      <or>
        <attribute CMDB-attribute-name="host_hostname" column-
name="Device_Name"/>
        <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PREFERREDIPAddress"/>
      </or>
    </reconciliation-by-two-nodes>
  </CMDB-class>
  <class CMDB-class-name="host_card" default-table-name="hwCards" connected-
CMDB-class-name="host" link-class-name="contained">
    <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID"/>
    <primary-key column-name="hwCards_Seq"/>
    <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
    <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor"/>
    <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
  </class>
</generic-DB-adapter-config>
```

この調整は host_hostname AND ip_address です。

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../META-CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="host" default-table-name="Device">
    <primary-key column-name="Device_ID"/>
    <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-
link-type="contained">
      <and>
        <attribute CMDB-attribute-name="host_hostname" column-
name="Device_Name"/>
        <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PREFERREDIPAddress"/>
      </and>
    </reconciliation-by-two-nodes>
  </CMDB-class>
  <class CMDB-class-name="host_card" default-table-name="hwCards" connected-
CMDB-class-name="host" link-class-name="contained">
    <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID"/>
    <primary-key column-name="hwCards_Seq"/>
    <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
    <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor"/>
    <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
  </class>
</generic-DB-adapter-config>
```

この調整は `ip_address` です。

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="META-CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="host" default-table-name="Device">
    <primary-key column-name="Device_ID"/>
    <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-
link-type="contained">
      <or>
        <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PREFERREDIPAddress"/>
      </or>
    </reconciliation-by-two-nodes>
  </CMDB-class>
  <class CMDB-class-name="host_card" default-table-name="hwCards" connected-
CMDB-class-name="host" link-class-name="contained">
    <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID"/>
    <primary-key column-name="hwCards_Seq"/>
    <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
    <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor"/>
    <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
  </class>
</generic-DB-adapter-config>
```

詳細な定義

orm.xml ファイル

このファイルには調整式が定義されていないので、OR および AND の両方にも、`ip_address` だけでも同じバージョンを使用できます。

orm.xml ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd" version="1.0">
  <description>Generic DB adapter orm</description>
  <package>generic_DB_adapter</package>
  <entity class="generic_DB_adapter.host" name="host">
    <table name="Device"/>
    <attributes>
```

```

<id name="id1">
  <s column name="Device_ID" insertable="false" updatable="false"/>
  <generated-value strategy="TABLE"/>
</id>
<basic name="host_hostname">
  <column name="Device_Name" insertable="false" updatable="false"/>
</basic>
<basic name="ip_ip_address">
  <column name="Device_PreferredIPAddress" insertable="false" updatable="false"/>
</basic>
</attributes>
</entity>
<entity class="generic_DB_adapter.host_card" name="host_card">
  <table name="hwCards"/>
  <attributes>
    <id name="id1">
      <column name="hwCards_Seq" insertable="false" updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <basic name="card_class">
      <column name="hwCardClass" insertable="false" updatable="false"/>
    </basic>
    <basic name="card_vendor">
      <column name="hwCardVendor" insertable="false" updatable="false"/>
    </basic>
    <basic name="card_name">
      <column name="hwCardName" insertable="false" updatable="false"/>
    </basic>
  </attributes>
</entity>
<entity class="generic_DB_adapter.host_container_f_host_card"
name="host_container_f_host_card">
  <table name="hwCards"/>
  <attributes>
    <id name="id1">
      <column name="hwCards_Seq" insertable="false" updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <many-to-one name="end1" target-entity="host">
      <join-column name="Device_ID" insertable="false" updatable="false"/>
    </many-to-one>
    <one-to-one name="end2" target-entity="host_card">
      <join-column name="hwCards_Seq" referenced-column-name="hwCards_Seq"
insertable="false" updatable="false"/>
    </one-to-one>
  </attributes>

```

```
</entity>  
</entity-mappings>
```

reconciliation_rules.txt ファイル

詳細については、182 ページ「reconciliation_rules.txt ファイル」を参照してください。

```
multinode[host] expression[ip.ip_address OR host.host_hostname] end1_type[host]  
end2_type[ip] link_type[contained]
```

```
multinode[host] expression[ip.ip_address AND host.host_hostname] end1_type[host]  
end2_type[ip] link_type[contained]
```

```
multinode[host] expression[ip.ip_address] end1_type[host] end2_type[ip]  
link_type[contained]
```

transformation.txt ファイル

この例では値を変換する必要がないので、このファイルは空のままです。

複数のカラムが含まれているプライマリ・キーの使い方

プライマリ・キーが複数のカラムで構成されている場合は、次のコードを XMLS 定義に追加します。

簡単な定義

複数のプライマリ・キー・タグがあり、各カラムごとにタグがあります。

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="host" default-table-name="Device">
    <primary-key column-name="Device_ID"/>
    <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-
link-type="contained">
      <or>
        <attribute CMDB-attribute-name="host_hostname" column-
name="Device_Name"/>
        <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PreferedIPAddress"/>
      </or>
    </reconciliation-by-two-nodes>
  </CMDB-class>
  <class CMDB-class-name="host_card" default-table-name="hwCards" connected-
CMDB-class-name="host" link-class-name="contained">
    <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID"/>
    <primary-key column-name="Device_ID"/>
    <primary-key column-name="hwBusesSupported_Seq"/>
    <primary-key column-name="hwCards_Seq"/>
    <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
    <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor"/>
    <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
  </class>
</generic-DB-adapter-config>
```

詳細な定義

orm.xml ファイル

プライマリ・キー・カラムにマップされる新しい id エンティティを追加します。この id エンティティを使用するエンティティで、特別なタグを追加する必要があります。

このようなプライマリ・キーに外部キーを使用する場合は、外部キーの各カラムをプライマリ・キーのカラムにマップする必要があります。

詳細については、177 ページ「orm.xml ファイル」を参照してください。

orm.xml ファイルの例

```

<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd" version="1.0">
  <description>Generic DB adapter orm</description>
  <package>generic_DB_adapter</package>
  <entity class="generic_DB_adapter.host" name="host">
    <table name="Device"/>
    <attributes>
      <id name="id1">
        <column name="Device_ID" insertable="false" updatable="false"/>
        <generated-value strategy="TABLE"/>
      </id>
      <basic name="host_hostname">
        <column name="Device_Name"/>
      </basic>
    </attributes>
  </entity>
  <entity class="generic_DB_adapter.host_card" name="host_card">
    <table name="hwCards"/>
    <attributes>
      <id name="id1">
        <column name="Device_ID" insertable="false" updatable="false"/>
        <generated-value strategy="TABLE"/>
      </id>
      <id name="id2">
        <column name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
        <generated-value strategy="TABLE"/>
      </id>
      <id name="id3">
        <column name="hwCards_Seq" insertable="false" updatable="false"/>
        <generated-value strategy="TABLE"/>
      </id>
      <basic name="card_class">
        <column name="hwCardClass" insertable="false" updatable="false"/>
      </basic>
      <basic name="card_vendor">
        <column name="hwCardVendor" insertable="false" updatable="false"/>
      </basic>
      <basic name="card_name">
        <column name="hwCardName" insertable="false" updatable="false"/>
      </basic>
    </attributes>
  </entity>

```



```

<entity class="generic_DB_adapter.host_contained_host_card" name="host_contained_host_card">
  <table name="hwCards"/>
  <attributes>
    <id name="id1">
      <column name="Device_ID" insertable="false" updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id2">
      <column name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <id name="id3">
      <column name="hwCards_Seq" insertable="false" updatable="false"/>
      <generated-value strategy="TABLE"/>
    </id>
    <many-to-one name="end1" target-entity="host">
      <join-column name="Device_ID" insertable="false" updatable="false"/>
    </many-to-one>
    <one-to-one name="end2" target-entity="host_card">
      <join-column name="Device_ID" referenced-column-name="Device_ID" insertable="false"
updatable="false"/>
      <join-column name="hwBusesSupported_Seq" referenced-column-
name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
      <join-column name="hwCards_Seq" referenced-column-name="hwCards_Seq"
insertable="false" updatable="false"/>
    </one-to-one>
  </attributes>
</entity>
</entity-mappings>

```

変換の仕方

次の例では、一般的な **enum** 変換子によって、`host_hostname` カラムで値 1, 2, 3 から値 **a**, **b**, **c** にそれぞれ変換されます。

マッピング・ファイルは `generic-enum-transformer-example.xml` です。

```

<enum-transformer CMDB-type="string" DB-type="string" non-existing-value-
action="return-original" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/generic-enum-transformer.xsd">
  <value CMDB-value="1" external-DB-value="a"/>
  <value CMDB-value="2" external-DB-value="b"/>
  <value CMDB-value="3" external-DB-value="c"/>
</enum-transformer>

```

簡単な定義

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="META-CONF/simplifiedConfiguration.xsd">
  <CMDB-class CMDB-class-name="host" default-table-name="Device">
    <primary-key column-name="Device_ID"/>
    <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip" CMDB-
link-type="contained">
      <or>
        <attribute CMDB-attribute-name="host_hostname" column-
name="Device_Name" from-CMDB-
converter="com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.GenericE
numTransformer(generic-enum-transformer-example.xml)" to-CMDB-
converter="com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.GenericE
numTransformer(generic-enum-transformer-example.xml)"/>
        <connected-node-attribute CMDB-attribute-name="ip_address" column-
name="Device_PREFERREDIPAddress"/>
      </or>
    </reconciliation-by-two-nodes>
  </CMDB-class>
  <class CMDB-class-name="host_card" default-table-name="hwCards" connected-
CMDB-class-name="host" link-class-name="contained">
    <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID"/>
    <primary-key column-name="hwCards_Seq"/>
    <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"/>
    <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor"/>
    <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
  </class>
</generic-DB-adapter-config>
```

詳細な定義

transformation.txt ファイルのみに変更があります。

transformation.txt ファイル

属性名とエンティティ名を orm.xml ファイルと同じにします。

```
entity[host] attribute[host_hostname]
to_DB_class[com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.Generic
EnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcldb.adapters.dbAdapter.dal.transform.impl.Gene
ricEnumTransformer(generic-enum-transformer-example.xml)]
```

フェデレート・データベース・ログ・ファイル

計算フロー，アダプタ・ライフサイクルを理解したり，デバッグ情報を表示するには，次のログ・ファイルを参照します。

本項の内容

- ▶ 203 ページ「ログ・レベル」
- ▶ 204 ページ「ログの保管場所」

ログ・レベル

各ログのログ・レベルを設定できます。

次のファイルをテキスト・エディタで開きます。

< HP Universal CMDDB ルート・ディレクトリ > ¥2f¥conf¥core¥Tools¥
log4j¥fcldb¥fcldb.gdba.properties

標準設定のログ・レベルは **ERROR** です。

```
#loglevel can be any of DEBUG INFO WARN ERROR FATAL
loglevel=ERROR
```

- ▶ すべてのログ・ファイルのログ・レベルを上げるには，**loglevel=ERROR** を **loglevel=DEBUG** または **loglevel=INFO** に変更します。

- ▶ 特定ファイルのログ・レベルを変更するには、特定の **log4j** カテゴリ行を適宜変更します。たとえば、`fcmdb.gdba.dal.sql.log` のログ・レベルを **INFO** に変更するには、次の行を変更します。

```
log4j.category.fcmdb.gdba.dal.SQL=${loglevel},fcmdb.gdba.dal.SQL.appender
```

次のように変更します。

```
log4j.category.fcmdb.gdba.dal.SQL=INFO,fcmdb.gdba.dal.SQL.appender
```

ログの保管場所

ログ・ファイルは、**< HP Universal CMDB ルート・ディレクトリ > ¥j2f¥log¥ fcmdb¥**ディレクトリに保管されます。

▶ **Fcmdb.gdba.log**

アダプタ・ライフサイクル・ログ。アダプタの開始または停止、当該アダプタでサポートしている CIT に関する詳細を提供します。

開始エラー（アダプタのロード/アンロード）を参照してください。

▶ **fcmdb.log**

例外を参照してください。

▶ **cmdb.log**

例外を参照してください。

▶ **Fcmdb.gdba.mapping.engine.log**

マッピング・エンジン・ログ。マッピング・エンジンが使用している調整 TQL、および接続フェーズで比較される調整トポロジに関する詳細を提供します。

データベースに関連 CI があることが分かっているにもかかわらず、TQL が結果を提供しないか、結果が予期しないものである場合は、このログを参照します（調整をチェックします）。

▶ **Fcmdb.gdba.TQL.log**

TQL ログ。TQL とその結果に関する詳細を提供します。

TQL が結果を返さず、マッピング・エンジン・ログがフェデレート・データ・ソースに結果がないことを示す場合は、このログを参照します。

▶ **Fcmdb.gdba.dal.log**

DAL ライフサイクル・ログ。CIT の生成に関する詳細とデータベース接続の詳細を提供します。

データベースに接続できない場合、またはクエリでサポートされていない CIT または属性がある場合は、このログを参照します。

▶ **Fcmdb.gdba.dal.command.log**

DAL 動作ログ。呼び出された DAL 内部動作に関する詳細を提供します（このログは `cmdb.dal.command.log` と似ています）。

▶ **Fcmdb.gdba.dal.SQL.log**

DAL SQL クエリ・ログ。呼び出された JPAQL（オブジェクト指向 SQL クエリ）とその結果に関する詳細を提供します。

データベースに接続できない場合、またはクエリでサポートされていない CIT または属性がある場合は、このログを参照します。

▶ **Fcmdb.gdba.hibernate.log**

Hibernate ログ。実行された SQL クエリ、各 JPAQL から SQL の解析、クエリの結果、Hibernate キャッシュに関するデータなどの詳細を提供します。Hibernate の詳細については、136 ページ「JPA プロバイダとしての Hibernate」を参照してください。

外部参照

JavaBeans 3.0 の仕様の詳細については、<http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html> を参照してください。

トラブルシューティングと制限事項

以下の制限に注意してください。

JPA 制限事項

- ▶ すべてのテーブルには、プライマリ・キー・カラムがなければなりません。
- ▶ CMDB クラスの属性名は、JavaBeans の命名規則に従う必要があります（たとえば、名前の最初は小文字でなければなりません）。
- ▶ クラス・モデルの1つの関係と接続する2つの CI は、データベースに直接関連する必要があります（たとえば、`host` を `ticket` に接続する場合は、それらを接続する外部キーまたはリンクがなければなりません）。
- ▶ 同じ CIT にマップされる複数のテーブルでは、同じプライマリ・キー・テーブルを共有する必要があります。

機能上の制限事項

- ▶ CMDB とフェデレート CIT との間には、手動で関係を作成できません。仮想の関係を定義するには、特別な関係ロジックを定義する必要があります（この関係はフェデレート・クラスのプロパティをベースにできます）。
- ▶ フェデレート CIT は `multinode` から継承できません（ほとんどの場合は、Host CIT の下に新しいフェデレート CIT を作成できないというものです）。
- ▶ フェデレート CIT は同じデータ・ストアにないかぎり、別のフェデレート CIT から継承できません。
- ▶ フェデレート・データを表示するには、フェデレート CIT 自体をその親 CIT ではなく、TQL に追加する必要があります（親 CIT を使用している場合は、フェデレート・インスタンスが結果に表示されません）。
- ▶ フェデレート CIT は相関ルールで CIT をトリガできませんが、相関 TQL に含めることはできます。
- ▶ フェデレート CIT はエンリッチメント TQL の一部になりますが、エンリッチメントを実行するノードとして使用することはできません（フェデレート CIT を追加、更新、または削除できません）。
- ▶ CI タイプ・リストのプロパティはサポートされていません。
- ▶ 条件でクラス修飾子を使用することはサポートされていません。
- ▶ サブグラフはサポートされていません。

- ▶ 複合関係はサポートされていません。
- ▶ 外部の CI CMDB id はキー属性ではなく、そのプライマリ・キーで構成されます。
- ▶ **bytes** タイプのカラムは、Microsoft SQL Server でプライマリ・キー・カラムとして使用できません。

第7章

Federation Framework SDK

本章では、API を使ってフェデレート・ソースから情報を取得する Federation Framework の機能について説明します。

本章の内容

概念

- ▶ Federation Framework — 概要 (210 ページ)
- ▶ アダプタおよびマッピングの Federation Framework とのやり取り (213 ページ)
- ▶ FTQL 用の Federation Framework フロー (214 ページ)
- ▶ レプリケーション用の Federation Framework フロー (225 ページ)
- ▶ HP Release Control フェデレーション・アダプタ (226 ページ)
- ▶ アダプタ・インタフェース (230 ページ)

タスク

- ▶ 新しい外部データ・ストア用のアダプタの追加 (231 ページ)

参照先

- ▶ アダプタ機能 (241 ページ)

Federation Framework — 概要

注：

- ▶ 「関係」は、「リンク」と同じ意味です。
 - ▶ 「CI」は、「オブジェクト」と同じ意味です。
 - ▶ 「グラフ」は、ノードとリンクの集合です。
 - ▶ 定義と用語の一覧については、『用語集』を参照してください。
-

Federation Framework は次の2つの機能を提供します。

- ▶ **オンザフライ・フェデレーション**：すべてのクエリが元のデータ・ストアに対して実行され、結果がすぐに HP Universal CMDB に作成されます。
- ▶ **データ・レプリケーション**：データ・ストア間でデータ（トポロジ・データと CI プロパティ）を複製します。

どちらのタイプのアクションでも、データ・ストアごとにアダプタが必要です。アダプタは、データ・ストアの特定の機能を提供し、必要なデータの取得、更新、またはその両方を実行できます。データ・ストアに対するすべての要求は、アダプタを介して行われます。

本項の内容

- ▶ 210 ページ「オンザフライ・フェデレーション」
- ▶ 212 ページ「データ・レプリケーション」

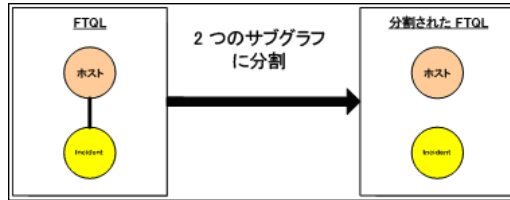
オンザフライ・フェデレーション

フェデレート TQL を使って、任意の外部データ・ストアからデータを複製せずに取得できます。

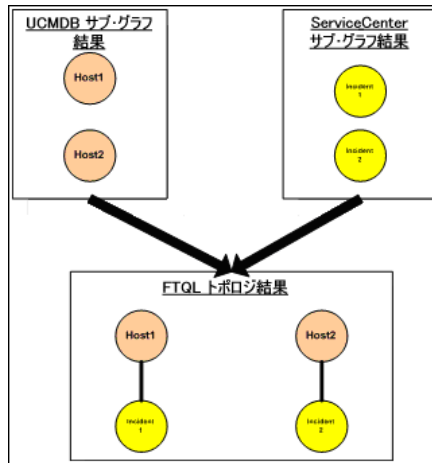
フェデレート TQL クエリは、外部データ・ストアを表すアダプタを使って、さまざまな外部データ・ストア CI と HP Universal CMDB CI の間に適切な外部関係を作成します。

オンザフライ・フェデレーション・フローの例

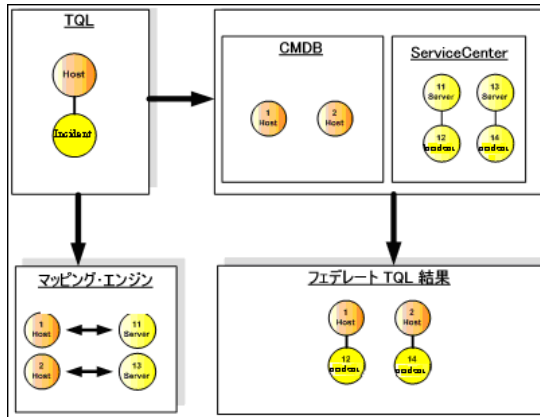
- 1 Federation Framework は、1つのフェデレート TQL (FTQL) を複数のサブグラフに分割します。サブグラフ内のすべてのノードは同じデータ・ストアを参照します。各サブグラフは、仮想関係によってほかのサブグラフに接続されます(ただし、サブグラフ自体に仮想関係は含まれていません)。



- 2 FTQL をサブグラフに分割した後で、Federation Framework は各サブグラフのトポロジを計算し、該当するノード間の仮想関係を作成することにより、該当する2つのサブグラフを接続します。



- 3 FTQL のトポロジを計算した後で、Federation Framework はトポロジ結果のレイアウトを取得します。



データ・レプリケーション

大量のデータを含む複数のデータ・ストアがあり、それらのデータ・ストアのデータを含むビューを使用する別のデータ・ストアがある場合は、データを複製します。

データ・レプリケーションでは、データ・ストアはソースとターゲットの2つに分類されます。データは、ソース・データ・ストアから取得され、ターゲット・データ・ストアに更新されます。レプリケーションはクエリ名に基づいて行われます。つまり、ソース・データ・ストアとターゲット・データ・ストアの間でデータの同期が行われ、ソース・データ・ストア内のクエリ名によってデータが取得されます。たとえば、HP Universal CMDB では TQL の名前がクエリ名です。しかし、別のデータ・ストアでは、データを返すコード名がクエリ名である可能性があります。アダプタは、クエリ名を正しく処理できるように設計されています。

ソース・データ・ストアの各クエリ名は、排他的クエリとして定義できます。つまり、クエリ結果の CI と関係がターゲット・データ・ストア内で一意になり、ほかのクエリではそれらをターゲットに提供できません。ソース・データ・ストアのアダプタは、特定のクエリをサポートし、そのデータ・ストアからデータを取得できます。ターゲット・データ・ストアのアダプタは、取得したデータをそのデータ・ストア上で更新できます。

レプリケーション・プロセスのフローには、以下の手順が含まれます。

- 1 ソース・データ・ストアからシグネチャを含むトポロジ結果を取得します。
- 2 新しい結果を前の結果と比較します。
- 3 変更された結果のみに関して、CI と関係の完全なレイアウト（つまり、すべての CI プロパティ）を取得します。
- 4 受け取った CI と関係の完全なレイアウトを使ってターゲット・データ・ストアを更新します。ソース・データ・ストアで CI または関係が削除され、クエリが排他的である場合は、レプリケーション・プロセスによって、ターゲット・データ・ストアでもその CI または関係が削除されます。

アダプタおよびマッピングの Federation Framework とのやり取り

アダプタは、外部データ（HP Universal CMDB に保存されないデータ）を表すフェデレート CMDB のエンティティです。フェデレート・フローでは、外部データ・ソースとのすべてのやり取りがアダプタを介して行われます。Federation Framework のやり取りのフローとアダプタ・インタフェースは、レプリケーションと FTQL で異なります。

本項の内容

- ▶ 213 ページ「アダプタのライフサイクル」
- ▶ 214 ページ「アダプタの assist メソッド」

アダプタのライフサイクル

アダプタ・インスタンスは、外部データ・ストアごとに作成されます。アダプタは、そのアダプタに最初に適用されたアクション（「TQL の計算」や「データの取得 / 更新」など）によってそのライフサイクルを開始します。**start** メソッドが呼び出されると、アダプタはデータ・ストアの設定やログ機能などの環境情報を受け取ります。アダプタのライフサイクルは、設定からデータ・ストアが削除され、**shutdown** メソッドが呼び出されたときに終了します。つまり、アダプタはステートフルであり、必要な場合は外部データ・ストアへの接続をアダプタに含めることができます。

アダプタの assist メソッド

アダプタには、外部データ・ストア設定を追加できる複数の **assist** メソッドがあります。これらのメソッドは、アダプタのライフサイクルには含まれず、呼び出すたびに新しいアダプタを作成します。

- ▶ 最初のメソッドは、特定の構成のために外部データ・ストアへの接続をテストします。
- ▶ 2番目のメソッドは、ソース・アダプタにのみ関係し、レプリケーション用のサポートされているクエリを返します。
- ▶ 3番目のメソッドは、FTQL にのみ関係し、外部データ・ストアによってサポートされている外部クラスを返します。

これらのメソッドは、いずれも新しいデータ・ストア設定を作成するときに使用されます。

FTQL 用の Federation Framework フロー

本項の内容

- ▶ 214 ページ「定義と用語」
- ▶ 215 ページ「マッピング・エンジン」
- ▶ 215 ページ「FTQL アダプタ」
- ▶ 216 ページ「フロー図」

定義と用語

調整データ：UCMDB と外部データ・ストアから取得された特定タイプの CI を照合するためのルール。調整ルールには次の 3 種類があります。

- ▶ **ID 調整**：これは、外部データ・ストアに調整オブジェクトの UCMDB ID が含まれている場合にのみ使用されます。
- ▶ **プロパティ調整**：これは、調整 CI タイプのプロパティによって照合が行われる場合にのみ使用されます。
- ▶ **トポロジ調整**：これは、調整 CI の照合を行うために追加の CIT のプロパティが必要な場合に使用されます。たとえば、ip CIT に属する ip_address プロパティによってホスト・タイプの調整を実行できます。

調整オブジェクト：このオブジェクトは、アダプタが受け取った調整データに従って作成します。このオブジェクトは、何らかの外部 CI を参照する必要があります。マッピング・エンジンは、このオブジェクトを使って外部 CI と UC MDB CI を接続します。

調整 CI タイプ：調整オブジェクトを表す CI のタイプ。これらの CI は、UC MDB と外部データ・ストアの両方に格納されている必要があります。

マッピング・エンジン：相互に仮想関係を持つ、異なるデータ・ストアの CI 間の関係を識別するコンポーネント。この識別は、UC MDB の調整オブジェクトと外部 CI の調整オブジェクトを調整することによって行われます。

マッピング・エンジン

Federation Framework は、マッピング・エンジンを使って FTQL を計算します。マッピング・エンジンは、異なるデータ・ストアから取得され、仮想関係によって接続された CI どうしを接続します。マッピング・エンジンは、仮想関係の調整データも提供します。仮想関係の一方のエンドは、Universal CMDB を参照する必要があります。このエンドのタイプは、調整タイプになります。2 つのサブグラフの計算では、いずれのエンド・ノードからでも仮想関係を開始できます。

FTQL アダプタ

FTQL アダプタは、外部データ・ストアから 2 種類のデータを取得するために要求されます。1 つは外部 CI データであり、もう 1 つは外部 CI に属する調整オブジェクトです。

外部 CI データ：UC MDB に存在しない外部データ。UC MDB のターゲットとなる外部データ・ストアのデータです。

調整オブジェクト・データ：Federation Framework が UC MDB CI と外部データを接続するために使用する補助データ。各調整オブジェクトは、外部 CI を参照する必要があります。調整オブジェクトのタイプは、UC MDB からデータが取得される仮想関係のいずれかのエンドのタイプ (サブタイプ) です。調整オブジェクトは、アダプタが受け取る調整データに適合する必要があります。

`IdReconciliationObject`, `PropertyReconciliationObject`, および `TopologyReconciliationObject` の 3 つのタイプがあります。

フロー図

以下の 2 つの図は、Federation Framework, Universal CMDB, アダプタ, およびマッピング・エンジンを示したものです。図例の FTQL に含まれる仮想関係は 1 つだけなので、この FTQL には Universal CMDB と 1 つの外部データ・ストアだけが関与します。最初の図では Universal CMDB で計算が開始され、2 番目の図では外部アダプタで計算が開始されます。図内の各手順では、アダプタまたはマッピング・エンジンのインタフェースの適切なメソッド呼び出しを参照しています。

HP Universal CMDB エンドで開始される計算

次のシーケンス図は、Federation Framework、Universal CMDB、アダプタ、およびマッピング・エンジン間のやり取りを示したものです。図例の FTQL に含まれる仮想関係は 1 つだけなので、この FTQL には Universal CMDB と 1 つの外部データ・ストアだけが関与します。

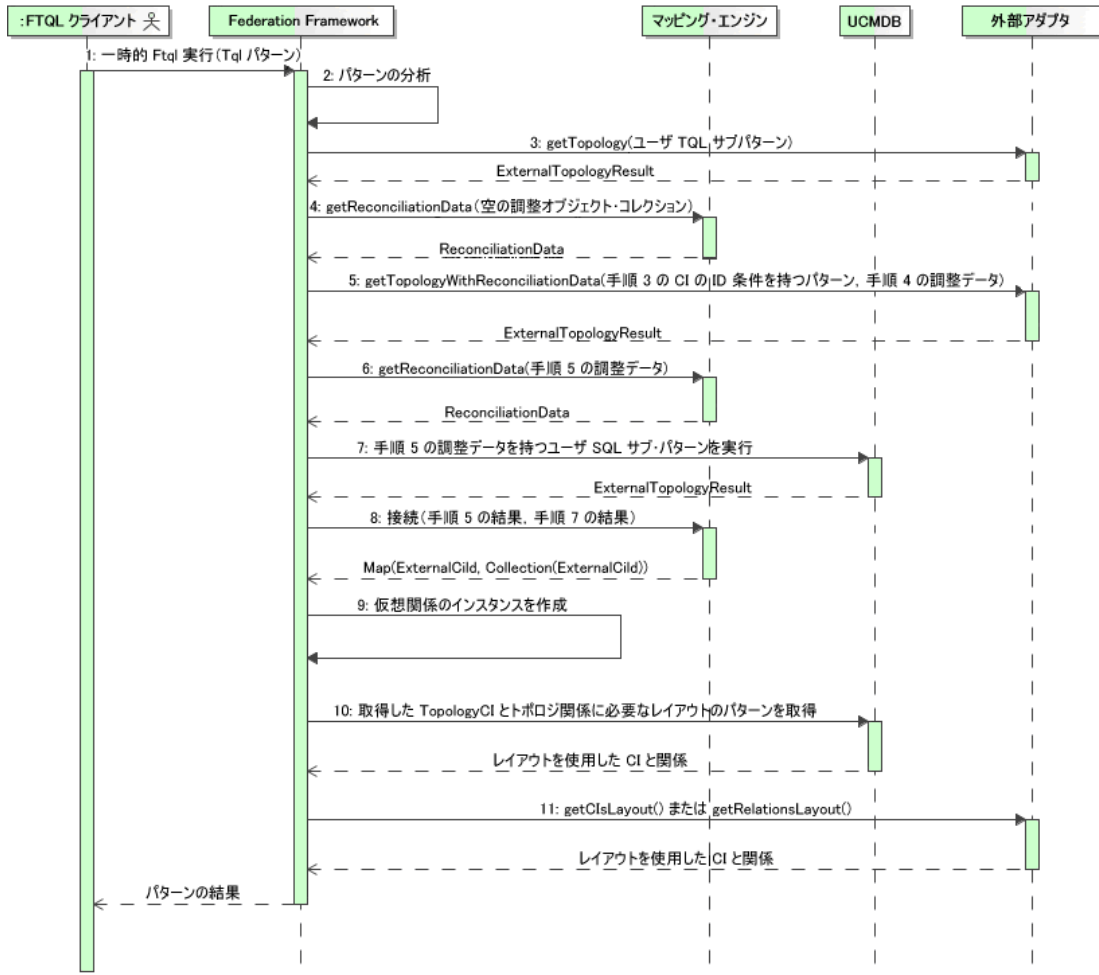


以下では、この図の各番号について説明します。

- 1 Federation Framework は、FTQL 計算の呼び出しを受け取ります。
- 2 Federation Framework はパターンを分析し、仮想関係を検出し、元の TQL を Universal CMDB 用と外部データ・ストア用の2つのサブパターンに分割します。
- 3 Federation Framework は、Universal CMDB にサブ TQL のトポロジを要求します。
- 4 Federation Framework は、トポロジ結果を受け取った後、現在の仮想関係に対する適切なマッピング・エンジンを呼び出し、調整データを要求します。この段階では `reconciliationObject` パラメータは空です。つまり、この呼び出しでは調整データに条件を追加しません。返された調整データには、UCMDB と外部データ・ストアの調整 CI どうしを照合するのに必要なデータが定義されています。調整データには、次の3種類があります。
 - ▶ **IdReconciliationData** : ID によって調整が行われます。
 - ▶ **PropertyReconciliationData** : 1つの CI のプロパティによって調整が行われます。
 - ▶ **TopologyReconciliationData** : トポロジによって調整が行われます (たとえば、ホスト CI を調整するには、IP の IP アドレスも必要です)。
- 5 Federation Framework は、手順3で受け取った仮想関係エンドの CI の調整データを Universal CMDB に要求します。
- 6 Federation Framework は、マッピング・エンジンを呼び出して調整データを取得します。この状況では (手順3と比較して)、マッピング・エンジンは手順5の調整オブジェクトをパラメータとして受け取ります。マッピング・エンジンは、受け取った調整オブジェクトを調整データに対する条件に変換します。
- 7 Federation Framework は、外部データ・ストアにサブ TQL のトポロジを要求します。外部アダプタは、手順6の調整データをパラメータとして受け取ります。
- 8 Federation Framework は、マッピング・エンジンを呼び出して、受け取った結果どうしを接続します。`firstResult` パラメータは、手順5で UCMDB から受け取った外部トポロジ結果です。`secondResult` パラメータは、手順7で外部アダプタから受け取った外部トポロジ結果です。マッピング・エンジンは、1つ目のデータ・ストア (ここでは Universal CMDB) の外部 CI ID が2つ目の (外部) データ・ストアの外部 CI ID にマップされたマップを返します。
- 9 各マッピングに対して、Federation Framework は仮想関係を作成します。

- 10 (トポロジ段階でのみ) FTQL の結果を計算した後で、Federation Framework は結果として得られた CI と関係の元の TQL レイアウトを該当するデータ・ストアから取得します。

外部アダプタ・エンドで開始される計算

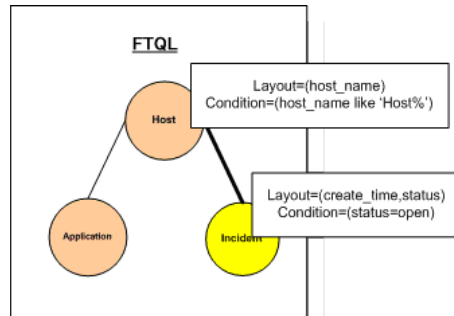


- 1 Federation Framework は、FTQL 計算の呼び出しを受け取ります。
- 2 Federation Framework はパターンを分析し、仮想関係を検出し、元の TQL を Universal CMDB 用と外部データ・ストア用の 2 つのサブパターンに分割します。

- 3 Federation Framework は、外部アダプタにサブ TQL のトポロジを要求します。要求には調整データが含まれないため、返された ExternalTopologyResult に調整オブジェクトが含まれることはありません。
- 4 Federation Framework は、トポロジ結果を受け取った後、現在の仮想関係を使って適切なマッピング・エンジンを呼び出し、調整データを要求します。この状況では、reconciliationObjects パラメータは空です。つまり、この呼び出しでは調整データに条件を追加しません。返された調整データには、UCMDB と外部データ・ストアの調整 CI どうしを照合するのに必要なデータが定義されています。調整データには、次の3種類があります。
 - ▶ IdReconciliationData : ID によって調整が行われます。
 - ▶ PropertyReconciliationData : 1 つの CI のプロパティによって調整が行われます。
 - ▶ TopologyReconciliationData : トポロジによって調整が行われます (たとえば、ホスト CI を調整するには、IP の IP アドレスも必要です)。
- 5 Federation Framework は、手順3で受け取った CI の調整オブジェクトを外部データ・ストアに要求します。Federation Framework は、外部アダプタの getTopologyWithReconciliationData() メソッドを呼び出します。要求されるトポロジは、手順3で受け取った CI を ID 条件として持ち、手順4の調整データを含む1ノード・トポロジです。
- 6 Federation Framework は、マッピング・エンジンを呼び出して調整データを取得します。この状況では(手順3と比較して)、マッピング・エンジンは手順5の調整オブジェクトをパラメータとして受け取ります。マッピング・エンジンは、受け取った調整オブジェクトを調整データに対する条件に変換します。
- 7 Federation Framework は、UCMDB に手順6の調整データを含むサブ TQL のトポロジを要求します。
- 8 Federation Framework は、マッピング・エンジンを呼び出して、受け取った結果どうしを接続します。firstResult パラメータは、手順5で外部アダプタから受け取った外部トポロジ結果です。secondResult パラメータは、手順7で UCMDB から受け取った外部トポロジ結果です。マッピング・エンジンは、1つ目のデータ・ストア(ここでは外部データ・ストア)の外部 CI ID が2つ目(UCMDB)の外部 CI ID にマップされたマップを返します。
- 9 各マッピングに対して、Federation Framework は仮想関係を作成します。
- 10 (トポロジ段階でのみ) FTQL の結果を計算した後で、Federation Framework は結果として得られた CI と関係の元の TQL レイアウトを該当するデータ・ストアから取得します。

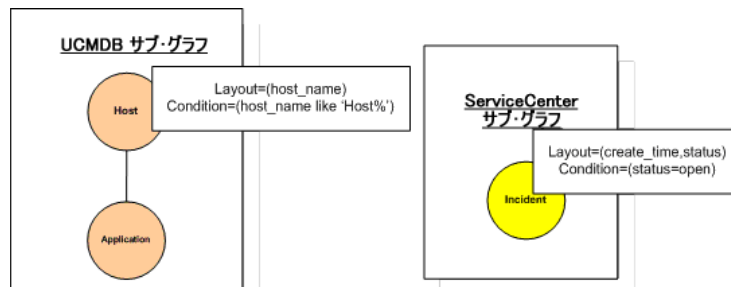
FTQL 用の Federation Framework フローの例

この例では、特定のホスト上で開いているすべてのインシデントを表示する方法について説明します。ServiceCenter データ・ストアは外部データ・ストアです。ホスト・インスタンスは Universal UCMDDB に格納され、インシデント・インスタンスは ServiceCenter に格納されています。インシデント・インスタンスを適切なホストに接続するには、ホストおよび IP の `host_name` および `ip_address` プロパティが必要であるとします。これらは、Universal UCMDDB で ServiceCenter のホストを識別する調整プロパティです。

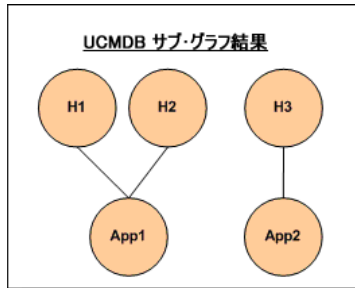


注：属性のフェデレーションのために、アダプタの `getTopology` メソッドが呼び出されます。調整データは、ユーザ TQL（この場合は CI 要素）で調整されます。

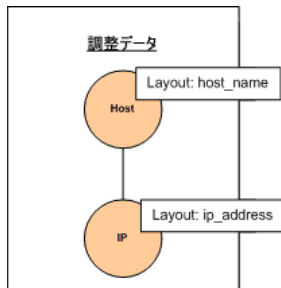
- 1 Federation Framework は、パターンを分析した後で、Host と Incident の仮想関係を認識し、FTQL を次の 2 つのサブグラフに分割します。



- 2 Federation Framework は、Universal UCMDB サブグラフを実行してトポロジを要求し、次の結果を受け取ります。

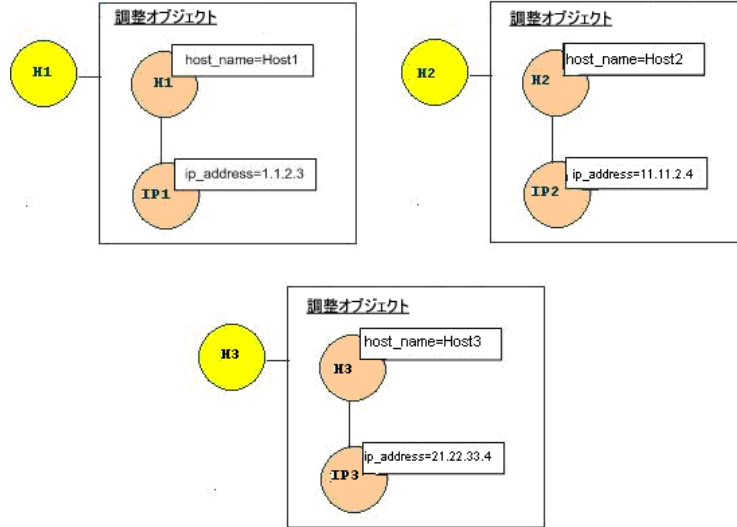


- 3 Federation Framework は、2つのデータ・ストアから受け取ったデータどうしを接続するための情報を含む1つ目のデータ・ストア（HP Universal CMDB）の調整データを、適切なマッピング・エンジンに要求します。この場合の調整データは次のとおりです。

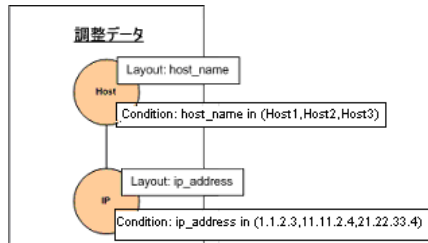


- 4 Federation Framework は、前の結果 (H1, H2, H3 の host_id) から、Host ノードとそれに対する ID 条件を含む 1 ノード・トポロジを作成し、そのクエリを HP Universal CMDB 上の必要な調整データとともに実行します。この結果には、ID 条件に関連する Host CI と、各 CI の適切な調整オブジェクトが含まれています。

ReconciliationData を使用した getTopology の結果

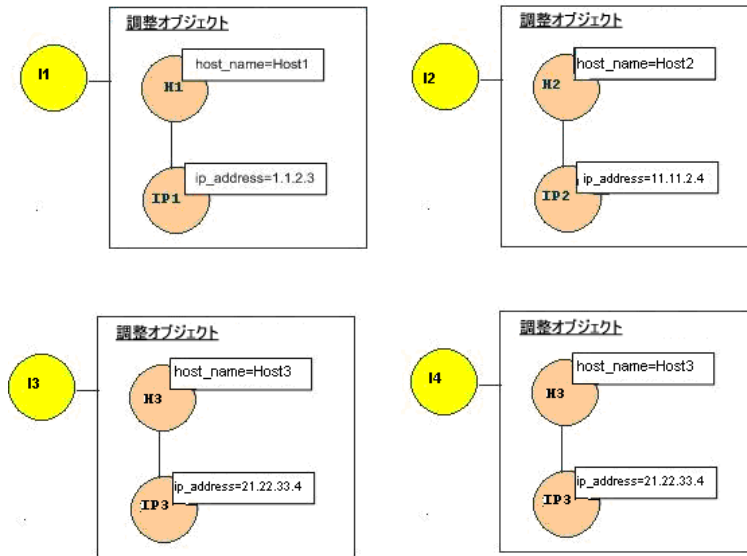


- 5 ServiceCenter の調整データには、HP Universal CMDB から受け取った調整オブジェクトから派生された host_name と ip_address の条件が含まれているはずです。

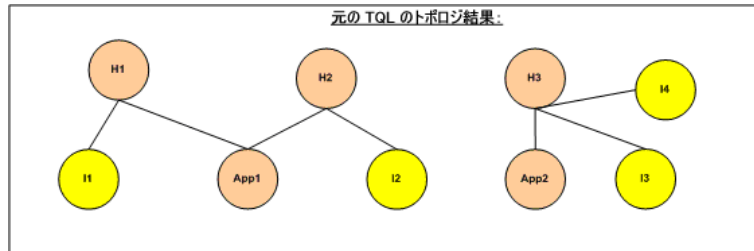


- 6 Federation Framework は、調整データとともに ServiceCenter サブグラフを実行してトポロジと適切な調整オブジェクトを要求し、次の結果を受け取ります。

調整データを持つ getTopology の ServiceCenter の結果



- 7 マッピング・エンジンでの接続と仮想関係の作成後の結果は、次のようになります。



- 8 Federation Framework は、受け取ったインスタンスの元の TQL レイアウトを HP Universal CMDB と ServiceCenter に要求します。

🔗 レプリケーション用の Federation Framework フロー

本項の内容

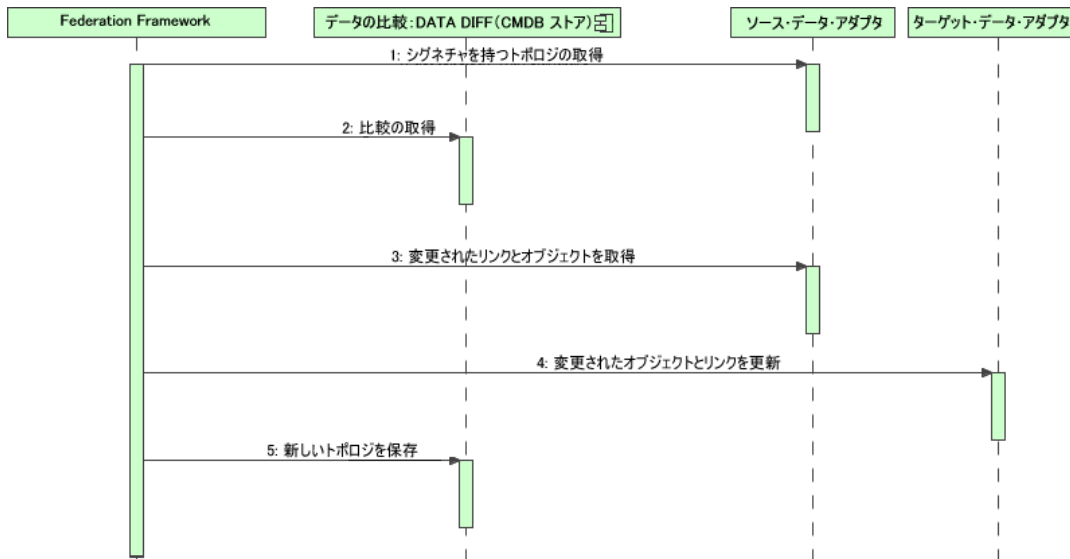
- ▶ 225 ページ「定義と用語」
- ▶ 225 ページ「フロー図」

定義と用語

シグネチャ : CI のプロパティの状態を示します。CI のプロパティ値が変更された場合は、CI シグネチャも変更される必要があります。「CI シグネチャ」によって、すべての CI プロパティを取得して比較しなくても、CI が変更されたかどうかを検出できます。CI と「CI シグネチャ」は、いずれも対応するアダプタによって提供されます。アダプタは、CI のプロパティが変更されたときに CI シグネチャを変更します。

フロー図

次のシーケンス図は、レプリケーション・フローにおける Federation Framework とソースおよびターゲット・アダプタ間のやり取りを示したものです。



- 1 Federation Framework は、ソース・アダプタからクエリ結果のトポロジを受け取ります。アダプタは、クエリをその名前で認識し、そのクエリを外部データ・ストアに対して実行します。トポロジ結果には、結果内の各 CI および関係の ID とシグネチャが含まれています。この ID は、外部データ・ストア内で CI を一意に定義する論理的 ID です。CI または関係が変更された場合は、シグネチャを変更する必要があります。
- 2 Federation Framework は、シグネチャを使って、新しく受けとったトポロジ・クエリの結果を保存されている結果と比較し、どの CI が変更されたかを特定します。
- 3 Federation Framework は、変更された CI と関係を見つけると、変更された CI と関係の ID をパラメータにしてソース・アダプタを呼び出し、それらの完全なレイアウトを取得します。
- 4 Federation Framework は、ターゲット・アダプタに更新を送信します。ターゲット・アダプタは、受信したデータを使って、それが表している外部データ・ソースを更新します。
- 5 更新後、Federation Framework は最新のクエリ結果を保存します。

HP Release Control フェデレーション・アダプタ

HP Release Control フェデレーション・アダプタは、HP Release Control からのデータ取得をサポートします。HP Release Control に対するフェデレート・クエリ計算の要求は、すべてこのアダプタを介して行われます。このアダプタは、**Planned Change CI** タイプをサポートします。クエリを作成するには、**service desk** リンクを使用します。

以下のユース・ケースでは、このアダプタの使用法について説明します。

- ▶ あるユーザは、特定期間内の**任意の CI に対する計画済みの変更**を表示する必要があります。
- ▶ あるユーザは、**特定のシステム CI に対する計画済みの変更**を表示する必要があります。

この場合、HP Universal CMDB はシステム CI を直接変更する変更を取得し、システム CI に間接的に影響を及ぼす変更は取得しません。

- ▶ あるユーザは、**特定のビジネス CI に対する計画済みの変更**を表示する必要があります。

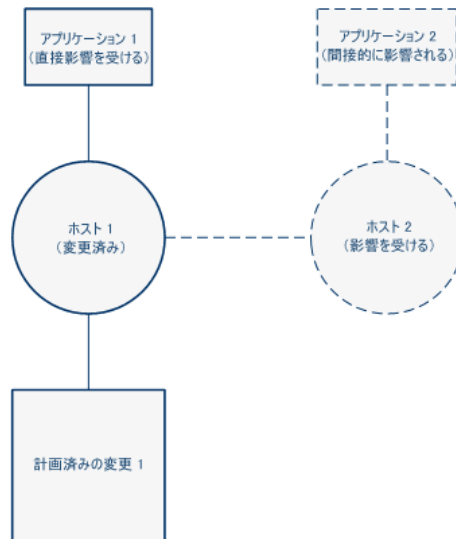
この場合、HP Universal CMDB はビジネス CI に直接影響を及ぼす変更を取得し、ビジネス CI に間接的に影響を及ぼす変更は取得しません。

前述のいずれの場合でも、HP Universal CMDB は親の変更と独立したタスクを取得します。HP Universal CMDB は親の要求に含まれるタスクを取得しません。

例

次の例は、ユース・ケースのいくつかを示したものです。HP Release Control に、**Planned Change 1** という計画済みの変更があるとします。

- ▶ **Planned Change 1** は **Host 1** で実行されます。
- ▶ **Application 1** は **Host 1** で実行されるため、変更の影響を直接受けます。
- ▶ **Host 2** は **Host 1** に接続されており、**Planned Change 1** の影響を受ける可能性があります、実際には **Host 2** に対する変更は行われません。
- ▶ **Application 2** は **Host 2** で実行され、変更の影響を間接的に受ける可能性があります。



ユーザが **Host 1** または **Application 1** に対する計画済みの変更を取得するクエリを実行すると、HP Universal CMDB に **Planned Change 1** が表示されます。この変更がそれらの CI に直接影響を与えるためです。

ユーザが **Host 2** または **Application 2** に対する計画済みの変更を取得するクエリを実行すると、これらの CI に直接影響を与える変更はないため、HP Universal CMDB に変更は表示されません。

本項の内容

- ▶ 228 ページ「フェデレーション・アダプタの設定」
- ▶ 229 ページ「計画済みの変更の属性の取得」
- ▶ 229 ページ「フェデレーション・アダプタへのカスタム・フィールドの追加」

フェデレーション・アダプタの設定

本項では、HP Universal CMDB と連携して動作するように HP Release Control フェデレーション・アダプタを設定する方法について説明します。

このアダプタを設定するには、次の手順を実行します。

- 1 パッケージ・マネージャで次のファイルをデプロイします。
 - ▶ **< Release Control のインストール・ディレクトリ > %conf%uCmdb- <バージョン番号> -extensions%federation%rc_federation.zip**
詳細については、『モデル管理』の第 14 章「パッケージ・マネージャ」を参照してください。
- 2 **< Release Control のインストール・ディレクトリ > %conf%uCmdb- <バージョン番号> -extensions%federation%CcmChangeAdapter** ディレクトリを次の場所にコピーします。
< UCMDB のインストール・ディレクトリ > %UCMDBServer%j2f%fcmdb%CodeBase%
- 3 HP Universal CMDB JMX コンソールで **CcmChangeAdapter** を再ロードします。
- 4 HP Universal CMDB で、**[設定] > [フェデレート CMDB]** に移動し、**CcmChangeAdapter** を設定します。以下の必要な情報を入力します。
 - ▶ **名前** : アダプタの論理名。
 - ▶ **ホスト** : HP Release Control の URL。
 - ▶ **ユーザ** : HP Release Control の管理者ユーザのユーザ名。
 - ▶ **パスワード** : 前述で指定した管理者ユーザのパスワード。

計画済みの変更の属性の取得

HP Universal CMDB には、[**CI Type Manager Attributes**] タブで選択した **planned change** の属性のリストが含まれています。HP Universal CMDB は、次のルールを使って HP Release Control から **planned change** の属性を取得します。

HP Universal CMDB は、属性名に含まれるすべてのアンダスコア (_) をハイフン (-) に変換し、HP Release Control のフィールドから一致するフィールドを検索します。HP Release Control のフィールドのリストは、管理者モジュールの [**フィールド**] タブにあります。

また、特定の属性プロパティは **CcmChangeAdapter** ディレクトリにある **convertfields.properties** ファイルの特定のフィールドにマップされます。追加の属性は、このファイルの HP Release Control フィールドにマップできます。

フェデレーション・アダプタへのカスタム・フィールドの追加

本項では、フェデレーション・アダプタにカスタム・フィールドを追加する方法について説明します。

フェデレーション・アダプタに新しいカスタム・フィールドを追加するには、次の手順を実行します。

- 1 HP Release Control で、管理者モジュールの [**フィールド**] タブに必要なフィールドを追加します。カスタム・フィールドの追加の詳細については、『**HP Release Control Installation and Configuration Guide**』（英語版）を参照してください。
- 2 HP Universal CMDB で、**planned change** CI タイプを見つけ、新しい属性名を追加します。
 - ▶ HP Release Control で作成したカスタム・フィールドと同じ名前を属性名として使用します。ただし、フィールド名にハイフン (-) を使用した場合は、属性名のハイフンをアンダスコア (_) に置き換えます。
 - ▶ カスタム・フィールド名とは異なる属性名を使用する場合は、その属性名を **CcmChangeAdapter** ディレクトリにある **convertfields.properties** ファイルの特定のフィールド名にマップできます。

アダプタ・インタフェース

本項の内容

- ▶ 230 ページ「定義と用語」
- ▶ 230 ページ「FTQL 用のアダプタ・インタフェース」
- ▶ 231 ページ「レプリケーション用のアダプタ・インタフェース」

定義と用語

外部関係：同じアダプタによってサポートされる2つの外部 CI タイプ間の関係。

FTQL 用のアダプタ・インタフェース

次のように、各アダプタの適切なアダプタ・インタフェースを使用します。

1 ノード・トポロジ・インタフェースは、アダプタが外部関係をサポートしない場合に使用されます。つまり、アダプタが複数の外部 CI を含む要求を受け取ることはありません。すべての1 ノード・インタフェースは、ワークフローを簡略化するために作成されます。より詳細なクエリを使用する必要がある場合は、**パターン・トポロジ・インタフェース**を使用します。

パターン・トポロジ・インタフェースは、アダプタが複数の外部 CI タイプをサポートし、サポートされている外部 CI タイプ間の関係タイプを少なくとも1 つサポートする場合に使用されます。つまり、関係を含む外部データに関するクエリを受け取る予定であるアダプタは、**パターン・トポロジ・インタフェース**を実装する必要があります。

1 ノード・インタフェース

以下のインタフェースは、それぞれ異なるタイプの調整データを持っています。

- ▶ **OneNodeTopologyIdReconciliationDataAdapter**：アダプタが**単一ノード TQL**をサポートし、データ・ストア間の調整が ID によって計算される場合に使用します。
- ▶ **OneNodeTopologyPropertyReconciliationDataAdapter**：アダプタが**単一ノード TQL**をサポートし、データ・ストア間の調整が1つの CI のプロパティによって行われる場合に使用します。
- ▶ **OneNodeTopologyDataAdapter**：アダプタが**単一ノード TQL**をサポートし、データ・ストア間の調整がトポロジによって行われる場合に使用します。

パターン・トポロジ・インタフェース

以下のインタフェースは、それぞれ異なるタイプの調整データを持っています。

- ▶ **PatternTopologyIdReconciliationDataAdapter** : アダプタが**複合 TQL**をサポートし、データ・ストア間の調整が ID によって行われる場合に使用します。
- ▶ **PatternTopologyPropertyReconciliationDataAdapter** : アダプタが**複合 TQL**をサポートし、データ・ストア間の調整が単一ノードのプロパティによって行われる場合に使用します。
- ▶ **PatternTopologyDataAdapter** : アダプタが**複合 TQL**をサポートし、データ・ストア間の調整がトポロジによって行われる場合に使用します。
- ▶ **SortResultDataAdapter** : 外部データ・ストアで結果の CI を並べ替えることができる場合に使用します。
- ▶ **FunctionalLayoutDataAdapter** : 外部データ・ストアで機能のレイアウトを計算できる場合に使用します。

レプリケーション用のアダプタ・インタフェース

- ▶ **SourceDataAdapter** : レプリケーション・ジョブのソース・アダプタに使用します。
- ▶ **TargetDataAdapter** : レプリケーション・ジョブのターゲット・アダプタに使用します。

新しい外部データ・ストア用のアダプタの追加

このタスクでは、新しい外部データ・ソースをサポートするアダプタを定義する方法について説明します。

このタスクには次の手順が含まれます。

- ▶ 232 ページ「サポートされるアダプタ・クラスを CMDB クラス・モデル内の CI および関係としてモデル化」
- ▶ 232 ページ「仮想関係に対応する有効な関係の定義」
- ▶ 233 ページ「アダプタ設定の定義」
- ▶ 235 ページ「アダプタの実装」

- ▶ 235 ページ「マッピング・エンジンの実装」
- ▶ 235 ページ「実装に必要な JAR のクラス・パスへの追加」
- ▶ 235 ページ「アダプタのデプロイ」
- ▶ 236 ページ「アダプタの再デプロイ」
- ▶ 241 ページ「ServiceCenter アダプタとマッピング・エンジンの実装の記述」

1 サポートされるアダプタ・クラスを CMDB クラス・モデル内の CI および関係としてモデル化

アダプタの開発者は、次の作業を行う必要があります。

- ▶ HP Universal CMDB の CI タイプの階層に関する知識を持ち、外部 CIT が HP Universal CMDB の CIT とどのように関連付けられるかを理解します。
- ▶ 外部 CIT を CMDB クラス・モデルでモデル化します。
- ▶ 新しい CI タイプとそれらの関係の定義を追加します。
- ▶ アダプタの内部クラス間の有効な関係に対応する、CMDB クラス・モデル内の有効な関係を定義します（これらの CIT は、CMDB クラス・モデル・ツリーの任意のレベルに配置できます）。

モデル化は、フェデレーションのタイプ（オンザフライかレプリケーションか）に関係なく同じである必要があります。CMDB クラス・モデルに新しい CIT 定義を追加する方法の詳細については、『モデル管理』の「CI タイプ・マネージャ」を参照してください。

2 仮想関係に対応する有効な関係の定義

注：本項は、FTQL アダプタのみに適用されます。

該当するデータと HP Universal CMDB データとの関係を決定します（つまり、仮想関係を定義します）。そのためには、関係の一方のエンドがアダプタによってサポートされるクラスを参照しない有効な関係を追加します。仮想関係に対応する有効な関係が通常の有効な関係と異なるのは、マッピング・エンジン・クラスの実装を定義する修飾子を持つことだけです。

有効な関係定義の例

次の有効な関係定義の例では、`it_world` タイプのインスタンスと `HistoryChange` タイプのインスタンスの間の `history_link` タイプの関係が有効です。Federation Framework は、これらのタイプのインスタンスを接続するために以下の `HistoryMappingEngine` 実装クラスを使用する必要があります。

```
<Valid-Link>
  <Class-Ref class-name="history_link" />
  <End1 class-name="it_world" />
  <End2 class-name="HistoryChange" />
  <Valid-Link-Qualifiers>
    <Valid-Link-Qualifier name="EXTENDED_VALID_LINK">
      <Data-Items>
        <Data-Item name="mapping_engine_class" type="string">
          com.mercury.topaz.adapters.CmdbHistoryAdapter.HistoryMappingEngine
        </Data-Item>
      </Data-Items>
    </Valid-Link-Qualifier>
  </Valid-Link-Qualifiers>
</Valid-Link>
```

すべての仮想関係に対して 1 つのマッピング・エンジン実装しかない場合は、有効な関係の修飾子としてではなく、アダプタ設定でそれを定義できます。

3 アダプタ設定の定義

以下の詳細情報を含む XML アダプタ設定ファイルを追加します。

- ▶ **アダプタ ID** : アダプタの一意の ID。
- ▶ **アダプタ名** : アダプタの完全修飾 Java 実装クラス名。
- ▶ **アダプタ機能** : アダプタがサポートする機能を定義します。詳細については、241 ページ「アダプタ機能」を参照してください。
- ▶ **接続するフィールド** : 外部データ・ストアに接続するためにユーザが指定する必要があるフィールドを定義します。

- ▶ **標準設定のマッピング・エンジン**：仮想関係用の標準設定のマッピング・エンジンを定義します。FTQL をサポートするアダプタにのみ適用されます。アダプタ設定のスキーマについては、スキーマ・フォルダを参照してください。

アダプタ設定定義の例

アダプタ設定定義の例を次に示します。

```
<adapter-config adapter-id="CmdbRmiAdapter">
  <class-name>com.mercury.topaz.adapters.cmdb.CmdbRmiAdapter</class-
name>
  <adapter-capabilities>
    <support-federated-query>
      <supported-classes>
        <supported-class is-derived="true" all-attributes-
supported="true" name="hostresource"/>
      </supported-classes>
      <topology>
        <pattern-topology>
          <functional-layout/>
        </pattern-topology>
        <advanced-capabilities>
          <calculated-attribute></calculated-attribute>
        </advanced-capabilities>
      </topology>
      <result>
        <sort-result/>
      </result>
    </support-federated-query>
    <support-replicatioin-data>
      <source/>
      <target/>
    </support-replicatioin-data>
  </adapter-capabilities>
  <fields-to-connect>
    <field>host</field>
    <field>customerId</field>
  </fields-to-connect>
  <default-mapping-
engine>com.mercury.topaz.adapters.cmdb.CmdbMappingEngine</default-mapping-
engine>
</adapter-config>
```

4 アダプタの実装

アダプタに定義された機能に従って、正しいアダプタ実装クラスを選択します。アダプタ実装クラスは、定義された機能に従って適切なインタフェースを実装します。

5 マッピング・エンジンの実装

アダプタがフェデレート・クエリをサポートする場合は、仮想関係ごとのマッピング・エンジン実装をサポートする必要があります。実装クラスは、マッピング・エンジン・インタフェースを実装する必要があります。

6 実装に必要な JAR のクラス・パスへの追加

クラスを実装するには、クラス・パスに次の JAR を追加します。

- ▶ federation_api.jar

7 アダプタのデプロイ

- a アダプタ・パッケージをデプロイします。パッケージのデプロイの詳細については、『**モデル管理**』の「パッケージのデプロイ」を参照してください。

パッケージには、以下のエンティティが含まれている必要があります。

- ▶ 新しい CIT の定義（任意選択）：

アダプタが UCMDB にまだ存在しない新しい CI タイプをサポートする場合にのみ使用されます。

新しい CIT の定義は、パッケージ内の **class** フォルダにあります。

- ▶ 新しいデータ型の定義（任意選択）：

新しい CIT に新しいデータ型が必要な場合にのみ使用されます。

新しいデータ型の定義は、パッケージ内の **typedef** フォルダにあります。

- ▶ 新しい有効な関係の定義（任意選択）：

アダプタがフェデレート TQL をサポートする場合にのみ使用されます。

新しい有効な関係の定義は、パッケージ内の **validlinks** フォルダにあります。

仮想関係は、有効な関係として定義され、マッピング・エンジンの完全修飾クラス名を定義する **mapping_engine_class** という名前のデータ項目を含むことができます。

- ▶ アダプタ設定定義のXMLファイルは、パッケージ内の **adapter** フォルダに置く必要があります。
 - ▶ **記述子**：パッケージ定義を定義します。
- b** 次のようにしてコードをデプロイします。
- ▶ 必要なすべてのアダプタ・インタフェースを実装するクラスを作成します。
 - ▶ マッピング・エンジンを実装します（フェデレート・クエリ・アダプタを記述している場合）。
 - ▶ コンパイルしたクラス（通常はjarファイル）を、ベースとなるすべての *jar ファイルとともに **< HP Universal CMDB のルート・ディレクトリ > ¥UCMDBServer¥j2f¥fcmdb¥CodeBase¥ <アダプタ ID >** フォルダに置きます。

注：<アダプタ ID >フォルダの名前は、アダプタ設定の値と同じです。

- ▶ 独自の設定ファイルを作成した場合は、**< HP Universal CMDB のルート・ディレクトリ > ¥UCMDBServer¥j2f¥fcmdb¥CodeBase¥ <アダプタ ID >** フォルダをルート・フォルダとしても使用する必要があります。

8 アダプタの再デプロイ

アダプタの定義と実装は、外部クラスの定義、アダプタ機能の変更、または実装の変更を行った結果、変わる可能性があります。その場合は、定義または実装を再デプロイします。

- a** アダプタ・パッケージを再デプロイします。

外部クラスの定義またはアダプタの定義が変更された場合は、更新したアダプタ・パッケージを作成し、パッケージ・メカニズムを使ってそれを再デプロイします。詳細については、『**モデル管理**』の「**パッケージのデプロイ**」を参照してください。

- b** コードを再デプロイします。

実装コードまたは専用の設定が変更された場合は、以下を実行します。

- ▶ 更新した *.jar または設定ファイルを作成し、それらを **< HP Universal CMDB のルート・ディレクトリ >** `UCMDBServer\j2f\fcmdb\CodeBase` **<アダプタ ID >** フォルダに置きます。
- ▶ 適切なカスタマ ID およびアダプタ ID とともに、次の JMX メソッドを呼び出します。
[FCmdb Config Services] > [loadOrReloadCodeBaseForAdapterId]

新しいアダプタを追加する例

この例では、ServiceCenter 用のアダプタを追加する方法について説明します。

- 1 関連する ServiceCenter のエンティティは、Incident、Problem、および RFC です。エンティティごとに CIT 定義を作成します。これらの定義を HP Universal CMDB のクラス・モデル階層のどこに置くかを決定します。これらのエンティティは IT プロセスを参照するため、IT Process CIT の下に置くのが妥当です。RFC は IT 変更を参照するため、IT Change CIT の下に RFC クラスの定義を置きます。Incident、Problem、および RFC の各 CIT 間には関係は存在しないため、このような関係の定義は追加しません。
- 2 HP Universal CMDB 内のどの CIT を ServiceCenter のエンティティに関連付けるかを決定します。この例では、関連するエンティティは Host だけです。[Host, Incident]、[Host, Problem]、[Host, RFC] の各ペアの有効な関係を追加します。個々の有効な関係には、その関係のエンド間の接続をサポートするマッピング・エンジンの実装クラス名を含む修飾子を付ける必要があります。3 つのケースまたは（実装によっては）3 つのクラスのいずれにも、同じ Java クラスを使用できます。
- 3 アダプタ設定の XML ファイルを定義します。
 - a アダプタ ID を定義します。

この例では、ID が ServiceCenterAdapter で、クラス実装が adapter.ServiceCenterAdapterImpl であるとします。アダプタ設定ファイルの冒頭部分は、次のようになります。

```
<adapter-config adapter-id=" ServiceCenterAdapter ">
  <class-name> adapter.ServiceCenterAdapterImpl </class-name>
```

b アダプタ機能を定義します。

- ▶ この例では、アダプタはフェデレーション・クエリをサポートしますが、レプリケーション・データをサポートしません。したがって、その機能のXMLには **support-federated-query** 要素を含めますが、**support-replication-data** は含めません。
- ▶ サポートされるクラスを、サポートされる属性条件とともに追加します。3つのクラスのすべてに **status** プロパティがあり、システムがこの属性の等価条件のみをサポートするとします。**supported_classes** 要素は以下のように定義されます。

```

<supported-classes>
  <supported-class is-derived="true" all-attributes-supported="false"
name="incident">
    <supported-conditions>
      <attribute-operators attribute-name="status">
        <operator> EQUEL </operator>
      </attribute-operators>
    </supported-conditions>
  </supported-class>
  <supported-class is-derived="true" all-attributes-supported="false"
name="change">
    <supported-conditions>
      <attribute-operators attribute-name="status">
        <operator> EQUEL </operator>
      </attribute-operators>
    </supported-conditions>
  </supported-class>
  <supported-class is-derived="true" all-attributes-supported="false"
name="problem">
    <supported-conditions>
      <attribute-operators attribute-name="status">
        <operator> EQUEL </operator>
      </attribute-operators>
    </supported-conditions>
  </supported-class>
</supported-classes>

```

- ▶ **ServiceCenter** アダプタがサポートするトポロジを定義します。
ServiceCenter のエンティティ間には関係が存在しないため、**ServiceCenter** アダプタは単一ノード・トポロジをサポートするだけで十分です。話を簡単にするため、**ServiceCenter** アダプタは高度な機能を備えておらず、計算される属性の値を計算できないとします。**topology** 要素の定義は次のとおりです。

```
<topology>
  <one-node-topology/>
</topology>
```

- ▶ 話を簡単にするため、**ServiceCenter** は並べ替えをサポートしないとします。このため、機能には **result** を含めません。
- c **ServiceCenter** アダプタが **ServiceCenter** に接続するのに必要な情報を定義する要素を追加します。これには、ホスト、ポート、およびユーザが含まれるとします。**ServiceCenter** 用のパスワードは空なので、このフィールドは要求事項として定義しません。ただし、**ServiceCenter** アダプタはこのフィールドを確認し、値が存在する場合は、そのパスワード値を使って接続します。**fields-to-connect** 要素は以下のように定義されます。

```
<fields-to-connect>
  <field>host</field>
  <field>port</field>
  <field>user</field>
</fields-to-connect>
```

Service Desk アダプタの設定の XML 定義が完成しました。完成した定義を以下に示します。

```

<adapter-config adapter-id=" ServiceCenterAdapter ">
  <class-name> adapter.ServiceCenterAdapterImpl </class-name>
  <adapter-capabilities>
    <support-federated-query>
      <supported-classes>
        <supported-class is-derived="true" all-attributes-
supported="false" name="incident">
          <supported-conditions>
            <attribute-operators attribute-name="status">
              <operator> EQUAL </operator>
            </attribute-operators>
          </supported-conditions>
        </supported-class>
        <supported-class is-derived="true" all-attributes-
supported="false" name="change">
          <supported-conditions>
            <attribute-operators attribute-name="status">
              <operator>EQUEL</operator>
            </attribute-operators>
          </supported-conditions>
        </supported-class>
        <supported-class is-derived="true" all-attributes-
supported="false" name="problem">
          <supported-conditions>
            <attribute-operators attribute-name="status">
              <operator>EQUEL</operator>
            </attribute-operators>
          </supported-conditions>
        </supported-class>
      </supported-classes>
      <topology>
        <one-node-topolgy/>
      </topology>
    </<support-federated-query>
  </adapter-capabilities>
  <fields-to-connect>
    <field>host</field>
    <field>port</field>
    <field>user</field>
  </fields-to-connect>
</adapter-config>

```


9 ServiceCenter アダプタとマッピング・エンジンの実装の記述

マッピング・エンジンの実装は、MappingEngine インタフェースを実装する必要があります。ServiceCenter アダプタは、その機能と、調整がトポロジによって行われる点に基づいて、OneNodeTopologyDataAdapter インタフェースを実装する必要があります。

アダプタ機能

本項では、アダプタ機能について説明します。

- ▶ **フェデレート・クエリのサポート**：アダプタ実装が FTQL をサポートする場合に、アダプタ機能に含まれます。フェデレート・クエリには以下の機能が含まれます。
 - ▶ **サポートされるクラス**：フェデレート・クエリでサポートされるクラスを定義します。サポートされるクラスの定義には、サポートされるクラスの名前、このクラスの派生クラスもサポートされるかどうか、およびクラス属性の可能な条件演算が含まれます。派生クラスがサポートされる場合は、クラス属性の条件演算も派生されます。アダプタ機能設定に含まれるサポートされるクラスの定義を無視するには、FTqlDataAdapter インタフェースの getSupportedClasses メソッドを実装します。これは、サポートされるクラスまたはサポートされる属性条件演算子が動的に変更される可能性がある場合に便利です。
 - ▶ **トポロジ**：パターン・トポロジと単一ノード・トポロジのどちらをサポートするか、およびフェデレート TQL の高度な機能を定義します。
 - ▶ **結果**：アダプタが結果の CI を並べ替えることができるかどうかを定義します。
- ▶ **レプリケーション・データのサポート**：アダプタ実装がデータ・レプリケーションをサポートする場合に、アダプタ機能に含まれます。レプリケーション機能には、以下の機能が含まれます。
- ▶ **ソース**：アダプタをレプリケーション・ジョブのソースとして使用できることを定義します。
- ▶ **ターゲット**：アダプタをレプリケーション・ジョブのターゲットとして使用できることを定義します。

第 8 章

HP ServiceCenter/Service Manager Adapter

本章では、HP ServiceCenter/Service Manager Adapter バージョン 1.0 に関する情報を提供します。このアダプタは、HP Universal CMDB バージョン 7.0 以降、HP ServiceCenter バージョン 6.2、および HP Service Manager バージョン 7.0 (WSDL 構成の変更以降) に対応しています。

注： このアダプタは、ServiceDesk Adapter の特定構成です。

本章の内容

概念

- ▶ アダプタの使用方法 (244 ページ)
- ▶ アダプタ構成ファイル (245 ページ)

タスク

- ▶ アダプタのデプロイ (254 ページ)
- ▶ ServiceDesk アダプタのデプロイ (255 ページ)
- ▶ ServiceCenter/Service Manager CIT への属性の追加 (262 ページ)

🔗 アダプタの使用手法

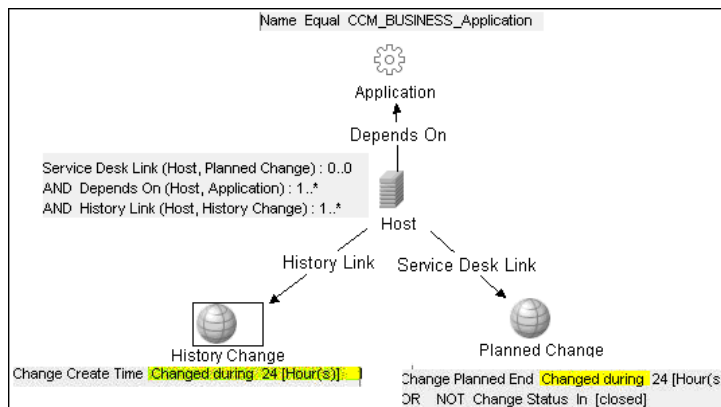
ServiceCenter/Service Manager Adapter は、HP ServiceCenter および HP Service Manager からのデータの取得をサポートしています。このアダプタは、Web サービス API を使って ServiceCenter/Service Manager に接続し、それらからデータを取得します。ServiceCenter/Service Manager に対するフェデレート・クエリの要求はすべて、このアダプタを通じて行われます。

このアダプタは、3つの外部 CI タイプ、Incident、Problem、および Planned Change をサポートしています。アダプタは、ServiceCenter/Service Manager から、指定されたフィルタによって（調整および/または CI フィルタを使って）要求されたレイアウトで、これらのタイプの CI を取得します。これらの CIT はそれぞれ、UCMDB の内部 CIT の 1 つ（Host、Business Service、Application）に関連付けることができます。各 UCMDB 内部 CIT は、ServiceCenter/Service Manager 構成の中に動的に変化する調整ルールを含んでいます（詳細については、249 ページ「調整データの構成」を参照してください）。ただし、アダプタがサポートする CIT 間に内部関係はありません。

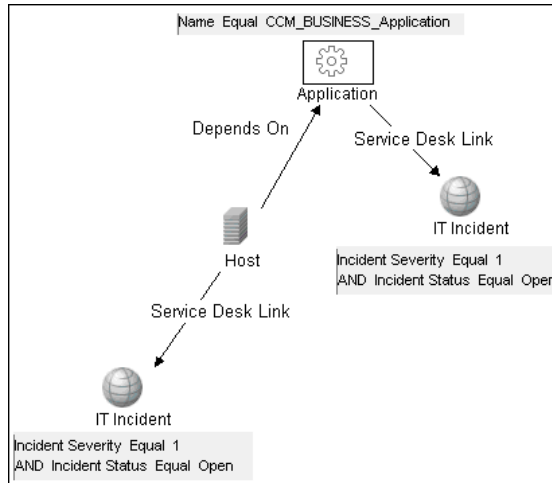
サポートされている CIT と仮想関係のモデリングは、アダプタとともに提供されます。CIT に属性を追加できます（詳細については、262 ページ「ServiceCenter/Service Manager CIT への属性の追加」を参照してください）。

次の使用例（TQL の例を含む）は、アダプタをどのように使用できるかを示しています。

- 1 特定のアプリケーションを実行しているすべてのホストに対して過去 24 時間以内に加えられた未計画の変更をすべて表示する必要がある場合



- 2 アプリケーションとそのホストにおいて開いている重要なインシデントをすべて表示する必要がある場合



アダプタ構成ファイル

アダプタ構成ファイル **serviceDeskConfiguration.xml** は、次のディレクトリにあります。

< HP Universal CMDB のルート・ディレクトリ > ¥UCMDBServer¥2f¥fcmdb¥CodeBase¥ServiceDeskAdapter

このファイルには、次の 3 つの部分があります。

- 1 **ucmdbClassConfigurations** 要素によって定義される最初の部分には、アダプタがサポートする外部 CIT 構成が含まれています。詳細については、246 ページ「外部 CIT の構成」を参照してください。
- 2 **reconciliationClassConfigurations** 要素によって定義される 2 番目の部分には、適切な UCMDB CIT に関する調整データ情報が含まれています。詳細については、249 ページ「調整データの構成」を参照してください。
- 3 **globalConnectorConfig** 要素によって定義される 3 番目の部分には、特定のコネクタ実装のグローバル構成が含まれています。詳細については、253 ページ「グローバル構成」を参照してください。

重要：アダプタは、標準の構成ファイルなしで配布されます。データ・ストアを定義する前に、使用する ServiceCenter/Service Manager のバージョンに応じた適切なファイルを準備する必要があります。

- ▶ **...¥fcmdb¥CodeBase¥ServiceDeskAdapter** フォルダを見つけます。このフォルダには、次の 3 つの構成ファイルがあります。
 - serviceDeskConfiguration.xml.6.xx (ServiceCenter バージョン 6.xx 用)
 - serviceDeskConfiguration.xml.7.0x (Service Manager バージョン 7.0x 用)
 - serviceDeskConfiguration.xml.7.1x (Service Manager バージョン 7.1x 用)
 - ▶ 該当する構成ファイルのサフィックスを削除します。たとえば、Service Manager 7.0x を使用している場合は、**serviceDeskConfiguration.xml.7.0x** ファイルを見つけて **.7.0x** を削除し、ファイルの名前を **serviceDeskConfiguration.xml** にします。
-

外部 CIT の構成

アダプタがサポートする各 CIT は、アダプタ構成ファイルの最初のセクションで定義されています。

この `ucmdbClassConfiguration` セクションは、サポートされている CIT 構成だけを表します。この要素は、UCMDB クラス・モデル (`ucmdbClassName` 属性) で定義された CIT 名、そのすべての属性のマッピング (`attributeMappings` 要素)、および特定のコネクタ実装のプライベート構成 (`classConnectorConfiguration` 要素) を含んでいます。

- ▶ `ucmdbClassName` 属性は、UCMDB クラス・モデル名を定義します。
- ▶ `attributeMappings` 要素は、`attributeMapping` 要素を含んでいます。

`attributeMapping` 要素は、UCMDB モデル属性名 (`ucmdbAttributeName` 属性) と、対応する ServiceCenter/Service Manager 属性名 (`serviceDeskAttributeName` 属性) とのマッピングを定義します。

次に例を示します。

```
<attributeMapping ucmdbAttributeName="problem_brief_description"
serviceDeskAttributeName="brief.description"/>
```

この要素には、オプションで次のコンバータ属性を含めることもできます。

- ▶ **converterClassName** 属性：これは、UCMDB 属性値を ServiceDesk 属性値に変換するコンバータ・クラス名です。
- ▶ **reversedConverterClassName** 属性：これは、ServiceDesk 属性値を UCMDB 属性値に変換するコンバータ・クラス名です。
- ▶ **classConnectorConfiguration** 要素には、現在の外部 CIT に関する特定のコネクタ実装の構成が含まれています。この構成に特殊な XML 文字（たとえば、& と置き換えられる **&** など）が含まれている場合は、この構成を CDATA でラップしてください。

Service Manager の **classConnectorConfiguration** 要素では、次のフィールドが役に立ちます。

- ▶ **device_key_property_names** 要素には、現在のオブジェクトの WSDL 情報内にある、デバイス ID が収められている可能性があるフィールドの名前（たとえば **ConfigurationItem**）が含まれています。各フィールドは、**device_key_property_name** 要素として追加しなければなりません。
- ▶ **id_property_name** 要素には、WSDL 情報内にある、現在のオブジェクトの ID が収められているフィールドの名前が含まれています。

次の例は、`serviceDeskConfiguration.xml` ファイルの `ucmdbClassConfiguration` セクションを示しています。このセクションには、`ServiceCenter` コネクタが実装されたインシデント CIT の `ucmdbClassName` 要素が含まれています。

```
<ucmdbClassConfiguration ucmdbClassName="it_incident">
  <attributeMappings>
    <attributeMapping ucmdbAttributeName="incident_id"
serviceDeskAttributeName="IncidentID"/>
    <attributeMapping ucmdbAttributeName="incident_brief_description"
serviceDeskAttributeName="BriefDescription"/>
    <attributeMapping ucmdbAttributeName="incident_category"
serviceDeskAttributeName="Category"/>
    <attributeMapping ucmdbAttributeName="incident_severity"
serviceDeskAttributeName="Severity"/>
    <attributeMapping ucmdbAttributeName="incident_open_time"
serviceDeskAttributeName="OpenTime"/>
    <attributeMapping ucmdbAttributeName="incident_update_time"
serviceDeskAttributeName="UpdateTime"/>
    <attributeMapping ucmdbAttributeName="incident_close_time"
serviceDeskAttributeName="ClosedTime"/>
    <attributeMapping ucmdbAttributeName="incident_status"
serviceDeskAttributeName="IMTicketStatus"/>
  </attributeMappings>
  <classConnectorConfiguration>
    <![CDATA[ <class_configuration
connector_class_name="com.mercury.topaz.fcmbd.adapters.serviceDeskAdapter.servi
ceCenterConnector.impl.SimpleServiceCenterObjectConnector">
<device_key_property_names>
<device_key_property_name>ConfigurationItem</device_key_property_name>
</device_key_property_names>
<id_property_name>IncidentID</id_property_name>
<keys_action_info>
<request_name>RetrieveIncidentKeysListRequest</request_name>
<response_name>RetrieveIncidentKeysListResponse</response_name>
</keys_action_info>
```



```

<properties_action_info>
  <request_name>RetrieveIncidentListRequest</request_name>
  <response_name>RetrieveIncidentListResponse</response_name>
</properties_action_info>
</class_configuration> ]]>
  </classConnectorConfiguration>
</ucmdbClassConfiguration>

```

CIT への属性の追加

アダプタがサポートしている CIT について UCMDB モデルに属性を追加するときには、次の手順を実行します。

- 1 **serviceDeskConfiguration.xml** ファイルで、該当する **ucmdbClassConfiguration** 要素に **attributeMapping** 要素を追加します。
- 2 ServiceCenter/Service Manager がこの属性を自身の Web サービス API で具体化していることを確認します。
- 3 **serviceDeskConfiguration.xml** を保存します。
- 4 アダプタ **FCmdb Config Services > loadOrReloadCodeBaseForAdapterId** を再ロードするための呼び出しを、対応するカスタマ ID と **ServiceDeskAdapter** アダプタ ID を使って JMX に送信します。

調整データの構成

アダプタがサポートする CIT に関連付けることのできる各 UCMDB CIT は、アダプタ構成ファイルの 2 番目のセクションで定義されます。

この `reconciliationClassConfigurations` セクションは、1 つの UCMDB CIT の調整データ構成を表します。この要素には、次の 2 つの属性が含まれます。

- ▶ `ucmdbClassName` 属性：これは、UCMDB クラス・モデルで定義された CIT 名です。
- ▶ `concreteMappingImplementationClass` 属性：これは、`ConcreteMappingEngine` インタフェースの具体的な実装のクラス名です。この属性を使って、UCMDB CIT のインスタンスと外部アダプタ CIT の間のマッピングを行います。使用される標準設定の実装は次のとおりです。

```
com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.mapping.impl.OneNodeMappingEngine
```

ホストの IP による整合のためのホスト調整 CIT に対してのみ使用される別の実装も存在します。

```
com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.mapping.impl.HostIpMappingEngine
```

`reconciliationClassConfiguration` 要素には、次の要素の 1 つを含めることができます。

- ▶ `reconciliationById` 要素：この要素は、ID による調整が行われるときに使用されます。この場合、この要素のテキスト値は、CMDB ID が収められる ServiceDesk フィールドの名前です。次に例を示します。

```
<reconciliationById>SerialNumber</reconciliationById>
```

この例では、ServiceDesk フィールド `SerialNumber` に、適切なホストの CMDB ID が含まれています。

- ▶ `reconciliationData` 要素：属性を比較することによって調整が行われる場合は、この要素を使用します。調整は、1 つの属性で実行することも、論理演算子 OR、AND、またはその両方を使って複数の属性で実行することもできます。

1 つの属性で調整を実行する場合、`reconciliationData` 子要素は `reconciliationAttribute` 要素でなければなりません。`reconciliationAttribute` 要素には、適切な UCMDB 属性名 (`ucmdbAttributeName` 属性) と適切な ServiceDesk 属性名 (`serviceDeskAttributeName` 属性) が含まれています。この要素には、適切な UCMDB CIT 名を定義する `ucmdbClassName` 属性も含めることができます。標準設定では、現在の調整 UCMDB CIT 名が使用されます。

また、`converterClassName` 属性と `reversedConverterClassName` 属性を使用することもできます。これらの属性は、UCMDB 属性値を `ServiceDesk` 属性値に、またはその逆に変換するコンバータ・クラスの名前を含んでいなければなりません。

次に例を示します。

```
<reconciliationData>
  <reconciliationAttribute ucmdbAttributeName="host_hostname"
  serviceDeskAttributeName="NetworkName"
  converterClassName="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.converter.PropertyValueConverterToUpperCase"/>
</reconciliationData>
```

複数の属性について調整を実行するには、論理演算子を使って調整属性を組み合わせます。

論理演算子 AND は、複数の `reconciliationAttribute` 要素を含むことができます (最低 2 個)。このケースでは、調整ルールは属性比較間に 1 個の AND 演算子を含んでいます。

次に例を示します。

```
<reconciliationData>
<AND>
  <reconciliationAttribute ucmdbAttributeName="host_hostname"
  serviceDeskAttributeName="NetworkName"
  converterClassName="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.converter.PropertyValueConverterToUpperCase"/>
  <reconciliationAttribute ucmdbClassName="ip"
  ucmdbAttributeName="ip_address" serviceDeskAttributeName="NetworkAddress" />
</AND>
</reconciliationData>
```

この例では、調整ルールは次の形式に従っています。

`host.host_hostname= NetworkName and ip.ip_address= NetworkAddress`

論理演算子 OR は、複数の `reconciliationAttribute` および AND 要素を含むことができます。このケースでは、調整ルールは属性および AND 式の間には 1 つの OR 演算子を含んでいます。XML は要素の順序を保証しないので、OR 要素タイプの各サブ要素に `priority` 属性を指定する必要があります。OR 式の間には比較は、指定された優先度に従って計算されます。

次に例を示します。

```
<reconciliationData>
<OR>
  <reconciliationAttribute ucmdbAttributeName="host_dnsname"
serviceDeskAttributeName="NetworkDNSName" priority="2" />
<AND priority="1" >
  <reconciliationAttribute ucmdbAttributeName="host_hostname"
serviceDeskAttributeName="NetworkName"
converterClassName="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.converter.PropertyValueConverterToUpperCase"/>
  <reconciliationAttribute ucmdbClassName="ip"
ucmdbAttributeName="ip_address" serviceDeskAttributeName="NetworkAddress" />
</AND>
</OR>
</reconciliationData>
```

この例では、調整ルールは次の形式に従っています
(host.host_dnsname= NetworkDNSName OR (host.host_hostname=
NetworkName and ip.ip_address= NetworkAddress))。AND 要素の priority 属性
値は 1 なので、最初に (host.host_hostname= NetworkName and ip.ip_address=
NetworkAddress) 条件がチェックされます。条件が満たされると、調整が実行さ
れます。条件が満たされない場合は、host.host_dnsname= NetworkDNSName
条件がチェックされます。

reconciliationClassConfiguration の追加のサブ要素は
classConnectorConfiguration です。classConnectorConfiguration 要素には、現
在の調整 CIT に関する特定のコネクタ実装の構成が含まれています。この構成
に特殊な XML 文字（たとえば、& と置き換えられる & など）が含まれて
いる場合は、この構成を CDATA でラップしてください。

CIT の調整ルールの変更

- 1 **serviceDeskConfiguration.xml** で、適切な reconciliationData 要素を新しい
ルールで更新します。
- 2 アダプタ **FCmdb Config Services > loadOrReloadCodeBaseForAdapterId**
を再ロードするために、対応するカスタマ ID と ServiceDeskAdapter アダプタ
ID を使って JMX を呼び出します。

ip_address または host_name によるホストの調整

ip_address または host_name を使ってホストの調整を実行するには、次の ReconciliationData 要素をアダプタ構成ファイルに含めます。

```

<reconciliationData>
  <OR>
    <reconciliationAttribute priority="1" ucmdbClassName="ip"
ucmdbAttributeName="ip_address" serviceDeskAttributeName="NetworkAddress"/>
    <reconciliationAttribute priority="2" ucmdbClassName="host"
ucmdbAttributeName="host_hostname" serviceDeskAttributeName="NetworkName"
converterClassName="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.conver
ter.PropertyValueConverterToUpperCase"/>
  </OR>
</reconciliationData>

```

reconciliationClassConfiguration 要素の concreteMappingImplementationClass 属性の値を次のように変更する必要があります。

```

="com.mercury.topaz.fcmdb.adapters.serviceDeskAdapter.mapping.impl.HostIpMappingEngine"

```

グローバル構成

アダプタ構成ファイルの 3 番目のセクションには、特定のコネクタ実装のグローバル構成が含まれています。

この globalConnectorConfig 構成に特殊な XML 文字（たとえば、& と置き換えられる & など）が含まれている場合は、この構成を CDATA でラップしてください。

Service Manager の globalConnectorConfig 要素では、次のフィールドが役に立ちます。

- 1 **date_pattern** 要素には、Service Manager が使用する日付パターンが含まれています。

標準設定は MM/dd/yy HH:mm:ss です。

日付パターンが誤っている場合、FTQL は間違った日付条件結果を返します。

- 2 **time_zone** 要素は、Service Manager のタイム・ゾーンを定義します。標準設定は、UCMDB サーバのタイム・ゾーンです。

Service Manager の日付パターンとタイム・ゾーンをチェックするには、次の手順を実行します。

- a **Service Manager バージョン 7** : [メニューナビゲーション] > [システム管理] > [ベースシステム設定] > [その他] > [システム情報レコード] にアクセスします。[日付情報] タブをクリックします。
 - b **ServiceCenter バージョン 6.1** : [メニューナビゲーション] > [ユーティリティ] > [アドミニストレーション] > [情報] > [システム情報レコード] にアクセスします。[日付情報] タブをクリックします。
- 3 **max_query_length** 要素は、Service Manager Web サービス要求における最大クエリ長を定義します。標準設定値は **1000000** です。
 - 4 **name_space_uri** 要素は、Service Manager Web サービスに接続するための名前空間 URI を定義します。標準設定値は **http://servicecenter.peregrine.com/PWS** です。
 - 5 **web_service_suffix** 要素は、Service Manager Web サービス・センターの URI サフィックスを定義します。標準設定値は **sc62server/ws** です。この値は、URL が作成されるときに使用されます。

アダプタのデプロイ

本項では、典型的なアダプタのデプロイメントについて説明します。

本項の内容

- 1 255 ページ 「ServiceDesk アダプタのデプロイ」
 - a 255 ページ 「アダプタ実装ファイルを抽出してパッケージをデプロイする」
 - b 256 ページ 「ServiceCenter/Service Manager 外部データ・ソースを追加する」
 - c 256 ページ 「HP ServiceCenter 6.2 の設定」 (HP ServiceCenter に接続する場合)
 - d 259 ページ 「HP Service Manager 7.0 の設定」 (HP Service Manager に接続する場合)

- 2 262 ページ「ServiceCenter/Service Manager CIT への属性の追加」
 - a 262 ページ「HP Universal CMDB モデルに属性を追加する」
 - b 263 ページ「構成を変更することによって、HP ServiceCenter から属性をエクスポートする」（HP ServiceCenter に接続する場合）
 - c 266 ページ「構成を変更することによって、Service Manager から属性をエクスポートする」（HP Service Manager に接続する場合）
 - d 269 ページ「アダプタ構成ファイルの変更」
 - e 269 ページ「変更のロード」

ServiceDesk アダプタのデプロイ

本項では、デプロイメントに必要なファイルをどこに置くべきかを説明します。

本項には、次の手順が含まれています。

- ▶ 255 ページ「アダプタ実装ファイルを抽出してパッケージをデプロイする」
- ▶ 256 ページ「ServiceCenter/Service Manager 外部データ・ソースを追加する」
- ▶ 256 ページ「HP ServiceCenter 6.2 の設定」
- ▶ 259 ページ「HP Service Manager 7.0 の設定」

1 アダプタ実装ファイルを抽出してパッケージをデプロイする

- a 次のフォルダとファイルの場所を確認します。
 - ▶ ServiceDeskAdapter
 - ▶ serviceDeskAdapter.zip
- b **serviceDeskAdapter.zip** パッケージを <HP Universal CMDB のルート・ディレクトリ>¥UCMDBServer¥root¥lib¥packages ディレクトリに移動します。
- c **serviceDeskAdapter.zip** パッケージをデプロイします。HP Universal CMDB にログインし、パッケージ・マネージャにアクセスします（[管理] > [設定] > [パッケージ マネージャ]）。パッケージを選択し、[デプロイ] ボタンをクリックします。

パッケージのデプロイの詳細については、『**モデル管理**』の「パッケージのデプロイ」を参照してください。

d ServiceDeskAdapter フォルダを次のディレクトリに移動します。

**< HP Universal CMDB のルート・ディレクトリ > ¥UCMDBServer¥j2f¥
fcmdb¥CodeBase**

2 ServiceCenter/Service Manager 外部データ・ソースを追加する

このステップでは、外部データ・ソースを追加します。

a HP Universal CMDB で、[フェデレート CMDB] ウィンドウにアクセスします ([**管理**] > [**設定**] > [**フェデレート CMDB**])。



b 左のボタンをクリックし、データ・ストアを追加します。表示された [データストア] ダイアログ・ボックスで、**ServiceDeskAdapter** を選択して必須フィールドに情報を入力します。

このダイアログ・ボックスの詳細については、116 ページ「[データストア] タブ」を参照してください。

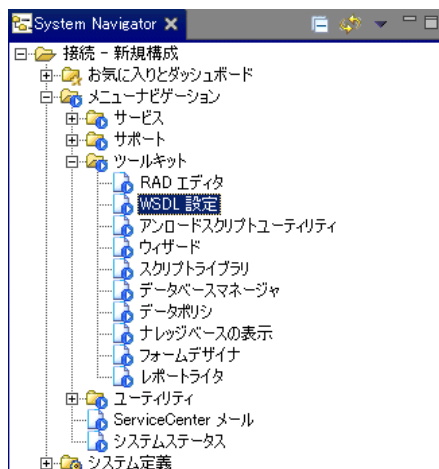
c 256 ページ「HP ServiceCenter 6.2 の設定」または 259 ページ「HP Service Manager 7.0 の設定」に進みます。

3 HP ServiceCenter 6.2 の設定

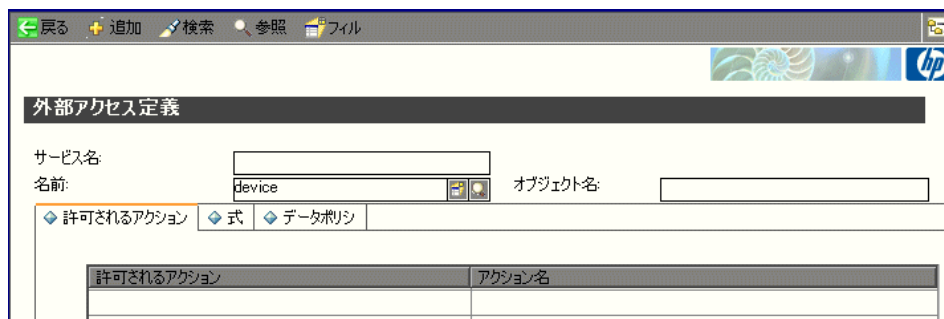
HP ServiceCenter に接続する場合は、次の手順を実行します。

a HP ServiceCenter を開き、ServiceCenter クライアントを開きます。

- b ナビゲータで [WSDL 設定] を表示します ([メインメニュー] > [メニューナビゲーション] > [ツールキット])。



- c [名前] フィールドに **device** と入力し, **Enter** キーを押します。



- d [データポリシ] タブを選択し、**network.name** 属性が空でないことを確認します（その値は **NetworkName** になっているはずです）。値を **false** に変更します。変更を保存します。

サービス名: ConfigurationManagement
 名前: device オブジェクト名: Device

◆ 許可されるアクション ◆ 式 ◆ データポリシ

フィールド名	API キャプション	除外	API データ型
logical.name.vj2		true	
logical.name.vj2.alias		true	
logical.name.vj3		true	
logical.name.vj3.alias		true	
logical.name.vj4		true	
mac.address		true	
manufacturer		true	
model	Model	false	
mtbf		true	
network.address		true	
network.name	NetworkName	false	
nm.id		true	
nondevice		true	
objid		true	
operator.name		true	

- e 保存したら、[キャンセル] ボタンをクリックします。
- f [オブジェクト名] フィールドに **Change** と入力し、**Enter** キーを押します。
- g [データポリシ] タブを選択し、次のことを確認します。
- ▶ **header.coordinator** 属性が空ではない（その値は **Coordinator** になっているはずです）。値を **false** に変更します。

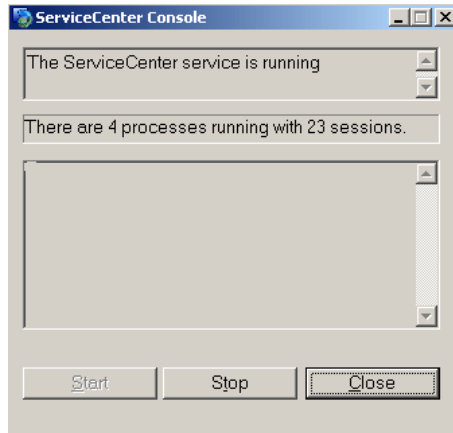
サービス名: ChangeManagement
 名前: cm3r オブジェクト名: Change

◆ 許可されるアクション ◆ 式 ◆ データポリシ

フィールド名	API キャプション	除外	API データ型
header.coord.date		true	
header.coord.dept		true	
header.coord.phone	CoordinatorPhone	false	
header.coordinator	Coordinator	false	
header.coreq.changes		true	

- ▶ **header.orig.operator** 属性が空ではない（その値は **OpenedBy** になっているはずです）。値を **false** に変更します。
- h 変更を保存します。

- i ServiceCenter を再起動します。[スタート] > [プログラム] > [ServiceCenter 6.2] > [Server] > [Console] を選択し、ServiceCenter コンソールを開きます。

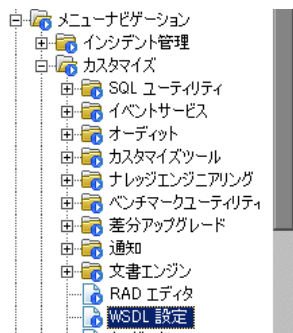


- j [Stop] をクリックし、その後 [Start] をクリックします。
- k 262 ページ「HP Universal CMDB モデルに属性を追加する」に進みます。

4 HP Service Manager 7.0 の設定

HP Service Manager に接続する場合は、次の手順を実行します。

- a HP Service Manager クライアントを開きます。
- b ナビゲータで [WSDL 設定] を表示します ([メインメニュー] > [メニューナビゲーション] > [カスタマイズ])。



- c [オブジェクト名] フィールドに **Device** と入力し，ENTER キーを押します。

外部アクセス定義

サービス名:

名前: オブジェクト名:

◆ 許可されるアクション ◆ 式 ◆ フィールド

許可されるアクション	アクション名	アクションタイプ

- d [フィールド] タブを選択し，[フィールド] リストに **network.name** 属性が含まれていて，そのキャプションが **NetworkName** であることを確認します。この属性が [フィールド] リストの中にある場合は，この属性を追加して変更を保存してください。

The screenshot shows the HP ServiceCenter/Service Manager interface. The main window displays an incident form with the following details:

- Incident Title: this s my first fed
- Alert Status: updated
- Category: security
- Subcategory: virus infection

The bottom pane shows the XML representation of the incident data. A **Find/Replace** dialog box is open over the XML, with the following settings:

- Find: myclass
- Replace With: (empty)
- Direction: Forward
- Scope: All
- Options: Case Sensitive, Wrap Search, Whole Word, Incremental, Regular expressions (all unchecked)

The XML content in the bottom pane includes the following lines:

```
<class sctype="string">myclass</class>
<alternate.contact sctype="string" NullValue="1" />
<site.visit.date sctype="dateTime" NullValue="1" />
<site.visit.technician sctype="string" NullValue="1" />
<operating.system sctype="string" NullValue="1" />
<os.release.level sctype="string" NullValue="1" />
<os.maint.level sctype="string" NullValue="1" />
<manufacturer sctype="string" NullValue="1" />
```

- e 保存したら，[キャンセル] ボタンをクリックします。
- f [オブジェクト名] フィールドに **Change** と入力し，**Enter** キーを押します。
- g [フィールド] タブを選択し，次のことを確認します。
- ▶ [フィールド] リストに **header,coordinator** 属性が含まれていて，そのキャプションが **Coordinator** である。この属性が [フィールド] リストの中にある場合は，この属性を追加します。

- ▶ [フィールド] リストに **header,orig.operator** 属性が含まれていて、そのキャプションが **OpenedBy** である。この属性が [フィールド] リストの中にある場合は、この属性を追加します。

h 変更を保存します。

- i 問題のフェデレーションをサポートするために、ナビゲータで [WSDL 設定] を表示し ([メインメニュー] > [メニューナビゲーション] > [カスタマイズ]), [オブジェクト名] フィールドに **Problem** と入力します。
ENTER キーを押します。

外部アクセス定義

サービス名:

名前: オブジェクト名:

◆ 許可されるアクション ◆ 式 ◆ フィールド

許可されるアクション	アクション名	アクションタイプ

- j [フィールド] タブをクリックし、以下のフィールドを追加して変更を保存します。

フィールド	キャプション	タイプ
id	ID	
brief.description	BriefDescription	
status	Status	
expected.resolution.time	ExpectedResolutionTime	DateTimeType
category	Category	
initial.impact	InitialImpact	
severity	Severity	
priority.code	PriorityCode	
assignment	Assignment	

フィールド	キャプション	タイプ
logical.name	CI	
affected.ci(ci.device.name)	CiDeviceName	

k 262 ページ「HP Universal CMDB モデルに属性を追加する」に進みます。

ServiceCenter/Service Manager CIT への属性の追加

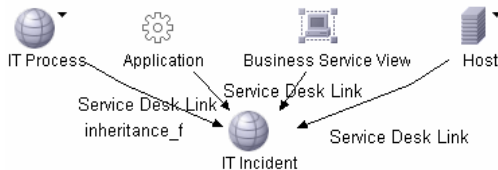
本項では、属性を追加することによって、ServiceCenter/Service Manager から追加のデータを取得する方法を説明します。

本項には、次の手順が含まれています。

- ▶ 262 ページ「HP Universal CMDB モデルに属性を追加する」
- ▶ 263 ページ「構成を変更することによって、HP ServiceCenter から属性をエクスポートする」
- ▶ 266 ページ「構成を変更することによって、Service Manager から属性をエクスポートする」
- ▶ 269 ページ「アダプタ構成ファイルの変更」
- ▶ 269 ページ「変更のロード」

1 HP Universal CMDB モデルに属性を追加する

- a HP Universal CMDB に新しい属性を追加します。インシデント CIT を編集するために、[管理] > [モデリング] > [CI タイプ マネージャ] を選択します。ビュー・エクスプローラで、[IT Process] > [IT Incident] を選択します。



b [属性] タブを選択し、新しい属性を追加します。

属性名: incident_class
表示名: incident_class
詳細:

属性タイプ:
 プリミティブ 列挙/リスト
string

値のサイズ: 250
標準設定値:

詳細
 インデックス 小文字 必須
 可視 編集可能 パスワード
 変更をモニタ 比較可能 アセット データ

OK キャンセル

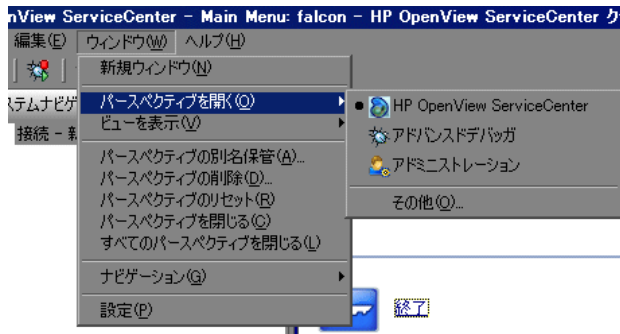
c 263 ページ「構成を変更することによって、HP ServiceCenter から属性をエクスポートする」または 266 ページ「構成を変更することによって、Service Manager から属性をエクスポートする」に進みます。

2 構成を変更することによって、HP ServiceCenter から属性をエクスポートする

HP ServiceCenter に接続する場合は、次の手順を実行します。

a HP ServiceCenter で、ServiceCenter クライアントを開きます。

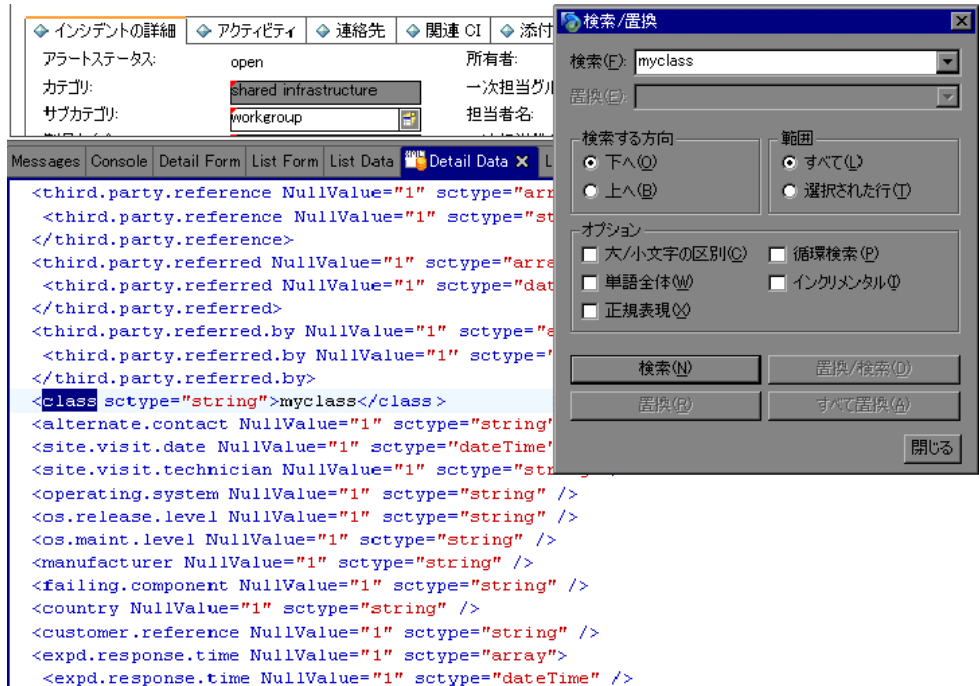
- b [ウィンドウ] > [パースペクティブを開く] > [アドミニストレーション] を選択します。



- c [インシデント管理] > [All Open Incidents] を選択し、作成したインシデントの 1 つを選択します。

注： [Class] フィールドの値が、HP Universal CMDB に報告する値であることを確認します。

- d 下に表示された XML ファイルの中で、[Class] フィールドに入力した値（つまり **myclass**）を検索します。これは、ServiceCenter 内での CI 名です。

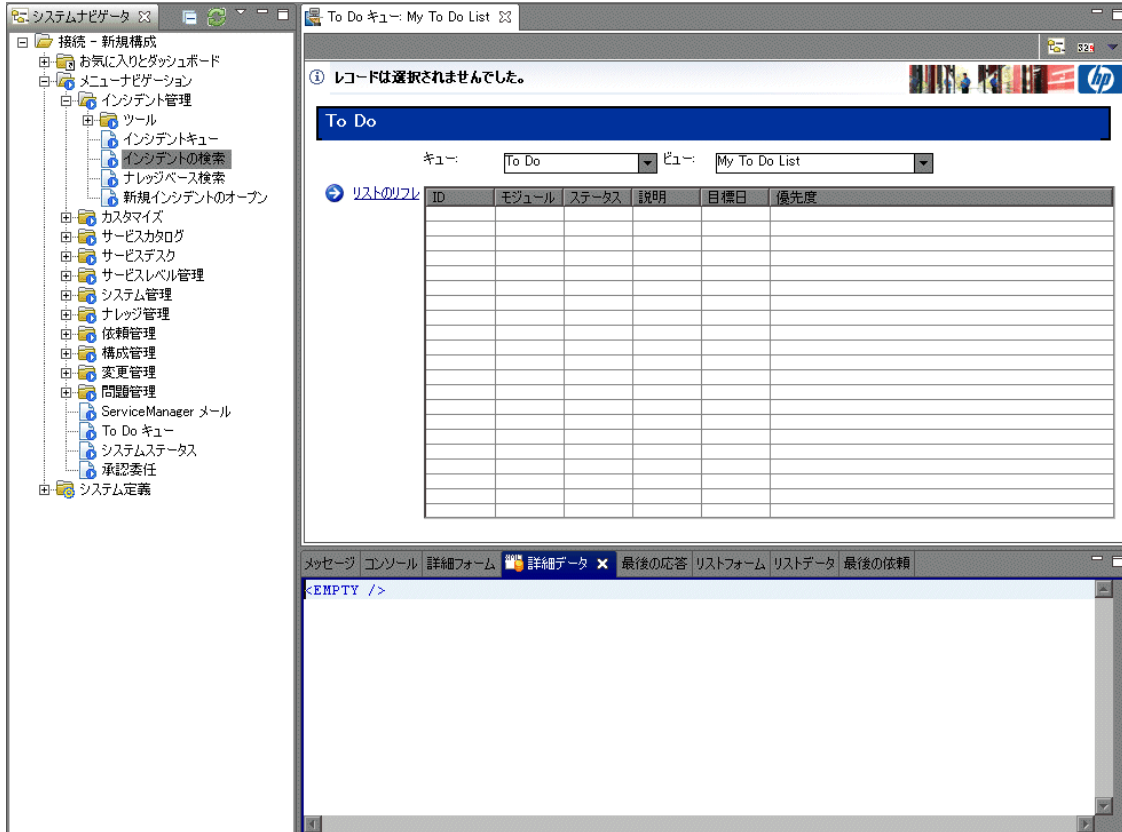


- e ナビゲータで [WSDL 設定] を表示します ([メインメニュー] > [メニューナビゲーション] > [ツールキット])。[オブジェクト名] フィールドを見つけて **Incident** と入力し、**Enter** キーを押します。
- f [データポリシー] タブを選択します。XML ファイル内に記述されている CI の名前（つまり **class**）を入力します。値を **false** に変更します。変更を保存します。
- g ServiceCenter を再起動します。[スタート] > [プログラム] > [ServiceCenter 6.2] > [Server] > [Console] を選択し、ServiceCenter コンソールを開きます。
- h [Stop] をクリックし、その後 [Start] をクリックします。
- i 269 ページ「アダプタ構成ファイルの変更」に進みます。

3 構成を変更することによって、Service Manager から属性をエクスポートする

HP Service Manager に接続する場合は、次の手順を実行します。

- a HP Service Manager クライアントで、**[復元]** ボタンをクリックして右下の表示枠を復元します。**[詳細データ]** タブをクリックします。



- b 作成したインシデントの 1 つを開きます。[インシデント管理] > [インシデントの検索] を選択します。検索ボタンをクリックします（検索対象を狭めるためにフィールドを絞り込むことができます）。

The screenshot displays the HP ServiceCenter/Service Manager interface. At the top, there is a table listing incidents with columns for Incident ID, Open Date/Time, Last Update Time, System Time, Update Time, Start Time, SLA Alert Type, Resolution Time, Close Time, and Alert Type. The incident IM1001 is highlighted.

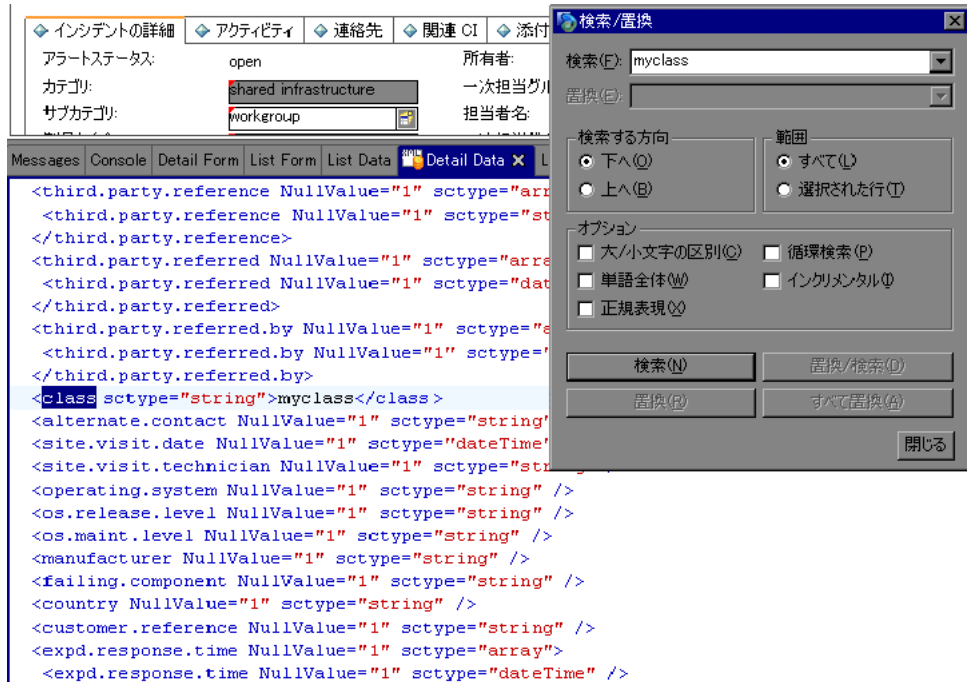
Below the table, the detailed view for incident IM1001 is shown. The incident title is "Phone is going dead intermittently." and the ticket status is "Open". The interface includes various tabs and fields for incident details, such as "Alert Status", "Category", "Sub-category", "Product type", "Problem type", "Manufacturer", "Class", and "Resolution time". There are also dropdown menus for "Primary group", "Secondary group", "Priority", and "Severity".

At the bottom, the console output shows the XML query used to retrieve the incident details:

```
<model name="probsummary" query="true">
</keys>
<number sctype="string">IM1001</number>
</keys>
<instance recordid="IM1001 - Phone is going dead intermittently." uniquequery="number="IM1001":>
<number type="string">IM1001</number>
<number.vj type="string">IM1001</number.vj>
<number.vj.alerts type="string">IM1001</number.vj.alerts>
<vj.number.1 type="string">IM1001</vj.number.1>
<vj.number.2 type="string">IM1001</vj.number.2>
<vj.number.3 type="string">IM1001</vj.number.3>
<vj.number.4 type="string">IM1001</vj.number.4>
<vj.number.5 type="string">IM1001</vj.number.5>
```

注： [Class] フィールドの値が、HP Universal CMDB に報告する値であることを確認します。

- c 下に表示された XML ファイルの中で、[Class] フィールドに入力した値（つまり **myclass**）を検索します。これは、Service Manager 内での CI 名です。



- d ナビゲータで [WSDL 設定] を表示します ([メインメニュー] > [メニューナビゲーション] > [カスタマイズ])。[オブジェクト名] フィールドを見つけて **Incident** と入力し、ENTER キーを押します。
- e [データ ポリシ] タブを選択します。
- f [フィールド] タブを選択し、XML ファイル内に記述されている CI の名前（つまり **class**）が [Fields] リストに含まれていて、そのキャプションが **ClassName** であることを確認します。この属性が [Field] リストの中になければ、この属性を追加して変更を保存してください。
- g HP Service Manager 7.00 Server サービスを再起動します。
- h 269 ページ「アダプタ構成ファイルの変更」に進みます。

4 アダプタ構成ファイルの変更

この手順はすべての構成について実行します。

- a 次の場所にある **ServiceDeskConfiguration.xml** ファイルを編集します。

```
< HP Universal CMDB のルート・ディレクトリ> ¥UCMDBServer¥j2f¥
fcmdb¥CodeBase¥ServiceDeskAdapter
```

- b 新しい属性行を [Incident] 領域の下に追加します。次のマーカを見つけてください。

```
<ucmdbClassConfiguration ucmdbClassName="it_incident">
<attributeMappings>
```

- c 次の行を追加します。

```
<attributeMapping ucmdbAttributeName="incident_class"
ServiceDeskAttributeName="ClassName"/>
```

説明：

- ▶ **ucmdbAttributeName="incident_class"** は、CI タイプ・マネージャで定義された値です。
- ▶ **ServiceDeskAttributeName="ClassName"** は、ServiceCenter/Service Manager で定義された値です。

- d 269 ページ「変更のロード」に進みます。

5 変更のロード

変更をロードするために、次の手順を実行します。

- a Web ブラウザを起動して次のアドレスを入力します。

```
http:// <マシン名または IP アドレス> :8080/jmx-console
```

<マシン名または IP アドレス>には、HP Universal CMDB がインストールされているマシンを指定します。

管理者のユーザ名とパスワードを使ってログインする必要がある場合があります。

- b [Topaz> service=Fcmdb Config Services] リンクをクリックします。

c [JMX MBEAN View] ページで次の演算を見つけます。

loadOrReloadCodeBaseForAdapterId()

d [customerID] フィールドに **1** を入力します。[AdapterId] フィールドに、アダプタ・フォルダの名前 (ServiceDeskAdapter) を入力します。[Invoke] をクリックします。

第 9 章

トラブルシューティングと制限事項

本章には、フェデレート CMDB 機能に関するトラブルシューティングと制限事項が記載されています。

本章の内容

参照先

- ▶ フェデレート CMDB のトラブルシューティングと制限事項 (271 ページ)

フェデレート CMDB のトラブルシューティングと制限事項

本項の内容

- ▶ 272 ページ「すべてのアダプタ」
- ▶ 272 ページ「RMI アダプタ」
- ▶ 272 ページ「CmdbChanges アダプタ」

すべてのアダプタ

- ▶ ビュー・マネージャで変更を行い、これらの変更が TQL の結果に影響を与えると、ビュー内のフェデレート CI は更新されません。これは、フェデレート TQL はその場で計算されるだけで、ビューの再計算時には更新されないためです。フェデレート CI を更新するには、ビュー・エクスプローラでビューを選択し、[ビューを再構築] ボタンをクリックします（再計算には長時間かかる場合があります）。詳細については、『モデル管理』の「参照モード」を参照してください。
- ▶ RMI アダプタがフェデレート・クエリもサポートしている場合は、ローカル UCMDB データ・ストアを設定するときに、[CITs Supported by Adapter] ダイアログ・ボックスで CIT を選択しないでください（ローカル UCMDB データ・ストア用のアダプタは、レプリケーション・ジョブに使用する場合にのみ追加してください）。
- ▶ 外部データ・ストアによってサポートされる CIT のインスタンスがローカル UCMDB 内に存在する場合、その CIT は選択しないでください。もし選択すると、状態の矛盾が発生する場合があります。たとえば、CPU CIT のインスタンスがローカル UCMDB の中に存在する場合、外部データストアを定義するときは、たとえ選択したアダプタが CPU をサポートしていても、CPU を選択してはなりません。
- ▶ 2 つの CMDB の間でレプリケーション・ジョブを設定する場合は、両方の CMDB のクラス・モデルが同じであることを確認してください。

RMI アダプタ

- ▶ RMI アダプタを使用する場合は、設定されたホストで実行される CMDB のバージョンが、UCMDB のバージョンと同じでなければなりません。

CmdbChanges アダプタ

- ▶ このアダプタは、Root ノードに対する root_class IN ... 条件以外のプロパティ条件はサポートしていません。
- ▶ クエリには、**Root** というラベルが付いた CI が 1 つ、または **Root** というラベルが付いた関係が 1 つ以上含まれていなければなりません。

ルート・ノードは、同期されるメイン CI で、その他のノードはメイン CI に含まれている CI です。たとえば、各ホストを同期させるときには、ホスト・ノードがルートとしてラベル付けされ、ホストのリソースはルートになりません。

- ▶ TQL グラフはサイクルを含んでいてはなりません。
- ▶ TQL は、Root CI とその CI に直接接続されたオプションの CI だけで構成されていなければなりません。
- ▶ 関係を同期するために使用されるクエリは、カーディナリティが 1...* で、関係間の OR 条件を含んでいなければなりません。
- ▶ このアダプタは複合関係をサポートしません。
- ▶ 複製される CI 属性はすべて、履歴データベースにそれらを書き込めるようにするために、[変更をモニタ] チェック・ボックス (STATIC 修飾子) が選択されていなければなりません。

属性の編集

属性名: host_os

表示名: Host Operating System

詳細: Operating System Model

属性タイプ:

プリミティブ 列挙/リスト

string

値のサイズ: 500

標準設定値:

詳細

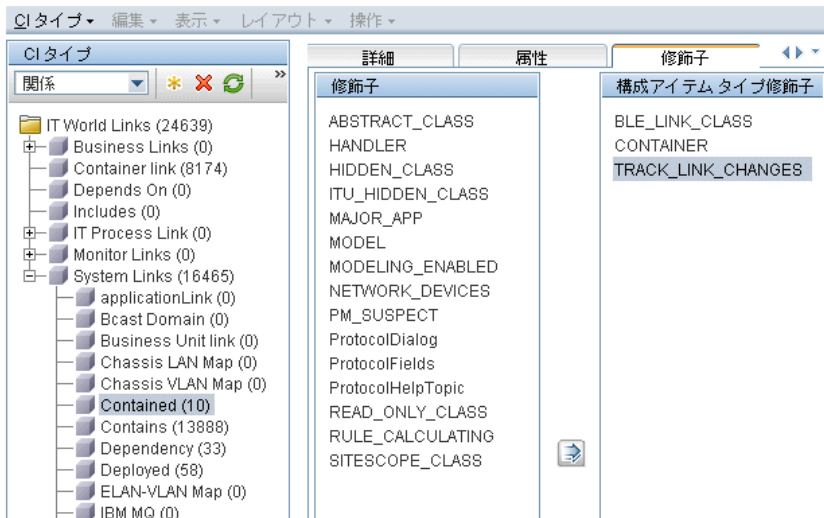
インデックス 小文字 必須

可視 編集可能 パスワード

変更をモニタ 比較可能 アセットデータ

OK キャンセル

- ▶ 関係は、履歴データベースにそれらを書き込めるようにするための **TRACK_LINK_CHANGES** 修飾子を含んでいなければなりません。



第 10 章

調整の概要

本章では、調整に関する情報を提供します。

本章の内容

概念

- ▶ 調整 — 概要 (275 ページ)
- ▶ ホスト調整ルール (275 ページ)
- ▶ クラスタ調整ルール (276 ページ)
- ▶ ソフトウェア要素調整ルール (277 ページ)

調整 — 概要

調整は、異なるデータ・ストア (DDM, DDMi, チェッキングなど) からのエンティティを識別して照合するプロセスです。この機能は、CI の重複を避けるように設計されています。

以降の各項では、設定済みのすぐに使える調整ルールについて詳しく説明します。

ホスト調整ルール

- ▶ UCMDB 内のホストはどれも、IP アドレスか strong ID の値 (MAC アドレスやハードウェア ID など) によって識別されます。

IP アドレスによって識別されたホストは、incomplete (不完全) (Host Is Complete 属性の値が false) とみなされます。

strong ID によって識別されたホストは、**complete**（完全）とみなされます。1 つのホストが複数の MAC アドレスを持っている場合、DDM は最も小さい MAC アドレスを識別子として使用します。

- ▶ **incomplete** ホストは、その IP アドレスによって **complete** ホストと調整させたり、同じ ID を持つ **incomplete** ホストと調整させることができます。
- ▶ **complete** ホストは、その IP アドレスによって **incomplete** ホストと調整させたり、同じ ID を持つ **complete** ホストと調整させることができます。
- ▶ CMDB 内に **incomplete** ホストが存在し、**complete** ホストが報告された場合、その **complete** ホストで **incomplete** ホストが置き換えられ、**incomplete** ホストに接続されていたトポロジ（関連 CI）は新しい **complete** ホストへ移されます。
- ▶ CMDB 内に **complete** ホストが存在し、**incomplete** ホストが報告された場合には、その **incomplete** ホストのプロパティが、CMDB 内にすでに存在する **complete** ホストにコピーされます。

注：プロパティは、競合解決ポリシーに従って更新されます。

クラスタ調整ルール

- ▶ クラスタ・サーバは **complete** ホストです。
- ▶ 1 つの **incomplete** ホストが 2 つの **complete** ホストとともに IP によって識別され、その **complete** ホストの 1 つがクラスタである場合、その **incomplete** ホストはクラスタとともにのみ識別されます。
- ▶ クラスタ・サーバと **complete** ホストが同じ VIP（仮想 IP）に接続されている場合、**application_ip** 属性にその VIP を含むソフトウェア要素はすべて、物理的な **complete** ホストからクラスタ・サーバへと移されます。

ソフトウェア要素調整ルール

- ▶ ソフトウェア要素 CIT は、強いタイプまたは弱いタイプとみなされます。
 - ▶ **強いタイプのソフトウェア要素 CIT** : これらの CIT は特殊化された CIT であり (ソフトウェア要素 CIT から派生します), より多くの属性, さまざまな識別ルール, 表示ラベルなどを含んでいます。
 - ▶ **弱いタイプのソフトウェア要素 CIT** : これらの CIT は, ソフトウェア要素 CI タイプ自体の CI インスタンスです。これらの CIT は, 基本構成属性だけを含みます。また, モデル内には各ソフトウェア・コンポーネント・タイプの1つのインスタンスだけが存在できます。つまり, 単一のホスト上で弱いタイプのソフトウェア要素を使って2つの異なる Oracle インスタンスをモデリングすることはできません。
- ▶ 強いタイプのソフトウェア要素が検出されると, 調整メカニズムによって, 弱いタイプのソフトウェア要素が対応する強いタイプにマッピングされます。このメカニズムにより, DDM が1つのメソッドによってソフトウェアの存在を検出し, その後別のメソッドを使ってそのソフトウェアが検出されたときに, 2つの CI が最終的に UC MDB 内でマージされることを知って, そのソフトウェアに関する追加の詳細を報告することが可能になります。
- ▶ 調整メカニズムは, 弱いタイプと強いタイプの共有属性である **Name** 属性 (**data_name**) の値に依存しています。すべてのソフトウェア要素 (強いタイプも弱いタイプも) は, 識別用の名前を付けて作成されます。弱いソフトウェア要素を含んでいるホストで, 同じ名前を持つ強いソフトウェア要素が検出された場合, 弱いソフトウェア要素は強いソフトウェア要素にマージされます。

第 III 部

インテグレーション

第 11 章

ダイレクト・リンクを使った UCMDB アプレットの埋め込み

本章では、ダイレクト・リンクを使って UCMDB アプレットを外部アプリケーションに埋め込む方法を説明します。

注：本章は、インテグレーション開発者のみを対象としています。

本章の内容

概念

- ▶ ダイレクト・リンクを使った UCMDB アプレットの埋め込み (282 ページ)
- ▶ UCMDB アプレット・タブー 概要 (283 ページ)

参照先

- ▶ ダイレクト・リンクの操作フロー (284 ページ)

ダイレクト・リンクを使った UCMDB アプレットの埋め込み

ダイレクト・リンクを使って、UCMDB アプレットを外部アプリケーションに直接埋め込むことができます。ダイレクト・リンクは、ダイレクト・リンク・ウィザードを使って作成する URL です。詳細については、『モデル管理』の「アプリケーション・ページへのダイレクトリンク」を参照してください。

UCMDB アプレットは次の方法で埋め込みます。

- ▶ URL 自体を使って CMDB の内容を外部アプリケーションで表示する：たとえば、特定の CI のプロパティを表示するための URL を、ダイレクト・リンク・ウィザードで作成できます。まだ HP Universal CMDB にログインしていない場合は、ログイン・ページが開きます。

注： URL を使ってインテグレーションを可能にするには、リンクを IFrame または新しいウィンドウで開く必要があります。したがって、サイト間スクリプティングの問題があるため、結果として実行される UCMDB アプレットを Javascript を使って管理することはできません。たとえば、オブジェクト ID を使って特定の CI のプロパティを表示する URL を使用する場合、同じ URL を使って別の CI のプロパティを表示することはできません。

- ▶ UCMDB アプレット・タグを使って、UCMDB アプレットを外部アプリケーションに埋め込む：これにより、インテグレーション開発者は HP Universal CMDB ユーザ・インタフェースの一部を、独自のアプリケーションの一部として表示することができます。さらに、Javascript を使って、埋め込まれた HP Universal CMDB ユーザ・インタフェースと対話することもできます。

UCMDDB アプレット・タブー 概要

UCMDDB アプレット・タグは、UCMDDB アプレットの実行に必要な環境 (HTML, Javascript, サーバ・セッション・パラメータ) を作成します。タグの背後のコードは、次のいずれかの場合に HP Universal CMDB サーバにログインできます。

- ▶ ログイン・パラメータが指定されていて、そのユーザがまだログインしていない。
- ▶ **clear session** パラメータが **true** になっている。

ユーザ・セッションが確立されると、タグは必要なアプレット環境 (HTML および Javascript コード) を JSP プロセッサの結果に書き込みます。その結果として作成される HTML ページには、指定されたサーバから UCMDDB アプレットをロードするために必要な Javascript コードが含まれています。エラーが発生した場合は、指定されたエラー文字列がそのページに出力されます。この文字列は、**userErrorMessage** パラメータを使ったカスタマイズが可能で、エラー処理シナリオを開始するための HTML 要素や HTML にラップされた Javascript コードを含めることができます。

UCMDDB アプレット・タグの例

```
<ucmdb:ucmdb_applet
serverConnectionString="x.com"
serverType="BAC"
directLinkParameters="cmd=ShowProperties&objectId=2d3b0fec35431ea01625a4&navigation=false&interfaceVersion=8.0.0"
userName="admin"
userPassword="admin"
clearSessionCookies="false"/>
```

重要：ダイレクト・リンク・ウィザードで作成した URL から、**directLinkParameters** を取得できます。パラメータの "**cmd=**" より前の部分は、UCMDDB アプレット・タグに必要な部分なので、この部分は削除してください。ダイレクト・リンクの作成方法の詳細については、『**モデル管理**』の「ダイレクト・リンクの生成ウィザード」を参照してください。

UCMDB アプレット・タグは、**< HP Universal CMDB のルート・フォルダ >**
¥j2f¥AppServer¥webapps¥site.war¥WEB-INF¥tags¥CMS¥ucmdb_applet.ta
g にあります。

UCMDB アプレット・タグを外部アプリケーション・サーバにコピーできます。
UCMDB アプレット・タグは、次の場合に HP Universal CMDB サーバからのロ
グイン・データを要求します。

- ▶ 外部アプリケーション・サーバが JSP タグと通信できる。
- ▶ HP Universal CMDB サーバが外部アプリケーション・サーバとブラウザにア
クセスできる。

ダイレクト・リンクの操作フロー

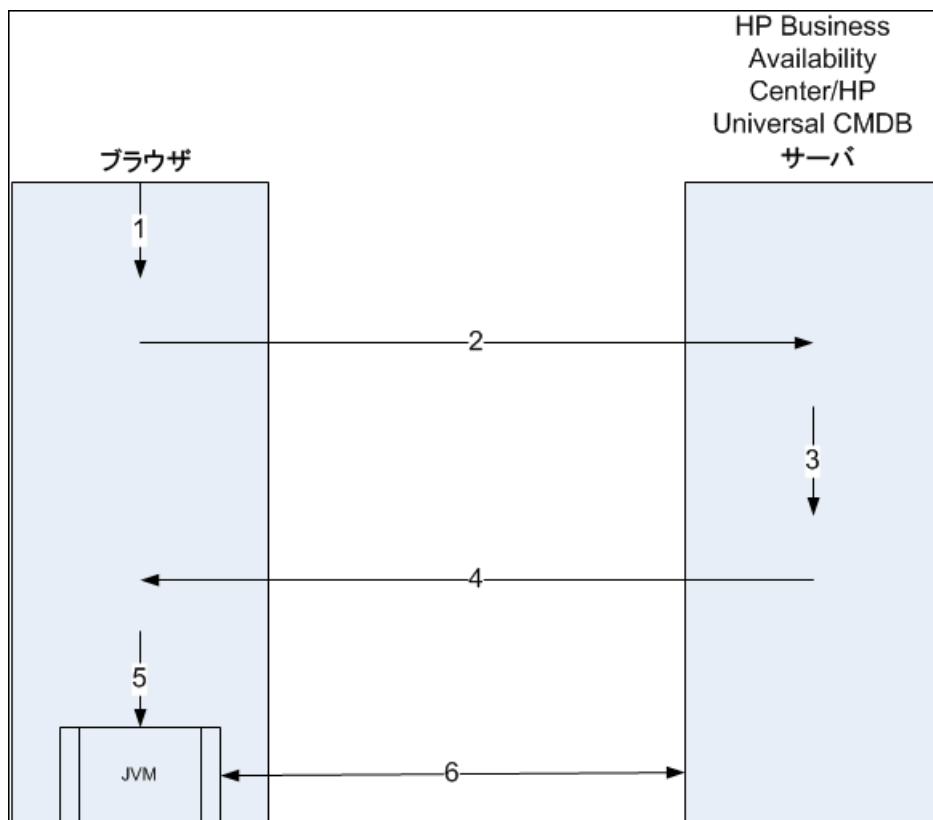
本項では、ダイレクト・リンクを使って UCMDB アプレットを外部マシンに埋
め込むシナリオをいくつか説明します。

本項は、次のフロー説明で構成されています。

- ▶ 285 ページ「通常のログイン」
- ▶ 287 ページ「ダイレクト・リンク URL を使って埋め込まれた UCMDB アプレッ
ト」
- ▶ 289 ページ「UCMDB アプレット・タグを使って埋め込まれた UCMDB アプ
レット」

通常のログイン

このフローでは、HP Universal CMDB サーバへの通常のログインについて説明します。このフローは、ダイレクト・リンク・フローの基本です。



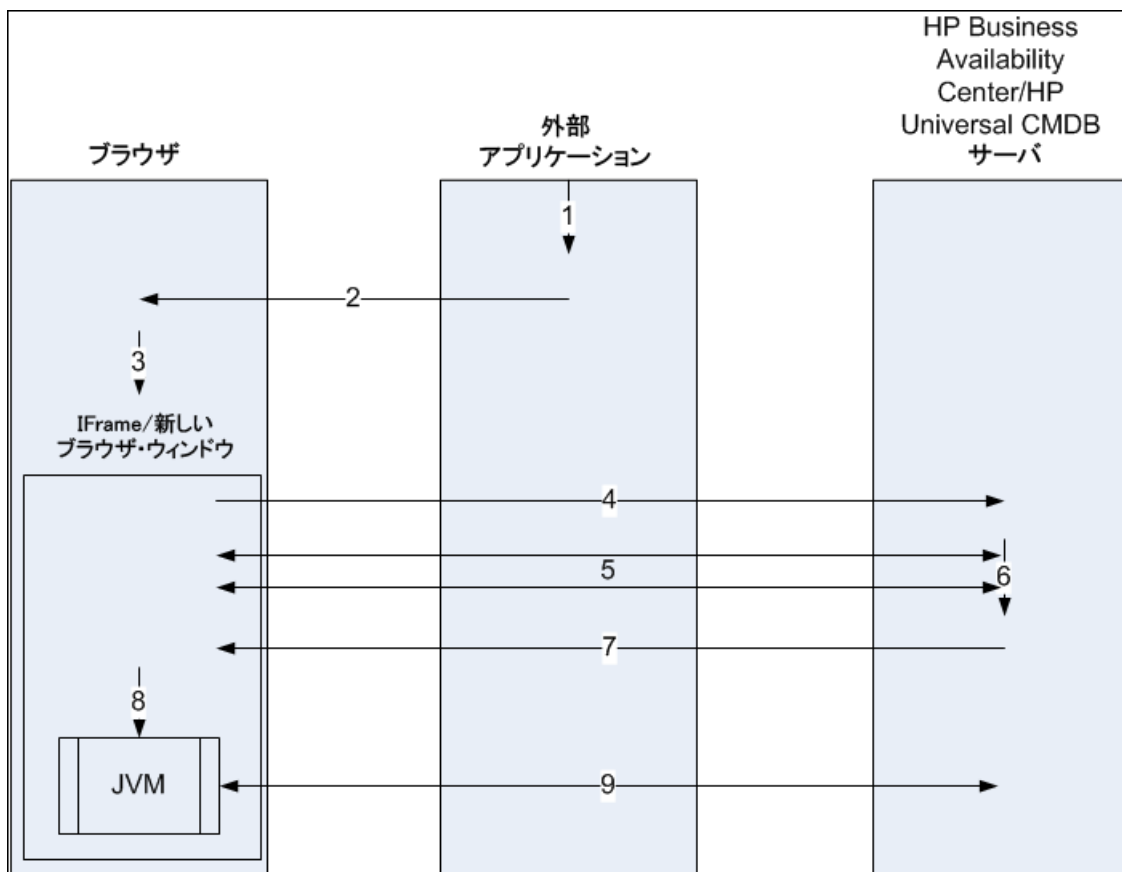
- 1 ユーザは、HP Universal CMDB サーバにログインするために、ダイレクト・リンク URL をブラウザに入力します。
- 2 ブラウザは、ログイン要求を HP Universal CMDB サーバに送信します。
- 3 サーバは、ユーザの資格情報をチェックして、必要ならばユーザ・セッションを作成します。
- 4 サーバは、要求された HP Universal CMDB の先頭ページを返します。

- 5 ブラウザは、**code base** 場所パラメータ (UCMDB アプレット・ファイルをどこからロードすべきかを JVM に指示するパラメータ) を HP Universal CMDB サーバとして、Java 仮想マシン (JVM) をロードします。
- 6 UCMDB アプレット・ファイル (jar) とデータが、HP Universal CMDB サーバとブラウザにロードされた JVM の間で転送されます。

この時点で、UCMDB アプレットがロードされ、HP Universal CMDB を通信相手のサーバとみなします。

ダイレクト・リンク URL を使って埋め込まれた UCMDB アプレット

このフローでは、ダイレクト・リンクを使って UCMDB アプレットを新しいブラウザ・フレーム内を開く（指定されたコンテキストに対して開かれます）方法について説明します。



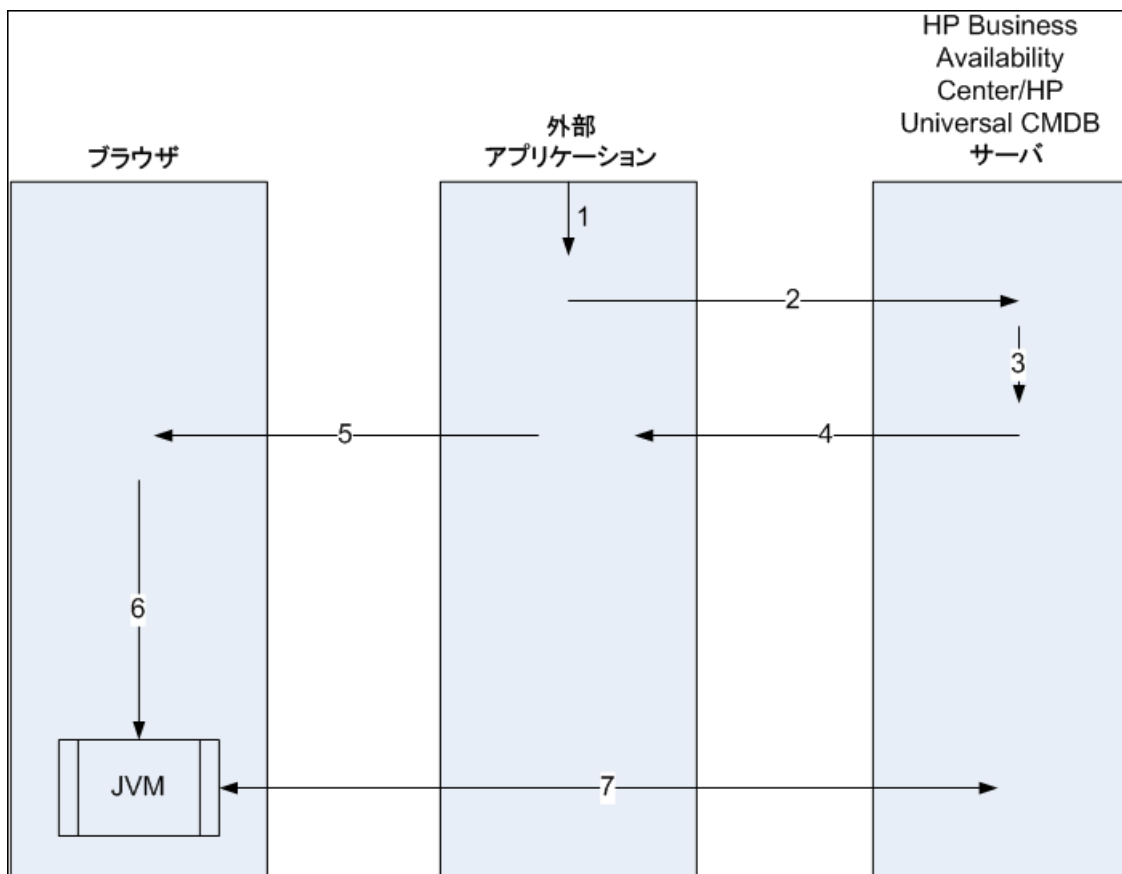
- 1 外部アプリケーションに、UCMDB アプレットを表示する必要があるページがあります。
- 2 返されるページには、ダイレクト・リンクを IFrame または新しいブラウザ・ウィンドウで開くためのディレクティブが含まれています。
- 3 ブラウザが、ダイレクト・リンクをフレーム URL として新しいフレーム (IFrame または新しいウィンドウ) を開きます。

- 4 新しいフレームが、ダイレクト・リンクを HP Universal CMDB サーバに送信します。
- 5 ユーザがまだ HP Universal CMDB にログインしていない場合は、HP Universal CMDB がログイン・ページを表示し、ユーザが資格情報を入力してログイン・ボタンをクリックするのを待ちます。
- 6 必要であれば、ユーザ・セッションが作成されます。
- 7 HP Universal CMDB サーバが、UCMDB アプレットをロードするためのディレクティブを含んだページを返します。
- 8 ブラウザがそのページを処理します。ブラウザは、UCMDB アプレットをロードするためのディレクティブに遭遇すると、JVM を起動します。ブラウザは、HP Universal CMDB サーバの場所を示す "code base" も含めて、必要なパラメータを JVM に渡します。
- 9 UCMDB アプレット・ファイル (jar) とデータが、HP Universal CMDB サーバとブラウザにロードされた JVM の間で転送されます。

重要：この HP Universal CMDB サーバは、ブラウザ・マシンからアクセス可能である必要があります。

UCMDB アプレット・タグを使って埋め込まれた UCMDB アプレット

このフローでは、UCMDB アプレット・タグをダイレクト・リンクと組み合わせて使用して、UCMDB アプレット（指定されたコンテキストに対して開かれます）を外部アプリケーション・ページに埋め込む方法について説明します。



- 1 UCMDB アプレット・タグが、外部アプリケーション内の処理済み JSP に含まれます。
- 2 UCMDB アプレット・タグが、渡されたパラメータを使って HP Universal CMDB サーバへの HTTP/HTTPS 接続を作成し、アプレットに HTML-let を要求します。

- 3 ユーザがまだログインしていない場合は、UCMDB アプレット・タグからのパラメータが使用されます（このステップに関する参考情報については、タグ自体を参照してください）。ログインが失敗すると、UCMDB アプレットはエラー・メッセージを返します。このメッセージは、あらかじめ設定されているものか、UCMDB アプレット・タグのパラメータで指定されたものです。
- 4 HP Universal CMDB サーバが、UCMDB アプレットをロードする HTML-let (Javascript が埋め込まれています) を返します。これで、外部アプリケーションがこの HTML-let をブラウザに送るページに組み込めるようになります。
- 5 外部アプリケーションが、HTML-let を含む完全なページをブラウザに送信します。
- 6 ブラウザがそのページを処理します。ブラウザが UCMDB アプレットをロードする HTML-let に遭遇すると、JVM が起動します。ブラウザは、HP Universal CMDB サーバの場所を示す "code base" も含めて、必要なパラメータを JVM に渡します。
- 7 UCMDB アプレット・ファイル (jar) とデータが、HP Universal CMDB サーバとブラウザにロードされた JVM の間で転送されます。

重要：この HP Universal CMDB サーバは、ブラウザ・マシンからも外部アプリケーション・マシンからもアクセス可能である必要があります。

索引

A

adapter.conf 169

API

HP Universal CMDB に含まれている 13

UCMDB Java

UCMDB Java API 91

UCMDB Web サービス 15

概要 13

D

discriminator.properties 185

F

Federation Framework

アダプタ・インタフェース 230

アダプタおよびマッピングのやり取り 213

fixed_values.txt 186

H

Hibernate マッピング・ツール 136

HP Release Control フェデレーション・アダプタ 226

J

Java

UCMDB API 91

O

orm.xml 177

P

persistence.xml 183

R

reconciliation_rules.txt 182

replication_config.txt 186

S

ServiceCenter/Service Manager

CIT への属性の追加 262

アダプタのデプロイメント 254

ServiceDesk アダプタ

デプロイメント 255

simplifiedConfiguration.xml 170

T

TopologyMap

UCMDB webservice API 19

TQL

フェデレート・データベース・アダプタ

タでサポートしているクエリ 133

transformations.txt 182

U

UCMDB

クエリ

webservice 23

UCMDB Java API

jar ファイル 95

アプリケーション構造 94

インテグレーション・ユーザ, 作成 95

権限 93

使用 92

UCMDB Web サービス API

使用 16

UCMDB webservice API

addCIsAndRelations 47

calculateImpact 50

chunkInfo 90

CIT 名 87

deleteCIsAndRelations 49

executeTopologyQueryByName 32

executeTopologyQueryByNameWithParameters 33

- executeTopologyQueryWithParameters 34
 - getAllClassesHierarchy 30
 - getChangedCIs 35
 - getCIsByID 37
 - getCIsByType 38
 - getClassAncestors 29
 - getCmdbClassDefinition 30
 - getFilteredCIsByType 39
 - getImpactPath 51
 - getImpactRulesByNamePrefix 52
 - getQueryNameOfView 43
 - getTopologyQueryExistingResultByName 44
 - getTopologyQueryResultCountByName 44
 - ShallowRelation 89
 - TopologyMap 19
 - TQL クエリ 19
 - updateCIsAndRelations 49
 - Web サービス, 呼び出し 22
 - 影響分析メソッドの識別子 31
 - エラー 22
 - 関係 88
 - キー属性 86
 - クエリ, 返されるプロパティ 24
 - クエリ・メソッド 32
 - クラス名 87
 - 継承したプロパティの問い合わせ 42
 - 更新メソッド 29, 47, 50
 - 構成タイプ名 87
 - パラメータ形式 27, 85, 89
 - ラベル 19
 - 例外 22
 - UCMDB Web サービス API
 - 権限 18
 - UCMDB アプレット・タブ
 - 概要 283
- W**
- Web サービス
 - UCMDB API 15
 - UCMDB webservice API 22
- あ**
- アダプタ 108
 - Federation Framework とのやり取り 213
 - ServiceCenter/Service Manager での構成ファイル 245
 - ServiceCenter/Service Manager での使用方法 244
 - ServiceCenter/Service Manager 用のデプロイメント 254
 - ServiceDesk アダプタのデプロイメント 255
 - 新しい外部データ・ストア用のアダプタの追加 231
 - インタフェース 230
 - 新規暗号鍵ファイルの生成 114
 - アダプタ機能 241
 - アプレットの埋め込み
 - ダイレクト・リンクを使用 281
- か**
- 関係
 - UCMDB webservice API 88
- く**
- クエリ
 - UCMDB webservice API 19
- こ**
- 構成タイプ
 - UCMDB webservice API 87
 - 構成ファイル
 - ServiceCenter/Service Manager アダプタの 245
 - コンバータ
 - 汎用データベース・アダプタ 186
- し**
- [新規データ ストア] ウィザード 118
- そ**
- 属性
 - 外部データ・ストアからの取得 111
- た**
- ダイレクト・リンク埋め込みアプレット
 - 概要 282
 - ダイレクト・リンクの動作フロー 284
- ち**
- 調整
 - 概要 275

クラスタに関するルール 276
 ソフトウェア要素に関するルール 277
 ホストに関するルール 275

て

データ・ストア
 新しい外部データ・ストア用のアダプタの追加 231
 作成 118
 複数からのデータの取得 109
 データ・ストア・タブ 116
 データベース・アダプタ
 設定例 190

は

パスワード
 フェデレート・アダプタの新規暗号鍵ファイルの生成 114
 派生 (Derived) プロパティ 26
 汎用データベース・アダプタ
 概要 132
 コンバータ 186
 調整 134
 デプロイメント, 最小メソッド 138
 デプロイメント, 詳細メソッド 144
 フェデレート・データベース構成ファイル
 汎用データベース・アダプタの構成ファイル 168
 プラグイン 190

ふ

フェデレート CMDB
 FTQL用の Federation Framework フロー 214
 概要 108
 トラブルシューティングと制限事項 271
 マッピング情報 112
 レプリケーション用の Federation Framework フロー 225
 ワークフロー 112
 [フェデレート CMDB] ウィンドウ 117
 フェデレート・アダプタ 108
 新規暗号鍵ファイルの生成 114
 フェデレート・データベース・アダプタ
 トラブルシューティング 206
 プラグイン

汎用データベース・アダプタ 190
 プロパティ
 派生 26

ま

マッピング
 Federation Framework とのやり取り 213

れ

[レプリケーション ジョブ] ダイアログ・ボックス 124
 レプリケーション・ジョブ・タブ 127
 [レプリケーション ジョブ統計] ウィンドウ 125
 フェデレート・データベース・アダプタ
 サポートしている SQL クエリ 133

ろ

ログ・ファイル
 フェデレート・データベース 203

