HP Server Automation

Ultimate Edition

ソフトウェアバージョン: 10.10

コンテンツユーティリティガイド

ドキュメントリリース日: 2014年6月30日 (英語版)
ソフトウェアリリース日: 2014年6月30日 (英語版)

ご注意

保証

HP製品、またはサービスの保証は、当該製品、およびサービスに付随する明示的な保証文によってのみ規定されるものとします。ここでの記載は、追加保証を提供するものではありません。ここに含まれる技術的、編集上の誤り、または欠如について、HPはいかなる責任も負いません。

ここに記載する情報は、予告なしに変更されることがあります。

権利の制限

機密性のあるコンピューターソフトウェアです。これらを所有、使用、または複製するには、HPからの有効な使用許諾が必要です。商用コンピューターソフトウェア、コンピューターソフトウェアに関する文書類、および商用アイテムの技術データは、FAR12.211および12.212の規定に従い、ベンダーの標準商用ライセンスに基づいて米国政府に使用許諾が付与されます。

著作権について

商標について

Adobe®は、Adobe Systems Incorporated (アドビシステムズ社)の登録商標です。

Intel®およびItanium®は、Intel Coporationの米国およびその他の国における登録商標です。

Microsoft®、Windows®、およびWindows® XPは，Microsoft Corporationの米国における登録商標です。

OracleとJavaは、Oracle Corporationおよびその関連会社の登録商標です。

UNIX®は、The Open Groupの登録商標です。

サポート

次のHPソフトウェアサポートオンラインのWebサイトを参照してください。

http://support.openview.hp.com

このサイトでは、HPのお客様窓口のほか、HPソフトウェアが提供する製品、サービス、およびサポートに関する詳細情報をご覧いただけます。

HPソフトウェアオンラインではセルフソルブ機能を提供しています。お客様のビジネスを管理するのに必要な対話型の技術サポートツールに、素早く効率的にアクセスできます。HPソフトウェアサポートのWebサイトでは、次のようなことができます。

- 関心のあるナレッジドキュメントの検索
- サポートケースの登録とエンハンスメント要求のトラッキング
- 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サポート窓口の検索
- 利用可能なサービスに関する情報の閲覧
- 他のソフトウェアカスタマーとの意見交換
- ソフトウェアトレーニングの検索と登録

一部のサポートを除き、サポートのご利用には、HP Passportユーザーとしてご登録の上、サインインしていただく必要があります。また、多くのサポートのご利用には、サポート契約が必要です。HP Passport IDを登録するには、次のWebサイトにアクセスしてください。

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

アクセスレベルの詳細については、次のWebサイトをご覧ください。

http://support.openview.hp.com/access_level.jsp

サポートマトリクス

サポートおよび互換性情報については、関連する製品リリースのサポートマトリクスを参照してください。サポートマトリクスと製品マニュアルは、次のHPソフトウェアサポートオンラインのWebサイトで参照できます。

http://h20230.www2.hp.com/sc/support_matrices.jsp

また、本リリースの『HP Server Automation Support and Compatibility Matrix』は、次のHPソフトウェアサポートオンラインの製品マニュアルWebサイトからダウンロードできます。

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

ドキュメントの更新情報

このリリースのServer Automation製品の最新のドキュメントは、すべて次のSA Documentation Libraryから入手できます。

http://support.openview.hp.com/selfsolve/document/KM00417675/binary/SA_10_docLibrary.html

SA Documentation Library では、このリリースに関連するガイドライン、リリースノード、サポートマトリクス、およびホワイトペーパーにアクセスできます。また、フルドキュメントセットを一括してダウンロードすることもできます。SA Documentation Libraryは、リリースごとに更新されます。また、リリースノートが更新されたときや、新しいホワイトペーパーが発行されたときにも更新されます。

情報リソースを見つける方法

Server Automationの情報リソースは、次のいずれの方法でもアクセスできます。

方法1: 新しいSA Documentation Libraryから、最新のドキュメントにタイトルとバージョンを指定してアクセスします。

方法2: [All Manuals Download] からローカルディレクトリにフルドキュメントセットを保存します。

方法3: サポートされるリリースのHP製品ドキュメントをHPソフトウェアドキュメントポータルで検索します。

各ドキュメントにアクセスするには、次の手順を実行します。

1. SA 10.x Documentation Libraryにアクセスします。

   http://support.openview.hp.com/selfsolve/document/KM00417675/binary/SA_10_docLibrary.html
2. HP Passportの資格情報を使ってログインします。
3. ドキュメントのタイトルとバージョンを指定して、[**go**] をクリックします。

ローカルディレクトリ内の完全なドキュメントセットを使用するには、次の手順を実行します。

1. フルドキュメントセットをローカルディレクトリにダウンロードするには、次の手順を実行します。
   a. SA Documentation Libraryにアクセスします。

      http://support.openview.hp.com/selfsolve/document/KM00417675/binary/SA_10_docLibrary.html
   b. HP Passportの資格情報を使ってログインします。
   c. SA 10.1バージョンの [All Manuals Download] タイトルを探します。

d [**go**] リンクをクリックして、ローカルディレクトリにZIPファイルをダウンロードします。

e ファイルを解凍します。

2 ローカルディレクトリ内のドキュメントを探すには、ドキュメントカタログ (docCatalog.html) を使用します。ローカルディレクトリにダウンロードしたドキュメントの索引ポータルが表示されます。

3 ドキュメントセット内のすべてのドキュメントを対象としてキーワードを検索するには、次の手順を実行します。

a ローカルディレクトリ内の任意のPDFドキュメントを開きます。

b [**編集**] > [**高度な検索**] を選択します (またはShift＋Ctrl＋Fキー)。

c [以下の場所にあるすべてのPDF文書] オプションを選択し、ローカルディレクトリを指定します。

d キーワードを入力し、[検索] をクリックします。

HPソフトウェアドキュメントポータルで追加ドキュメントを探すには、次の手順を実行します。

HPソフトウェアドキュメントポータルにアクセスします。

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

このサイトを利用するには、HP Passportへの登録とサインインが必要です。HP Passport IDの登録は、HP Passportのサインインページの [**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をお申し込みいただいたお客様は、更新版または最新版をご入手いただけます。詳細は、HPの営業担当にお問い合わせください。改訂状況については、「ドキュメントの更新情報」を参照してください。

製品エディション

Server Automationには、次の2つの製品エディションがあります。

- Server Automation (SA) は、Server AutomationのUltimate Editionです。Server Automationについては、『SAリリースノート』および『SAユーザーガイド: Server Automation』を参照してください。
- Server Automation Virtual Appliance (SAVA) は、Server Automation の Premium Edition です。SAVA の機能については、『SAVA Release Notes』および『SAVAクイックガイド』を参照してください。

ドキュメント変更に関する注

次の表は、前回リリースされたエディション以降の本ドキュメントに対する変更を示します。

日付	変更内容
2014年7月	SA 10.11に伴う本ドキュメントのオリジナルリリース。

目次

第1章 SAコンテンツのインポートとエクスポート

本書は、SAコンテンツの指定を担当するシステム管理者向けに作成されています。本書では、スクリプトのプログラミングとSAの基本的な知識が前提となります。詳細については、『SAユーザーガイド：Server Automation』を参照してください。

DCML Exchange Utility (DET)

DCML (Data Center Markup Language) はXMLベースの言語であり、データセンター環境にある要素とその関係を記述します。DCML Exchange Utility (DET) は、SAコンテンツをエクスポートおよびインポートするコマンドです。このコマンドを実行すると、新しくインストールしたSAマルチマスターメッシュに、既存のメッシュのコンテンツを追加できます。また、あるメッシュからコンテンツをエクスポートし、その一部を別のメッシュインスタンスにインポートすることも可能です。

cbtコマンド

DETは、SAに付属するcbtというコマンドです。cbtコマンドの詳細については、cbtコマンドの使用方法 (13ページ) およびcbtコマンドリファレンス (55ページ) を参照してください。

cbtpermコマンドは、DETの使用に関するアクセス権を設定します。cbtpermコマンドの詳細については、DETアクセス権コマンド、cbtperm (60ページ) を参照してください。

DETでは、「コンテンツ」とはユーザーが作成したSAサーバー管理情報を指します。コンテンツには、次のタイプがあります。

- アプリケーション構成、アプリケーション構成テンプレート
- カスタム拡張
- カスタムフィールド、カスタム属性
- カスタマー
- フォルダー
- パッケージ
- パッチとパッチポリシー
- サーバーコンプライアンス条件
- デバイスグループ
- ユーザーグループ

コンテンツには、管理対象となる環境に関する情報は含まれません。たとえば、ファシリティ情報やサーバープロパティはコンテンツではありません。

DETとDCMLの関係

DETがエクスポートしたコンテンツには、DCML Framework Specification v0.11との互換性があります。このバージョンは、初めて公開されたDCML仕様です。DCML Exchange Toolは、独自の拡張スキーマを使用して、Server Automationからエクスポートしたコンテンツを記述します。エクスポートではdata.rdfという有効なDCMLインスタンスドキュメントが作成され、DCMLに準拠したプロセッサーで解析できます。

カスタムフィールドとカスタム属性

各カスタムフィールドは、名前空間に存在するオブジェクトです。DETがアクセスできるのは、ユーザーが表示可能なデフォルトの名前空間内にあるオブジェクトに限定されます(したがってエクスポートのみ)。他の名前空間(OPSWAREなど)のオブジェクトはエクスポートできません。他の名前空間にあるオブジェクトをエクスポートする必要がある場合は(OSシーケンスなど)、アプリケーション専用のAPI(OSシーケンスAPIなど)を使用してエクスポートします。

カスタム属性はすべてエクスポートの対象となり、これにはエンドユーザーに表示されない属性(キーの先頭が__OPSW)も含まれます。

かカスタムフィールドとカスタム属性のインポートでは、インポートした値(Null値を含む)が既存の値に上書きされます。

第2章 cbtコマンドの使用方法

コンテンツのエクスポート

cbtコマンドは、指定したコンテンツをターゲットのSAメッシュからRDF/XMLファイルにエクスポートします。エクスポートしたコンテンツは、さらに別のSAメッシュにインポートできます。コンテンツのインポート (36ページ) を参照してください。

cbtコマンドは次のディレクトリに格納されています。

```
/opt/opsware/cbt/bin
```

次に、エクスポートコマンドを示します。

```
cbt -e <コンテンツディレクトリ> -f <フィルターファイル> -cf <ターゲットのコア構成>
```

次に、コマンドと引数を示します。

- コンテンツディレクトリ: エクスポートしたコンテンツの格納先となるディレクトリへのパス。このディレクトリが存在しない場合、エクスポート時に作成されます。
- フィルターファイル: ターゲットSAメッシュからエクスポートするコンテンツをDETに指示するルール群。このファイルの作成方法については、エクスポートフィルター (13ページ) を参照してください。
- ターゲットコア構成: 各種SAコンポーネントにアクセスする際に、コンポーネントの場所と使用するIDをDETに指示する構成ファイル。このファイルの作成方法については、ターゲットメッシュの構成ファイルの作成 (50ページ) を参照してください。

エクスポートコマンドは、同じ引数で何度も実行できますが、次の点に注意してください。

- フィルターを指定する場合、DETはコンテンツディレクトリ内にある前回エクスポートした内容を無視し、エクスポートプロセスを新しく実行します。
- 有効なエクスポート (過去に実行が成功したエクスポート) が保存されているコンテンツディレクトリを指定してエクスポートコマンドを実行すると、上書きを確認するプロンプトが表示されます。上書きしない場合、DETは終了します。

スタンドアロンメッシュでエクスポートまたはインポートを開始する場合、コマンドセンターのコアコンポーネントをシャットダウンしてください。これにより、プロセスが終了するまでユーザーはSAコンテンツを変更できなくなります。

マルチマスターメッシュでは、まずマルチマスターツールを使用してメッシュを取得し、競合がないことを確認します。その後、メッシュ内にあるコマンドセンターをすべてシャットダウンします。これにより、プロセスが終了するまでユーザーはSAコンテンツを変更できなくなります。

コマンドセンターのコアコンポーネントの停止と再開については、『SA 管理ガイド』を参照してください。

エクスポートフィルター

エクスポートフィルターとは、ユーザーが指定するルールであり、エクスポート対象のコンテンツ (エクスポート後にインポートするコンテンツ) をDETに指示します。エクスポートフィルターは、次のコンテンツタイプで使用します。

- アプリケーション構成のエクスポートフィルター

- アプリケーション構成テンプレートのエクスポートフィルター
- 監査フィルター
- カスタム拡張のエクスポートフィルター
- カスタムフィールドスキーマのエクスポートフィルター
- カスタマーのエクスポートフィルター
- フォルダーのエクスポートフィルター
- OSビルド計画のエクスポートフィルター
- OSのエクスポートフィルター
- パッケージのエクスポートフィルター
- パッチのエクスポートフィルター
- パッチポリシーのエクスポートフィルター
- サーバーコンプライアンス条件 (監査ポリシー) のエクスポートフィルター
- サーバー (デバイス) グループのエクスポートフィルター
- サービスレベルのエクスポートフィルター
- スナップショットフィルター
- テンプレートのエクスポートフィルター
- ユーザーグループのエクスポートフィルター

例: エクスポートフィルターファイル

DETは、指定したフィルターファイルに含まれるエクスポートフィルターを読み取ります。フィルターファイルは、RDF/XMLでエンコードされています。次の例は、エクスポートフィルタールールが1つ指定されている簡単なフィルターファイルです。

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE rdf:RDF [
3. <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
4. ]>
5. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6.    xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
7. <PackageFilter rdf:ID="exportCIPackages">
8. <packageType rdf:resource="&filter;RPM"/>
9. <packageName>software1.0.0-1.rpm</packageName>
10. </PackageFilter>
11. </rdf:RDF>
```

この例では、行1～6で、標準フィルターヘッダーが指定されています。この行はすべてのフィルターで共通です。また、行11は標準フィルターフッターです。

行7～10は、フィルターごとに固有の内容であり、フィルターの関数を示します。

この例では、エクスポートフィルタールールは1つしかありませんが、標準ヘッダーとフッターの行の間に、任意の数のフィルターを指定できます。たとえば、次のフィルターではエクスポートフィルタールールが3つ指定されています。

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <!DOCTYPE rdf:RDF [
3.   <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
4.  ]>
5.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
6.           xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
7.   <PackageFilter rdf:ID="pkg1">
8.    <packageType rdf:resource="&filter;RPM"/>
9.    <packageName>software1.0.0-1.rpm</packageName>
10.  </PackageFilter>
11.  <PackageFilter rdf:ID="pkg2">
12.     <packageType rdf:resource="&filter;ZIP"/>
13.  </PackageFilter>
14.  <CustomExtensionFilter rdf:ID="exportCustExtBulkPasswd">
15.  <scriptName>Bulk_Password_Changes</scriptName>
16.  <CustomExtensionFilter/>
17.  </rdf:RDF>
```

以上のサンプルフィルターは、DETインストールディレクトリにある次のディレクトリに格納されています。

<インストールディレクトリ>/filters

このディレクトリには、各タイプのサンプルフィルターが格納されています。また、all.rdfフィルターは、SAメッシュから既知のSAデータ型をすべてエクスポートするフィルターです。

CBTエクスポートコマンドの実行では、選択したフィルターの絶対パスを指定する必要があります。例:

```
cbt -e /tmp -f /opt/opsware/cbt/filters/filter.rdf
```

では次に、フィルタータイプとパラメーターについて詳しく説明します。一般的に、フィルタータイプは、SAクライアントで操作できるオブジェクトタイプに対応しています。たとえばパッチフィルターは、SAクライアントのパッチオブジェクトに対応しています。

アプリケーション構成のエクスポートフィルター

アプリケーション構成エクスポートフィルターは、DETでエクスポートするアプリケーション構成を指定します。アプリケーション構成とは、1つまたは複数のアプリケーション構成テンプレートファイルを格納するコンテナーです。したがって、アプリケーション構成をエクスポートすると、そこに含まれるテンプレートもすべてエクスポートされます。

表1　アプリケーション構成エクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があります。フィルターの名前は、rdf:ID="一意の名前" という形式で指定します。

表2　アプリケーション構成エクスポートフィルターのネスト要素

要素	説明
configurationName (オプション)	アプリケーション構成の名前を指定します(オプション)。アプリケーション構成の名前を指定してエクスポートする場合、この要素を使用します。
customerName (オプション)	指定したカスタマーに関連付けられているアプリケーション構成をすべてエクスポートします(オプション)。
osPlatform rdf:resource (オプション)	指定したOSに関連付けられているアプリケーション構成をすべてエクスポートします(オプション)。

アプリケーション構成エクスポートフィルターの例

アプリケーション構成をすべてエクスポートします。

```
<ApplicationConfigurationFilter rdf:ID="getAllAppConfigs"/>
```

"iPlanet" という名前のアプリケーション構成の中で、カスタマー独立であり、SunOS 5.8オペレーティングシステムに関連付けられているもののみをエクスポートします。

```
<ApplicationConfigurationFilter rdf:ID="getSpecificAppConfigs">

   <configurationName>iPlanet</configurationName>

   <customerName>Customer Independent</customerName>

   <osPlatform rdf:resource="&filter;SunOS_5.8"/>

</ApplicationConfigurationFilter>
```

アプリケーション構成テンプレートのエクスポートフィルター

アプリケーション構成テンプレートエクスポートフィルターは、DET でエクスポートするアプリケーション構成テンプレートファイルを指定します。

表3　アプリケーション構成テンプレートエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があります。フィルターの名前は、rdf:ID="一意の名前" という形式で指定します。

表4　アプリケーション構成テンプレートエクスポートフィルターのネスト要素

要素	説明
configurationFileName（オプション）	アプリケーション構成テンプレートの名前を指定します (オプション)。アプリケーション構成テンプレートの名前を指定してエクスポートする場合、この要素を使用します。
osPlatform rdf:resource（オプション）	指定した OS に関連付けられているアプリケーション構成をすべてエクスポートします (オプション)。
customerName （オプション）	指定したカスタマーに関連付けられているアプリケーション構成テンプレートをすべてエクスポートします (オプション)。

アプリケーション構成テンプレートエクスポートフィルターの例

アプリケーション構成テンプレートをすべてエクスポートします。

```
<ApplicationConfigurationFileFilter rdf:ID="getAllAppConfigTemps"/>
```

"iplanet6.1_mimetypes.tpl" という名前のアプリケーション構成テンプレートの中で、カスタマー独立であり、Red Hat Enterprise Linux AS 3 X86_64 オペレーティングシステムに関連付けられているものをエクスポートします。

```
<ApplicationConfigurationFileFilter rdf:ID="getSpecificAppConfigTemp">
```

```
<configurationFileName>iplanet6.1_mimetypes.tpl</configurationFileName>

<customerName>Customer Independent</customerName>

<osPlatform rdf:resource="&filter;Red_Hat_Enterprise_Linux_AS_3_X86_64"/>

</ApplicationConfigurationFileFilter>
```

監査フィルター

監査フィルターでは、DETによってSAコア/メッシュからエクスポートする監査を指定します。これにより、エクスポートした内容を別のSAコア/メッシュにインポートできます。

表5 監査フィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

表6 監査フィルターのネスト要素

要素	説明
auditPolicyName	エクスポートする監査ポリシーの名前。

例:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
]>
```

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
<AuditPolicyFilter rdf:ID="apf1">
  <auditPolicyName>My Audit Policy</auditPolicyName>
</AuditPolicyFilter>
</rdf:RDF>
```

カスタム拡張のエクスポートフィルター

カスタム拡張エクスポートフィルターは、DETでエクスポートするカスタム拡張を個別に指定、またはすべてを指定します。カスタム拡張の一部として複数のカスタム拡張をエクスポートする場合、エクスポートするカスタム拡張ごとにフィルターを作成します。

表7　　カスタム拡張エクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

表8　　カスタム拡張エクスポートフィルターのネスト要素

要素	説明
scriptName（オプション）	エクスポートするスクリプトを指定します(オプション)。スクリプト名は、アカウントプレフィックスを含みません。この要素を指定しないと、すべてのカスタム拡張スクリプトがエクスポートされます。

カスタム拡張エクスポートフィルターの例

Bulk_Password_Changesという名前のカスタム拡張スクリプトのみをエクスポートします。

```
<CustomExtensionFilter rdf:ID="exportCustExtBulkPasswd">
  <scriptName>Bulk_Password_Changes</scriptName>
</CustomExtensionFilter>
```

カスタム拡張スクリプトをすべてエクスポートします。

```
<CustomExtensionFilter rdf:ID="exportAllCustExtScripts"/>
```

カスタムフィールドスキーマのエクスポートフィルター

カスタムフィールドスキーマのエクスポートフィルターでは、DETによって、すべてのカスタムフィールド定義をメッシュからエクスポートします。

表9　　カスタムフィールドスキーマエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

カスタムフィールドスキーマエクスポートフィルターの例

メッシュからすべてのカスタムフィールド定義をエクスポートします。

```
<CustomFieldSchemaFilter rdf:ID="getCustomFieldsSchema"/>
```

カスタマーのエクスポートフィルター

カスタマーエクスポートフィルターは、DETによって、すべてまたは一部のカスタマーをメッシュからエクスポートします。

表10　　カスタマーエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

表11　　カスタマーエクスポートフィルターのネスト要素

要素	説明
customerName （オプション）	エクスポートするカスタマーを指定します(オプション)。

カスタマーエクスポートフィルターの例

メッシュからすべてのカスタマーをエクスポートします。

```
<CustomerFilter rdf:ID="exportAllCustomers"/>
```

"Acme Computers"という名前のカスタマーをメッシュからエクスポートします。

```
<CustomerFilter rdf:ID="exportAcmeCustomer">
  <customerName>Acme Computers</customerName>
</CustomerFilter>
```

フォルダーのエクスポートフィルター

フォルダーフィルターでは、DETでエクスポートする対象として、フォルダーに関連付けられている項目またはフォルダー内にある項目を指定します。次に、対象となる項目を示します。

- アプリケーション構成テンプレート
- アプリケーション構成
- 属性とカスタム属性
- フォルダー内にある監査ポリシー
- フォルダー内にあるOSビルド計画
- フォルダー内にあるOSシーケンス
- フォルダー内にあるパケット
- フォルダー内にあるスクリプト
- フォルダー内にあるソフトウェアポリシー
- 名前でユーザーグループを参照するFolderACL(ユーザーグループはエクスポートされません)

- パスに従って、指定したフォルダーまでのフォルダーをすべて示すプレースホルダー
- サブフォルダー(オプション)

表12　フォルダーエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

表13　フォルダーエクスポートフィルターのネスト要素

要素	説明
path（必須）	フォルダーパスを指定します(必須)。
recursive（オプション）	サブフォルダーをエクスポートします(オプション)。

フォルダーエクスポートフィルターの例

たとえば、次のフォルダー構造があるとします。

```
/
/A
/A/B
```

次の例では、上記のフォルダー構造において、エクスポートされるフォルダーを示します。

フォルダーAをエクスポート:

```
<FolderFilter rdf:ID="f1">
   <path>/A</path>
   <recursive rdf:resource="&filter;No"/>
</FolderFilter>
```

フォルダーBをエクスポート:

```
<FolderFilter rdf:ID="f1">
   <path>/A/B</path>
</FolderFilter>
```

フォルダーAとBをエクスポート:

```
<FolderFilter rdf:ID="f1">
   <path>/A</path>
   <recursive rdf:resource="&filter;Yes"/>
</FolderFilter>
```

フォルダーAとBをエクスポート:

```
<FolderFilter rdf:ID="f1">
   <path>/</path>
   <recursive rdf:resource="&filter;Yes"/>
</FolderFilter>
```

OSビルド計画のエクスポートフィルター

OSビルド計画のエクスポートフィルターは、DETでエクスポートするOSビルド計画を指定します。

表14　OSビルド計画のエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があります。フィルターの名前は、rdf:ID=“一意の名前"という形式で指定します。

表15　OSビルド計画のエクスポートフィルターのネスト要素

要素	説明
osBuildPlanName (オプション)	エクスポートするOSビルド計画名。
folderName (オプション)	エクスポートするOSビルド計画を含むフォルダーの名前。 **注:** folderNameネスト要素は、フォルダーパスではなく、フォルダー名を参照します。folderNameネスト要素を使用してOSビルド計画フィルターを実行するときには、指定した名前を持つすべてのフォルダーが考慮されます。

OSビルド計画のエクスポートフィルター例

すべてのOSビルド計画をエクスポートします。

```
<OSBuildPlanFilter rdf:ID="osbp1"/>
```

"OS Build Plan 1" という名前を持つすべてのOSビルド計画をエクスポートします。

```
<OSBuildPlanFilter rdf:ID="osbp2">
  <osBuildPlanName>OS Build Plan 1</osBuildPlanName>
</OSBuildPlanFilter>
```

"Build Plan Folder" フォルダー内にあるすべてのOSビルド計画をエクスポートします。

```
<OSBuildPlanFilter rdf:ID="osbp3">
  <folderName>Build Plan Folder</folderName>
</OSBuildPlanFilter>
```

OSのエクスポートフィルター

オペレーティングシステムのエクスポートフィルターは、DET でエクスポートするオペレーティングシステムのノードまたはタイプを指定します。

表16　OSエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

表17　OSエクスポートフィルターのネスト要素

要素	説明
osName（オプション）	SAクライアントでユーザーが割り当てたOSの名前。
osPlatform（必須）	必須のネスト要素です。これは空の要素であり、rdf:resourceパラメーターを指定します。このパラメーターは、『SA Support and Compatibility Matrix』に記載されたサポートされるオペレーティングシステムの1つを参照することができます。

OSエクスポートフィルターの例

"7.1 for mwp" Red Hat Linux 7.1 OSをエクスポートします。

```
<OSFilter rdf:ID="exportOSRHLinux71">
  <osPlatform rdf:resource="&filter;Red_Hat_Linux_7.1"/>
  <osName>7.1 for mwp</osName>
</OSFilter>Export all Solaris 5.6 operating systems.
<OSFilter rdf:ID="exportOSSun56">
  <osPlatform rdf:resource="&filter;SunOS_5.6"/>
</OSFilter>
```

パッケージのエクスポートフィルター

パッケージのエクスポートフィルターは、DETによって、すべてまたは一部のパッケージをメッシュからエクスポートします。フォルダーのプレースホルダーはエクスポートの対象です。また、フォルダーのパス上にあるすべてのフォルダーのプレースホルダーもエクスポートの対象となります。

Microsoftのホットフィックスとサービスパックの場合、エクスポートするMicrosoftパッケージがメッシュに表示されていても、パッケージのバイナリファイルがまだアップロードされていないことがあります。たとえば、Microsoftパッチデータベースをメッシュにアップロードしても、実際のバイナリファイルがアップロードされていない場合があり、このようなケースでは、パッケージのユニットレコードはSAモデルで作成されますが、エクスポートするコンテンツがありません。このような場合、パッケージエクスポートフィルターを使ってパッケージコンテンツをエクスポートしても、Microsoftパッケージのコンテンツはエクスポートされません。

表18　パッケージエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

表19　パッケージエクスポートフィルターのネスト要素

要素	説明
packageType（必須）	エクスポートするパッケージタイプを指定します(必須)。このパラメーターでは、次のいずれかのパッケージタイプを指定できます。 • AIX_Base_Fileset • AIX_LPP • AIX_Update_Fileset • APAR • Build_Customization_Script • Chef_Cookbook • DEB • HPUX_Depot • HPUX_Fileset • HPUX_Patch_Fileset • HPUX_Patch_Product • HPUX_Product • Relocatable_ZIP • RPM • Solaris_Package • Solaris_Package_Instance • Solaris_Patch • Solaris_Patch_Cluster • Unknown • Windows_Hotfix • Windows_MSI • Windows_OS_Service_Pack • Windows_Update_Rollup • ZIP
packageName（オプション）	パッケージの名前を指定します(オプション)。パッケージの名前とは、SA クライアントの[パッケージのプロパティ]ページにある[名前]フィールドの内容であり、パッケージのファイル名ではありません。
osPlatform（オプション）	指定したパッケージ名のオペレーティングシステムを指定します(オプション)。このパラメーターは、『SA Support and Compatibility Matrix』に記載されたサポートされるオペレーティングシステムの1つを参照することができます。
customerName（オプション）	指定したパッケージ名のカスタマーを指定します(オプション)。

パッケージエクスポートフィルターの例

プラットフォームSunOS_5.8に関連するすべてのRPMパッケージをエクスポートします。

```
<PackageFilter rdf:ID="exportCIPackages">
    <packageType rdf:resource="&filter;RPM"/>
    <osPlatform rdf:resource="&filter;SunOS_5.8"/>
</PackageFilter>
```

再配置可能なZIPファイルは、1つのサーバー上のさまざまな場所にインストールできます。再配置可能なZIPファイルの名前は、親のZIPファイルと同じなので、親の名前を指定するだけで、そのZIPファイルのすべてのバージョンがエクスポートされます。たとえば、ZIPファイルに次のような階層構造があるとします。

- ZIP `hmp.zip` (SunOS 5.8)
  - 再配置可能ZIPである`hmp.zip`は`/foo`にインストールされています。
  - 再配置可能ZIPである`hmp.zip`は`/bar`にインストールされています。

上記のZIPファイル構造で次のフィルターを指定すると、配置可能なZIPファイル (`/foo`と`/bar`) は両方ともエクスポートされます。

```
<PackageFilter rdf:ID="p1">
    <packageType rdf:resource="&filter;Relocatable_ZIP"/>
    <packageName>hmp.zip</packageName>
    <osPlatform rdf:resource="&filter;SunOS_5.8"/>
</PackageFilter>
```

パッチのエクスポートフィルター

パッチのエクスポートフィルターは、DETでエクスポートするパッチまたはパッチタイプを指定します。

Windowsパッチコンテンツを DET 2.5で定義している場合、Windows MBSAパッチ定義がソースメッシュとターゲットメッシュで同じであることを確認してください。未定義の Windows パッチはインポートの対象から除外されてしまいます。

表20 パッチエクスポートフィルターのパラメーター

パラメーター	説明
`rdf:ID`	各フィルターには一意の名前があり、次の形式で指定されます。 `rdf:ID="一意の名前"`

表21 パッチフィルターのネスト要素

要素	説明
`patchType` （必須）	`rdf:resource`パラメーターを持つ必須のネスト要素。このパラメーターでは、次のいずれかのパッチタイプを指定できます。 • APAR • APAR_FILESET • UPDATE_FILESET • AIX_Update_Fileset • HPUX_PATCH_PRODUCT • HPUX_Patch_Product • HPUX_PATCH_FILESET • HPUX_Patch_Fileset • SOL_PATCH • Solaris_Patch • SOL_PATCH_CLUSTER • Solaris_Patch_Cluster • HOTFIX • Windows_Hotfix • SERVICE_PACK • Windows_OS_Service_Pack • PATCH_META_DATA • Microsoft_Patch_Database

表21　　パッチフィルターのネスト要素（続き）

要素	説明
patchName（オプション）	個々のパッチの名前を指定します（オプション）。この名前にはパッチのunit_nameを指定します。これは、SAクライアントで表示されます。
patchLocale（オプション）	ロケールを指定します。これによって、Windowsパッチの言語が識別されます。Windows以外のパッチでは、この要素は無視されます。 ロケールにはen、ja、koなどがあり、順に英語、日本語、韓国語を示します。デフォルトは英語です。Windowsパッチで現在サポートされるロケールのリストは、『SAユーザーガイド：サーバーのパッチ適用』を参照してください。

パッチフィルターの例

IY13260 APARをエクスポートします。

```
<PatchFilter rdf:ID="exportAPARIY13260">
  <patchName>IY13260</patchName>
  <patchType rdf:resource="&filter;APAR"/>
</PatchFilter>
```

Solarisパッチをすべてエクスポートします。

```
<PatchFilter rdf:ID="exportSolPatches">
  <patchType rdf:resource="&filter;SOL_PATCH"/>
</PatchFilter>
```

パッチ名がQ123456であり、ロケールが日本語のパッチをエクスポートします。

```
<PatchFilter rdf:ID="pf1">
  <patchName>Q123456</patchName>
  <patchLocale>ja</patchLocale>
</PatchFilter>
```

パッチポリシーのエクスポートフィルター

パッチポリシーエクスポートフィルターは、DETでエクスポートするユーザー定義のパッチポリシーを指定します(ベンダー推奨のポリシーはエクスポート対象になりません)。

特定のパッチポリシーをフィルター処理する場合は、オプションのネスト要素である <patchPolicyName> と <osPlatform> を指定します。オプションのネスト要素を指定しないと、ターゲットメッシュに含まれるすべてのパッチポリシーがエクスポートの対象になります。

パッチポリシーフィルターは、Solarisパッチポリシーをエクスポートしません。Solarisパッチポリシーをエクスポートするには、フォルダーエクスポートフィルターを使用して親フィルターをエクスポートする必要があります。(フォルダーのエクスポートフィルター (19ページ) を参照してください)。

表22　　パッチポリシーエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

表23　　パッチポリシーエクスポートフィルターのネスト要素

要素	説明
patchPolicyName	パッチポリシーの名前を個別に指定します (オプション)。
osPlatform	rdf:resource パラメーターを使用して、パッチポリシーのオペレーティングシステムを指定します (オプション)。このパラメーターは、次の構文を使用してサポートされるWindowsオペレーティングシステムの1つを参照することができます。 <OS>_<Version>_<Revision(if applicable)>_<Architecture> たとえば、Windows 2008 R2 IA64の場合、有効なOSプラットフォーム属性は Windows_2008_R2_IA64 です。他のプラットフォーム例の場合、Windows_2008_R2_x64、Windows_2012_R2_x64などになります。

パッチポリシーエクスポートフィルターの例

ターゲットメッシュからすべてのパッチポリシーをエクスポートします。

```
<PatchPolicyFilter rdf:ID="PatchPolicies1"/>
```

名前が "BestWindowsPoliciesNT"、オペレーティングシステムがWindows NT 4.0のパッチポリシーと、名前が "BestWindowsPolicies2003"、オペレーティングシステムがWindows 2003のパッチポリシーのみをエクスポートします。

```
<PatchPolicyFilter rdf:ID="PatchPolicies2"/>
  <patchPolicyName>BestWindowsPoliciesNT</patchPolicyName>
  <osPlatform rdf:resource="&filter;Windows_NT_4.0/>
</PatchPolicyFilter>

<PatchPolicyFilter rdf:ID="PatchPolicies3"/>
  <patchPolicyName>BestWindowsPolicies2003</patchPolicyName>
  <osPlatform rdf:resource="&filter;Windows_2003"/>
</PatchPolicyFilter>
```

Windows 2003オペレーティングシステムのパッチポリシーをすべてエクスポートします。

```
<PatchPolicyFilter rdf:ID="PatchPolicies4"/>
  <osPlatform rdf:resource="&filter;Windows_2003"/>
</PatchPolicyFilter>
```

スクリプトのエクスポートフィルター

スクリプトは、その親フォルダーをエクスポートすることでのみエクスポートできます。フォルダーのエクスポートフィルター (19ページ) を参照してください。

サーバーコンプライアンス条件 (監査ポリシー) のエクスポートフィルター

サーバーコンプライアンス条件のエクスポートフィルターは、DETでエクスポートする監査ポリシーを指定します。

表24 サーバーコンプライアンス条件エクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があります。フィルターの名前は、rdf:ID="一意の名前" という形式で指定します。

表25 サーバーコンプライアンス条件フィルターのネスト要素

要素	説明
selectionCriteriaNam (オプション)	監査ポリシーの名前を指定します (オプション)。監査ポリシーの名前を指定してエクスポートする場合、この要素を使用します。
osType rdf:resource (オプション)	指定したOSに関連付けられている監査ポリシーをすべてエクスポートします (オプション)。このパラメーターで指定できる値は、WindowsまたはUnixのいずれかです。次に例を示します。 <osType rdf:resource="&filter;Windows"/> または <osType rdf:resource="&filter;Unix"/>

サーバーコンプライアンス条件エクスポートフィルターの例

すべての監査ポリシーをエクスポートします。

```
<ComplianceSelectionCriteriaFilter rdf:ID="getAllSelectionCriteria"/>
```

Windowsオペレーティングシステムに関連付けられ、"My Audit Policy" という名前の監査ポリシーをエクスポートします。

```
<ComplianceSelectionCriteriaFilter rdf:ID="getSpecificSelectionCriteria">
  <selectionCriteriaName>My Audit Policy</selectionCriteriaName>
  <osType rdf:resource="&filter;Windows"/>
</ComplianceSelectionCriteriaFilter>
```

サーバー(デバイス)グループのエクスポートフィルター

サーバーグループのエクスポートフィルターは、DETでエクスポートするサーバーグループを指定します。

表26　　サーバーグループエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

表27　　サーバーグループフィルターのネスト要素

要素	説明
path（必須）	エクスポートするサーバーグループの名前を指定します(必須)。
directive（必須）	空の必須コンテンツ要素であり、rdf:resourceパラメーターを1つ指定します。エクスポートするコンテンツグループを指定できます。このパラメーターは、次の3つの定数の1つを参照します。 • Node: パスのリーフノードのみをエクスポートします。パスが存在しない場合は、空のプレースホルダー(名前と説明、ルールは除外)が作成されます。 • Path: パスに沿ってすべてのグループ(名前、説明、ルール)をエクスポートしますが、子孫は除外します。 • Descendants: 指定したパス(リーフノードも含む)の子孫をすべてエクスポートします。 たとえば、次の構造を持つパスがあるとします。 /Group/A/B/C/D 次のように指定します。 /Group/A/B rdf:resourceパラメーターがNodeの場合、サーバーグループノードBがエクスポートされます。rdf:resourceパラメーターがPathの場合、サーバーグループノードAとBがエクスポートされます。 rdf:resourceパラメーターがDescendantsの場合、サーバーグループノードB、C、Dがエクスポートされます。

表27　サーバーグループフィルターのネスト要素（続き）

要素	説明
customerName （オプション）	接続サーバーグループノードのエクスポート対象を制限します (オプション)。指定したカスタマーが所有する接続ノードのみがエクスポートされます。 customerName要素を指定しても、動的なサーバーグループルールが参照するノードのエクスポートには影響しません。

注

- コア固有の情報はエクスポートされません。これには、グループメンバーシップ、「最終使用日」、履歴などのプロパティが含まれます。
- 静的グループのエクスポートも可能ですが、エクスポートの対象にはグループの名前と説明のみが含まれます。
- 動的なグループルールがカスタムフィールドを参照している場合、カスタムフィールドスキーマでエクスポート可能なのは個々のカスタムフィールドのみであり、スキーマ全体をエクスポートすることはできません。
- パスでは、グループがパブリックかプライベートかを指定します。パスに/Group/Publicと指定すると(Descendantsも指定)、パブリックグループをすべてエクスポートできます。
- プライベートグループはエクスポートできないので、パスに/Group/Privateと指定してエクスポートするとエラーが発生します。
- インポートした動的サーバーグループに、ルールがないという可能性もあります。このような状況は、ソースグループに "Facility is C07" や "Realm is Sat02" などのようなルールしかない場合に発生します。ファシリティやレルムはコア固有のルールなので、エクスポートの対象にはなりません。
- サーバーIDを参照するルールはエクスポートの対象になりません。たとえば、"Server ID equals 55500001" のようなルールはエクスポートされません。
- アタッチされたソフトウェアポリシーはすべてエクスポートされます。

サーバーグループエクスポートフィルターの例

すべてのパブリックサーバーグループをメッシュからエクスポートします。

```
<ServerGroupFilter rdf:ID="exportPubServGroups">
    <path>/Group/Public/</path>
    <directive rdf:resource="&filter;Descendants"/>
</ServerGroupFilter>
```

"NT Servers" という名前のパブリックサーバーグループと、そこに含まれるサブグループをすべてエクスポートします。

```
<ServerGroupFilter rdf:ID="exportNTServGroups">
    <path>/Group/Public/NT Servers</path>
    <directive rdf:resource="&filter;Descendants"/>
</ServerGroupFilter>
```

"Production Web Servers" という名前のパブリックサーバーグループ (サブグループは除く) のみをエクスポートします。

```
<ServerGroupFilter rdf:ID="exportProdWebServGroups">
    <path>/Group/Public/Production Web Servers</path>
    <directive rdf:resource="&filter;Node"/>
</ServerGroupFilter>
```

"Production Web Servers" という名前のパブリックグループと、"iPlanet" という名前のサブグループをエクスポートします。これ以外のサブグループはエクスポートしません。

```
<ServerGroupFilter rdf:ID="exportProdWebServGroupsIP">
  <path>/Group/Public/Production Web Servers/iPlanet</path>
  <directive rdf:resource="&filter;Path"/>
</ServerGroupFilter>
```

サービスレベルのエクスポートフィルター

サービスレベルのエクスポートフィルターは、DETでエクスポートするサービスレベルノードを指定します。

表28　サービスレベルエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

表29　サービスレベルエクスポートフィルターのネスト要素

要素	説明
path （必須）	最上位ノードを基準にした、エクスポートするノードの絶対パス。パスの区切り文字はスラッシュ記号 (/) です。
directive （必須）	空のコンテンツ要素であり、rdf:resourceパラメーターを1つ指定します。このパラメーターは、次の3つの定数の1つを参照します。 • Descendants: 指定したパス (リーフノードも含む) の子孫をすべてエクスポートします。 • Node: 指定したノードのみをエクスポートします。 • Path: パスのノードに沿ってすべてエクスポートし、他のノードはエクスポートしません。 たとえば、次の構造を持つパスがあるとします。 /Service Level/A/B/C/D 次のように指定します。 /Service Level/A/B rdf:resourceパラメーターがNodeの場合、ノードBがエクスポートされます。 rdf:resourceパラメーターがPathの場合、ノードAとBがエクスポートされます。 rdf:resourceパラメーターがDescendantsの場合、ノードB、C、Dがエクスポートされます。
customerName （オプション）	このオプション要素を使用すると、指定したパスとそれより下の階層で、このカスタマーが所有するノードのみをエクスポートします。パスで指定したノードの所有者が指定のカスタマーでない場合、エクスポートは実行されず、警告メッセージがログに記録されます。 この要素の使用例については、customerName要素の例 (34ページ) を参照してください。

サービスレベルエクスポートフィルターの例

/Service Level/Fooノードのみをエクスポートします。

```
<ServiceLevelFilter rdf:ID="exportServLevfoo">
  <path>/Service Level/Foo</path>
  <directive rdf:resource="&filter;Node"/>
</ServiceLevelFilter>
```

指定したパスに沿って、Barノードと Bazノードをエクスポートします。スタックルートはエクスポートされない点に注意してください。

```
<ServiceLevelFilter rdf:ID="exportServLevBarBaz">
  <path>/ServiceLevel/Bar/Baz</path>
  <directive rdf:resource="&filter;Path"/>
</ServiceLevelFilter>
```

サービスレベルノードであるGoldとその子孫をすべて、リーフノードを含めてエクスポートします。

```
<ServiceLevelFilter rdf:ID="exportServLevGold">
  <path>/ServiceLevel/Gold</path>
  <directive rdf:resource="&filter;Descendants"/>
</ServiceLevelFilter>
```

スナップショットフィルター

スナップショットフィルターは、DETを使用してSAコア/メッシュからエクスポートするスナップショットを指定します。これにより、エクスポートした内容を別のSAコア/メッシュにインポートできます。

表30　スナップショットフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

表31　スナップショットフィルターのネスト要素

要素	説明
snapshotResultId	エクスポートするスナップショットのID。スナップショット名は一意ではないので、IDの指定が必要です。このIDは、スナップショットブラウザーを開くと、ユーザーインタフェースの最初の画面に表示されます。

例:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
<SnapshotResultFilter rdf:ID="srf1">
  <snapshotResultId>20001</snapshotResultId>
</SnapshotResultFilter>
</rdf:RDF>
```

テンプレートのエクスポートフィルター

テンプレートのエクスポートフィルターは、DETでエクスポートするテンプレートノードを指定します。

表32　テンプレートエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があり、次の形式で指定されます。 rdf:ID="一意の名前"

表33　テンプレートエクスポートフィルターのネスト要素

要素	説明
path （必須）	最上位ノードを基準にした、エクスポートするノードの絶対パス。パスの区切り文字はスラッシュ記号 (/) です。
directive （必須）	空のコンテンツ要素であり、rdf:resourceパラメーターを1つ指定します。このパラメーターは、次の3つの定数の1つを参照します。 • Descendants: 指定したパス (リーフも含む) の子孫をすべてエクスポートします。 • Node: 指定したノードのみをエクスポートします。 • Path: パスのノードに沿ってすべてエクスポートし、他のノードはエクスポートしません。 たとえば、次の構造を持つパスがあるとします。 /Templates/A/B/C/D 次のように指定します。 /Templates/A/B rdf:resourceパラメーターがNodeの場合、ノードBがエクスポートされます。 rdf:resourceパラメーターがPathの場合、ノードAとBがエクスポートされます。 rdf:resourceパラメーターがDescendantsの場合、ノードB、C、Dがエクスポートされます。
customerName （オプション）	このオプション要素を使用すると、指定したパスとそれより下の階層で、このカスタマーが所有するノードのみをエクスポートします。パスで指定したノードの所有者が指定のカスタマーでない場合、エクスポートは実行されず、警告メッセージがログに記録されます。 この要素の使用例については、customerName要素の例 (34ページ) を参照してください。

テンプレートエクスポートフィルターの例

/Templates/Fooノードのみをエクスポートします。

```
<TemplateFilter rdf:ID="exportTemplatesfoo">
  <path>/Templates/Foo</path>
  <directive rdf:resource="&filter;Node"/>
</TemplateFilter>
```

指定したパスに沿って、Barノードと Bazノードをエクスポートします。スタックルートはエクスポートされない点に注意してください。

```
<TemplateFilter rdf:ID="exportTemplatesBarBaz">
  <path>/Templates/Bar/Baz</path>
  <directive rdf:resource="&filter;Path"/>
</TemplateFilter>
```

テンプレートノードであるAlphaとその子孫をすべて、リーフノードを含めてエクスポートします。

```
<TemplateFilter rdf:ID="exportTemplatesAlpha">
  <path>/Templates/Alpha</path>
  <directive rdf:resource="&filter;Descendants"/>
</TemplateFilter>
```

ユーザーグループのエクスポートフィルター

ユーザーグループエクスポートフィルターは、DETでエクスポートするユーザーグループを指定します。このエクスポートでは、各ユーザーグループに関する次の情報がエクスポートされます。

- 名前
- 説明
- SAクライアントの管理者の[ユーザーグループ]ビューの[アクションのアクセス権]ビュー内の各アクションのアクセス権の読み取り、読み取り/書き込み、なし、はい/いいえの状態
- [リソースのアクセス権]ビュー内の各顧客とデバイスグループの読み取り、読み取り/書き込み、なしの状態

表34　ユーザーグループエクスポートフィルターのパラメーター

パラメーター	説明
rdf:ID	各フィルターには一意の名前があります。フィルターの名前は、rdf:ID="一意の名前" という形式で指定します。

表35　ユーザーグループエクスポートフィルターのネスト要素

要素	説明
groupName （オプション）	ユーザーグループの名前を指定してエクスポートします(オプション)。groupNameを指定しない場合、すべてのユーザーグループがエクスポートされます。

注

- ユーザーのメンバーシップとファシリティのアクセス権([ユーザー]タブと[ファシリティ]タブ)はエクスポートの対象ではありません。
- [カスタマー]タブにはすべてのカスタマー、[デバイスグループ]タブにはすべてのサーバーグループが表示され、状態(読み取り、読み取り/書き込み、なし)を設定することができます。エクスポートの対象になるのは、読み取りまたは読み取り/書き込みが設定されているカスタマーとデバイスグループのみです。

ユーザーグループエクスポートフィルターの例

すべてのユーザーグループをメッシュからエクスポートします。

```
<UserGroupFilter rdf:ID="exportAllUserGroups"/>
```

Export the group named "SuperUsers":

```
<UserGroupFilter rdf:ID="exportUserGroups">
  <groupName>SuperUsers</groupName>
</UserGroupFilter>
```

名前が "AdvancedUsers"、"OpswareAdministrators"、"BasicUsers"のユーザーグループをエクスポートします。

```
<UserGroupFilter rdf:ID="exportAdvUsersGroup">
  <groupName>AdvancedUsers</groupName>
</UserGroupFilter>
<UserGroupFilter rdf:ID="exportOpsUsersGroup">
  <groupName>OpswareAdministrators</groupName>
</UserGroupFilter>
<UserGroupFilter rdf:ID="exportBasicUsersGroup">
  <groupName>BasicUsers</groupName>
</UserGroupFilter>
```

customerName要素の例

ここでは、アプリケーション、サービスレベル、テンプレート、サービスグループの各エクスポートフィルターについて、customerName要素を指定した場合の動作を説明します。

次の2つのトピックについて説明します。

- アプリケーション、サービスレベル、テンプレートでのcustomerNameの使用例
- サーバーグループでのcustomerNameの使用例

アプリケーション、サービスレベル、テンプレートでのcustomerNameの使用例

たとえば、次の階層構造を持つノードがあるとします。

```
Service Levels (所有者はCustomer Independent)
      A (Customer Independent)
           B (Customer Independent)
           C (NikeとAdidas)
                D (Nike)
```

- ファイルで次のフィルター定義を行うと、A、B、C、Dがエクスポートされます。つまり、すべてのカスタマーのサービスレベルがエクスポートの対象になります。

```
<ServiceLevelFilter rdf:ID="a1">
    <path>/Service Level/A</path>
    <directive rdf:resource="&filter;Descendants"/>
</ServiceLevelFilter>
```

- ファイルで次のフィルター定義を行うと、AとBがエクスポートされます。
  CとDはエクスポートされません。

```
ServiceLevelFilter rdf:ID="a1">
    <path>/Service Level/A</path>
    <directive rdf:resource="&filter;Descendants"/>
    <customerName>Customer Independent</customerName>
</ServiceLevelFilter>
```

- ファイルで次のフィルター定義を行うと、CとDがエクスポートされます。

```
<ServiceLevelFilter rdf:ID="a1">
    <path>/Service Level/A/C</path>
    <directive rdf:resource="&filter;Descendants"/>
    <customerName>Nike</customerName>
</ServiceLevelFilter>
```

- ファイルで次のフィルター定義を行うと、Cのみがエクスポートされます。Dは、所有者がAdidasではないので、エクスポートの対象にはなりません。

```
<ServiceLevelFilter rdf:ID="a1">
    <path>/Service Level/A/C</path>
    <directive rdf:resource="&filter;Descendants"/>
    <customerName>Adidas</customerName>
</ServiceLevelFilter>
```

- ファイルで次のフィルター定義を行うと、何もエクスポートされません。

```
<ServiceLevelFilter rdf:ID="a1">
<path>/Service Level/A</path>
<directive rdf:resource="&filter;Descendants"/>
<customerName>Nike</customerName>
</ServiceLevelFilter>
```

サーバーグループでのcustomerNameの使用例

サーバーグループフィルターでcustomerName要素を指定した場合の動作について、例を使って説明します。

たとえば、コアのサーバーグループに次のような階層構造があるとします。

```
サーバーグループ
            Public
            SG1
                      + /Application Servers/A (所有者はCustomer Independent)
                        + /System Utilities/B (Nike)
                        + /Web Servers/C (Adidas)
```

- ファイルで次のフィルター定義を行うと、SG1、A、B、Cがエクスポートされます。

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
</ServerGroupFilter>
```

- ファイルで次のフィルター定義を行うと、SG1とAがエクスポートされます。

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
    customerName>Customer Independent</customerName>
</ServerGroupFilter>
```

- ファイルで次のフィルター定義を行うと、SG1とBがエクスポートされます。

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
    <customerName>Nike</customerName>
</ServerGroupFilter>
```

- ファイルで次のフィルター定義を行うと、サーバーグループSG1がエクスポートされます。

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
    <customerName>Acme</customerName>
</ServerGroupFilter>
```

コンテンツのインポート

cbtコマンドを実行することにより、ターゲットのSAメッシュにコンテンツをインポートできます。

コンテンツのインポートでは上位互換性が必要です。つまり、SAの古いバージョンへのインポートはサポートされません。

cbtコマンドの実行可能ファイルは次のディレクトリに格納されています。

```
/opt/opsware/cbt/bin
```

次に、インポートコマンドを示します。

```
cbt -i <コンテンツディレクトリ> -p <ポリシー> -cf <ターゲットコア構成> --noop
```

次に、コマンドと引数を示します。

- コンテンツディレクトリ: エクスポート済みのコンテンツが保存されているディレクトリ
- ポリシー: DETがターゲットのSAメッシュ内で重複を検出する際に使用するインポートポリシー。詳細については、各コンテンツタイプでのインポートポリシー (37ページ) を参照してください。
- ターゲットコア構成: 各種SAコンポーネントにアクセスする際に、コンポーネントの場所と使用するIDをDETに指示する構成ファイル。このファイルの作成方法については、cbtコマンドの設定 (49ページ) を参照してください。
- --noop:「ドライラン」モードでインポートします。このモードでは、データは変更されません。代わりに、普通に実行した場合に発生する変更をサマリーとして出力します。

指定できるすべての引数と説明については、cbtコマンドリファレンス (55ページ) を参照してください。

DETを使用してアプリケーションをインポートする場合、SAメッシュ内のパッケージ名にはサフィックスとして "cbt" が付加されます。次に例を示します。

```
openssh-3.8p1-sol8-sparc-local_cbt796213986
```

各コンテンツタイプでのインポートポリシー

次の表では、コマンドラインで指定するポリシーについて、重複が検出された場合の動作をコンテンツタイプごとにまとめています。

次のいずれかを指定できます。

- `overwrite`: ポリシーを指定しない場合に適用されるデフォルトです。このオプションを指定した場合の動作は、表で示すようにコンテンツタイプごとに異なります。
- `duplicate`: このオプションを指定した場合の動作は、表で示すようにコンテンツタイプごとに異なります。
- `skip`: 重複が検出されると、セッションログにメッセージが記録され、インポート処理を続行します。この動作はすべてのコンテンツタイプで共通です。

指定できるすべての引数と説明については、cbtコマンドリファレンス (55ページ) を参照してください。

表36　DETによるインポートで指定できるポリシーの動作 (コンテンツタイプごと)

コンテンツタイプ	関連するコンテンツタイプ	インポートポリシー (overwrite)	インポートポリシー (duplicate)
アプリケーション構成	アプリケーション構成テンプレート	すべての属性が上書きモードで更新されます。	アプリケーション構成が新しく作成され、"Oldname-cbt-<random>" という名前が割り当てられます。
アプリケーション構成テンプレート		すべての属性が上書きモードで更新されます。	アプリケーション構成が新しく作成され、"Oldname-cbt-<random>" という名前が割り当てられます。
カスタム属性	なし	キーの新規作成と、既存のキーの上書きを行います。これにより、インポートしたキーと既存のキーをまとめることができます。	インストール順序の関係と同じです。
カスタム拡張	なし	新しいバージョンのスクリプトが作成されます	overwriteポリシーと同じです。
カスタムフィールドスキーマ	なし	表示名は、更新されたフィールドのみです。	重複では何も処理を行いません。
カスタマー	なし	重複では何も処理を行いません。	重複では何も処理を行いません。 マルチマスターメッシュとデルタの同期 (46ページ) に、カスタマーのインポートでの注意事項が記載されているので参照してください。

表36 DETによるインポートで指定できるポリシーの動作(コンテンツタイプごと)(続き)

コンテンツタイプ	関連するコンテンツタイプ	インポートポリシー (overwrite)	インポートポリシー (duplicate)
フォルダー	• パッケージ • ソフトウェアポリシー • OSシーケンス • カスタマー	このフォルダーまでのパス上にあるすべてのフォルダーのプレースホルダーが作成されます。このフォルダーの属性はすべて更新されます。既存データについては、フォルダーのコンテンツと関連付けられているカスタマーが上書きされます。--folderaclsを指定すると、既存のユーザーグループへのfolderACLが作成されます。	スキップ:フォルダーは重複しません。
インストールフック	なし	ユニットを参照してください。	ユニットを参照してください。
インストール順序の関係	なし	関係を新規作成し、既存の関係を上書きします。	この操作は親ノードを基準に行われ、親ノードは必ず作成されるので(名前は異なります)、新しい関係も必ず作成されます。
MRL	なし	ソースメッシュと同じ名前のMRLを、ターゲットメッシュに作成します。	overwriteポリシーと同じです。
OS	• カスタム属性 • カスタマー • インストールフック • MRL	ターゲットのSAメッシュ内にある既存ノードを上書きします。ノードIDは変更されません。コンテンツ情報で既存ノードを上書きします。	コンテンツ情報の名前が変更されます。アプリケーション名にサフィックスとして"cbt-<random>"が付加されます。

表36　DETによるインポートで指定できるポリシーの動作(コンテンツタイプごと)（続き）

コンテンツタイプ	関連するコンテンツタイプ	インポートポリシー (overwrite)	インポートポリシー (duplicate)
OSビルド計画	• アプリケーション構成テンプレート • カスタム属性 • カスタマー • デバイスグループ • スクリプト • ソフトウェアポリシー • ZIPパッケージ	カスタム属性は上書きされます。アプリケーション構成テンプレート、スクリプト、ソフトウェアポリシー、パッチポリシー、およびZIPパッケージは上書きされます。OSビルド計画は更新されます。 エクスポート後にビルド計画の名前を変更したり、ビルド計画を別の場所に移動したりすると、エクスポートしたビルド計画は新しいOSビルド計画として元の場所にインポートされます。この場合、名前を変更したり、別の場所に移動したOSビルド計画は変更されないまま残されます。	スキップ: OSビルド計画は複製されません。 OSビルド計画の内容は次のように複製されます。 • アプリケーション構成テンプレート、スクリプト、およびサーバーグループは複製されます • ZIPパッケージは上書きされます • カスタマーはスキップされます • 新しいソフトウェアポリシーが<software_policy>_cbt<random>という名前で作成されます
パッケージ	なし	パッケージがアップロードされると、パッケージタイプが「コンテナ」(LPP、HP-UXデポ、Solaris) のパッケージが上書きされます。このパッケージタイプについては、新しいコンテンツ (パッケージに含まれるコンテンツ) が古いコンテンツのスーパーセットである場合、新しいデータで上書きされます。スーパーセットでない場合、既存の「名前変更」モードに戻ります。 パッケージが別のフォルダーに存在しても、新しいパッケージとして新しいフォルダーにインポートされます。既存のパッケージが新しいフォルダーに移動することはありません。	overwriteポリシーと同じです。

表36　DETによるインポートで指定できるポリシーの動作(コンテンツタイプごと)（続き）

コンテンツタイプ	関連するコンテンツタイプ	インポートポリシー (overwrite)	インポートポリシー (duplicate)
パッチ	なし	物理パッチパッケージがアップロードされ、そこに含まれるユニットがソフトウェアリポジトリに作成されます。 AIX LPPとHPUXデポのパッケージタイプについては、新しいコンテンツ(パッケージに含まれるコンテンツ)が古いコンテンツのスーパーセットである場合、新しいデータで上書きされます。スーパーセットでない場合、既存の「名前変更」モードに戻ります。	overwriteポリシーと同じです。このような処理が行われるのは、Server Automationでは、ソフトウェアリポジトリ内にアップロードしているパッケージと同じパッケージが存在するかどうかを効率的かつ信頼性の高い方法で検出できないためです。
パッチナレッジ (PATCH_META_DATA)	なし	パッチデータベースをServer Automationにインポートすると、既存のデータベースがある場合は上書きします。インポートによって作成されるナレッジは、ターゲットSAメッシュでのパッチ設定に応じて異なります。	overwriteポリシーと同じです。
パッチポリシー	パッチ	説明とパッチリストが更新されます。	パッチポリシーが新しく作成され、"Oldname-cbt<random>" という名前が割り当てられます。
スクリプト[a]	なし	現在のバージョンのインポート済みスクリプトを既存のオブジェクトに追加します。	既存のオブジェクトを新しい名前で複製し、現在のバージョンのインポート済みスクリプトを複製に追加します。
サーバーコンプライアンス条件	なし	すべての属性が上書きモードで更新されます。	サーバーコンプライアンス条件が新しく作成され、"Oldname-cbt<random>" という名前が割り当てられます。
サーバー(デバイス)グループ	• アプリケーション • ソフトウェアポリシー • カスタム属性 • カスタムフィールドスキーマ • パッチ • サーバーグループ • サービスレベル	グループの説明とタイプが更新されます。動的なグループルールは上書きされます。「いずれかのルールに一致」と「すべてのルールに一致」のマッチ設定には、エクスポートで定義された内容が反映されます。カスタム属性は上書きされます。パッチの添付、アプリケーション、サービスレベルは上書きされます。	サーバーグループが新しく作成され、"Oldname-cbt<random>" という名前が割り当てられます。

表36　DETによるインポートで指定できるポリシーの動作(コンテンツタイプごと)（続き）

コンテンツタイプ	関連するコンテンツタイプ	インポートポリシー (overwrite)	インポートポリシー (duplicate)
サービスレベル	• カスタム属性 • カスタマー	ターゲットのSAメッシュ内にある既存ノードを上書きします。ノードIDは変更されません。コンテンツ情報で既存ノードを上書きします。	コンテンツ情報の名前が変更されます。テンプレート名にサフィックスとして"cbt<random>"が付加されます。
テンプレート	• カスタム属性 • カスタマー • アプリケーション • パッチ • OS • サービスレベル	ターゲットのSAメッシュ内にある既存ノードを上書きします。ノードIDは変更されません。コンテンツ情報で既存ノードを上書きします。	コンテンツ情報の名前が変更されます。テンプレート名にサフィックスとして"cbt<random>"が付加されます。
ユニット	ユニットスクリプト	ユニットは物理パッケージに関連付けられています。上記の「パッケージ」のコンテンツタイプを参照してください。仮想ユニットは、ターゲットのSAメッシュ内にある既存ユニットに必ず関連付けられます。これは、インポートセッション中に行う物理パッケージのアップロードによって作成されます。	overwriteポリシーと同じです。
ユニットスクリプト	なし	ユニットスクリプトの新規作成と、既存のユニットスクリプトの上書きを行います。	overwriteポリシーと同じです。
ユーザーグループ	カスタマー、サーバー(デバイス)グループ	ユーザーグループの説明が更新されます。さらに、機能のチェック済み状態([機能]タブと[その他]タブで表示)には、エクスポートした内容が反映されます。カスタマーの設定(読み取り、読み取り/書き込み、なし)、ノードスタック、クライアント機能には、エクスポートした内容が反映されます。サーバーグループの読み取りと読み取り/書き込みの設定も更新されます。	ユーザーグループが新しく作成され、"Oldname-cbt<random>"という名前が割り当てられます。

a. スクリプトは、その親フォルダーをインポートすることでのみインポートできます。

インポートでの削除条件

エクスポートで削除対象としてコンテンツをマークし、--deleteオプションを指定してインポートを行うと、マークした項目はターゲットメッシュから削除されます。

ただし、ターゲットメッシュ内に削除対象としてマークしたコンテンツがあり、これがSAモデルの一部で使用されている場合、DETは 'no harm' のアプローチに基づいて、コンテンツを削除せずに名前を変更します。または、-delオプションを指定してエクスポートを実行した後、-delオプションを指定せずにインポートを実行すると、エクスポートで削除対象としてマークされたコンテンツ項目は、ターゲットメッシュで削除されず、名前が変更されます。

ターゲットメッシュのコンテンツ項目の名前変更では、次の命名規則が適用されます。

<項目名>-cbtDeleted<12345>

たとえば、"foo" という名前のアプリケーション構成は、"foo-cbtDeleted134234" という名前に変更されます。

表37では、インポートでコンテンツ項目を削除するために必要になるすべての条件と、コンテンツ項目の名前が変更されるケースをまとめています。削除に必要な条件を満たしていない場合、命名規則に従って名前が変更されます。

また、エクスポートとインポートのいずれでも、削除オプションを指定すれば必ず削除されるコンテンツ項目や、どのような場合でも削除できないコンテンツ項目があります。

名前の変更後にオブジェクトを検索できなくなるケース

何らかの理由でコンテンツ項目の名前が変更された場合 (-delまたは "do no harm" を指定しない場合)、その後にインポートを実行しようとすると、DETはオブジェクトの場所を特定できなくなります。このようなケースに対応するために、コンテンツ項目の名前が変更された場合には、ターゲットメッシュにある項目の名前も変更されます。

たとえば、DETを実行した結果、"foo" という名前のアプリケーション項目の名前が "foo-cbtDeleted134234" に変更されたとします。その後DETを実行すると、"foo" という名前のアプリケーション構成は存在しないので、DETはアプリケーション構成の名前の再変更や削除を実行できなくなります。

名前の変更が原因で認識できなくなる依存関係を持つオブジェクトには、アプリケーション、アプリケーション構成、アプリケーション構成ファイル、コンプライアンス選択条件、カスタム拡張、分散スクリプト、OS、パッチポリシー、サーバー (デバイス) グループ、サービスレベル、テンプレートなどのタイプがあります。

表37　　-delオプションを指定したインポートでコンテンツ項目の削除に必要な条件

オブジェクトタイプ	削除に必要な条件
アプリケーション構成	アプリケーション構成を使用しているサーバーまたはデバイスグループがない。 アプリケーション構成を使用しているソフトウェアポリシーがない。
アプリケーション構成ファイル	アプリケーション構成ファイルを使用しているアプリケーション構成がない。
コンプライアンス選択条件	常に削除可能。
カスタム拡張	削除不可。常に名前を変更。
カスタムフィールドスキーマ	常に削除可能。

表37　　-delオプションを指定したインポートでコンテンツ項目の削除に必要な条件（続き）

オブジェクトタイプ	削除に必要な条件
カスタマー	アプリケーション、サービスレベル、テンプレートノードがない。 非アクティブなデバイスがない。 パッケージがない(ステータスが「削除」のパッケージを含む)。 IP範囲グループがない。 フォルダーがない。 注: SA内にパッケージがある場合(ステータスが「削除」のものを含む)、カスタマーは削除できません。ステータスが「削除」のオブジェクトは、1つ以上のサーバーで修復を行うためにパッケージが必要な状態、またはサテライトのソフトウェアリポジトリキャッシュ内にまだパッケージが格納されている状態のいずれかです。このような場合、削除対象としてマークされていても、カスタマーは削除されず、名前が変更されます。
デプロイメントステージの値	この値を使用しているデバイスがない。
フォルダー	パッケージ、ソフトウェアポリシー、サブフォルダーがない。
OS	アタッチされているデバイスがない。 子ノードがない。 このノードを含むテンプレートまたはデバイスグループがない。
パッケージ	次に示す削除可能なユニットタイプ このパッケージを使用するSolarisパッチクラスターまたはMRLがない。 このパッケージを使用するソフトウェアポリシーがない。 ZIPパッケージの場合、再配置可能な子のZIPがない。 このパッケージを使用するOS定義またはアプリケーションノードがない。 このパッケージを使用するソフトウェアポリシーがない。 パッチの場合: — パッチノードにアタッチされているデバイスがない。 — このパッチノードを含むテンプレートまたはデバイスグループがない。 — このパッチノードを含むパッチポリシーまたはパッチ例外がない。 LPP、HPUXデポ、Solarisパッケージの場合: — ソフトウェアポリシーが使用しているサブパッケージがない。 — サブパッケージを使用するOS定義またはアプリケーションノードがない。 — パッチであるサブパッケージの場合: – パッチノードにアタッチされているデバイスがない。 – このパッチノードを含むテンプレートまたはデバイスグループがない。 – このパッチノードを含むパッチポリシーまたはパッチ例外がない。 削除可能なパッケージユニットのタイプ*(この表の下のリストを参照)
パッチポリシー	アタッチされているデバイスがない。 アタッチされているデバイスグループがない。

表37　-delオプションを指定したインポートでコンテンツ項目の削除に必要な条件（続き）

オブジェクトタイプ	削除に必要な条件
サーバー(デバイス)グループ	アタッチされているデバイスがない。 子ノードがない。 アクセス制御によって使用されていない。 動的にバインドされているジョブがない。
サーバーの用途の値	この値を使用しているデバイスがない。
サービスレベル	アタッチされているデバイスがない。 子ノードがない。 このノードを含むテンプレートまたはデバイスグループがない。
テンプレート	子ノードがない。
ユーザーグループ	常に削除可能。

* 検出可能なパッケージユニットのタイプ:

- HOTFIX
- HPUX_DEPOT
- LPP
- MRL
- MSI
- PROV_INSTALL_HOOKS
- RPM
- SERVICE_PACK
- SOL_PATCH
- SOL_PATCH_CLUSTER
- SOL_PKG
- SP_RESPONSE_FILE
- UNKNOWN
- UPDATE_ROLLUP
- WINDOWS_UTILITY
- ZIP

カスタマーのインポートでの注意事項

現在DETでは、カスタマーに関連付けられているユーザーグループのアクセス権をエクスポートすることはできません。ただし、エクスポートするカスタマーの名前が、ターゲットメッシュ(インポート先メッシュ)内のカスタマーの名前と同じ場合にはエクスポートが可能です。

たとえば、ソースメッシュ内にiPlanetというソフトウェアアプリケーションノードがあるとします。Computing Machinesという名前のカスタマーに関連付けられているすべてのグループは、iPlanetに対して読み取りと書き込みを実行できます。さらに、カスタマーであるComputing Machinesに関連付けられているグループの1つにgroupAがあるとします。

では、アプリケーションノードiPlanetをソースメッシュからエクスポートし、新しいメッシュにインポートする操作を考えてみましょう。インポート先のメッシュにComputing Machinesという名前のカスタマーが存在しない場合、ターゲットメッシュにソフトウェアアプリケーションノードiPlanetはインポートされますが、groupA内のユーザーはこのノードを認識できません。

これに対して、Computing Machinesという名前のカスタマーが存在するメッシュにインポートする場合には、新しいメッシュですべてのアクセス権がそのまま保持されるので、groupAのすべてのユーザーはソフトウェアアプリケーションノードであるiPlanetにアクセス可能になります。つまり、カスタマーであるComputing Machinesに割り当てられているアクセス権(ソフトウェアノードiPlanetに対する読み取りと書き込み)はすべて保持されます。

カスタマーのインポートで発生した問題の回避方法

ユーザーグループがオブジェクト(カスタマーに関連付けられたサーバーなど)にアクセスできなくなったら、SAクライアントを使用してアクセス権を再度割り当てることができます。再割り当てを行うまでの間は、カスタマーと関連オブジェクトを表示できるのは管理者権限を持つユーザーに限定されます。

マルチマスターメッシュとデルタの同期

DETでは、「増分」エクスポートとインポートを実行できます。これにより、マルチマスターメッシュのコンテンツを同期し、最新の状態に更新することが可能になります。

たとえば、「ソース」メッシュから、他のメッシュに反映したいコンテンツをすべてエクスポートする処理を定期的に実行することができます。このリリースで新しく追加されたオプションでは、変更または削除されたコンテンツのみをエクスポートできるので、ターゲットメッシュとソースメッシュを整合状態に維持することができます。

デルタエクスポート

デルタエクスポートを実行するには、次のコマンドラインオプションを使用します。

- `--baseline` (省略形: `-b`)

  実行するエクスポートの比較対象となるベースラインエクスポートを指定します。エクスポートで`--incremental`または`--delete`のいずれかを指定する必要があります。

- `--incremental` (省略形: `-incr`)

  フィルターファイルで指定するコンテンツの中で、ベースライン以降に追加または変更されたコンテンツのみをエクスポートします。このオプションを指定しない場合、フィルターファイルで指定したすべてのコンテンツがエクスポートされます。このオプションを指定する場合は、`--baseline`も指定する必要があります。

- `--delete` (省略形: `-del`)

  フィルターファイルで指定(「削除」とマーク)されていないベースラインコンテンツをすべてエクスポートします。このオプションを指定しないと、「削除」とマークされているコンテンツはエクスポートされません。このオプションを指定する場合は、`--baseline`も指定する必要があります。

`--delete`と`--incremental`を`--baseline`と組み合わせてエクスポートを実行すると、次のような結果になります。

- 増分エクスポートのオプションを指定しない場合。

  フィルターファイルで指定したコンテンツがすべてエクスポートされます。

- `-incr`

  フィルターファイルで指定したコンテンツの中で、ベースライン以降に追加または変更されたコンテンツがすべてエクスポートされます。

- `-del`

  フィルターファイルで指定されているすべてのコンテンツ(`-incr`を指定していないため)と、フィルターファイルで指定(「削除」)されていないベースラインコンテンツがすべてエクスポートされます。

- `-incr -del`

  フィルターファイルで指定されているコンテンツの中で、ベースライン以降に追加または変更されたすべてのコンテンツと、フィルターファイルで指定(「削除」)されていないベースラインコンテンツがすべてエクスポートされます。

デルタインポート

デルタインポートを実行するには、次のコマンドラインオプションを指定します(オプションを指定してエクスポートを実行した場合)。

- `--delete` (省略形:`-del`)

  `--baseline`オプションと`--delete`オプションを指定してエクスポートした後、`--delete` オプションを指定してインポートする場合、エクスポートで削除対象として指定したオブジェクトが削除されます。

--baselineオプションと--deleteオプションを指定してエクスポートした後、--deleteを指定せずにインポートすると、削除対象として指定された項目は削除されず、名前が変更されます。削除不可のコンテンツや必ず名前が変更されるコンテンツ(メッシュ内にオブジェクトの依存関係が存在する場合など)の詳細については、インポートでの削除条件 (42ページ)を参照してください。

メッシュ同期の使用シナリオ

ここでは、コンテンツが削除および変更されているソースメッシュで増分によるエクスポートとインポートを実行する場合、一般的な実行サイクルの例を紹介します。

- 最初に、アプリケーション構成コンテンツをエクスポートするフィルターで全体をエクスポートします。

  ```
  cbt -e content/appConfig.0 -f ac_Filter.rdf -cf meshA_Config
  ```

- エクスポートしたコンテンツを別のメッシュにインポートします。

  ```
  cbt -i content/appConfig.0 -p overwrite -cf meshB_Config
  ```

  ソースメッシュ内のコンテンツが変更または削除されます。

- ソースメッシュから変更されたコンテンツと削除されたコンテンツをエクスポートするために、-b、-incr、-delを指定します。

  ```
  cbt -e content/appConfig.1 -f ac_Filter.rdf -b content/appConfig.0 -incr -
  del -cf meshA_Config
  ```

- ターゲットメッシュにデルタをインポートします。変更されたコンテンツを更新し、削除されたコンテンツを削除します。

  ```
  cbt -i content/appConfig.1 -p overwrite -cf meshB_Config -del
  ```

- 最新のエクスポートをベースラインとして指定して手順4と5を実行することによって、コンテンツを更新します。たとえば、次のサイクルでは次のように指定します。
  - -b content/appConfig.1を指定して、content/appConfig.2をエクスポートします。
  - content/appConfig.2をインポートします。

コンテンツディレクトリ

コンテンツディレクトリとは、エクスポートしたSAコンテンツの永続的なストアです。コンテンツディレクトリには次の内容が格納されています。

- data.rdf: エクスポートしたSA構成コンテンツのデータベース。
- filter.rdf: ユーザー作成のフィルターとDETによって生成されたフィルターのデータベース。
- blob/: エクスポートしたソフトウェアパッケージとスクリプトを保存するディレクトリ。
- var/: 過去10回分のインポートおよびエクスポートセッションのログを保存するディレクトリ。ログにはcbtexport {0-9}.logおよびcbtimport {0-9}.logという名前が割り当てられます。10個のセッションログのうち、0が最新のログで、9が最も古いログです。

次に、コンテンツディレクトリの例を示します。

```
% ls -R
.:
blob
data.rdf
```

```
filter.rdf
var

./blob:
unitid_140270007.pkg
unitid_166510007.pkg
unitid_166540007.pkg
unitid_2090007.pkg

./var:
cbtexport0.log
cbtexport0.log.lck
cbtimport0.log
```

セッションの例

管理対象サーバー上で、cbtコマンドをrootユーザーで実行するように事前に設定しておきます。この構成では、SAユーザー名とパスワードを指定するだけでエクスポートまたはインポートを実行できます。

次に、セッションの実行例を示します。この例は、ユーザーにインポートとエクスポートの実行権限が割り当てられていることが前提となります。詳細については、cbtコマンドの設定 (49ページ) を参照してください。

コマンドセンターサーバーでcshセッションを実行します。

```
% setenv JAVA_HOME <j2re 1.4.x installation>
% /opt/opsware/cbt/bin/cbt -e /tmp/foo -f \
/opt/opsware/cbt/filters/app.rdf \
--spike.username hermaime
Enter password for hermaime: ********
...
```

cbtコマンドは次のディレクトリに格納されています。

```
/opt/opsware/cbt/bin
```

cbtコマンドのインストール

cbtコマンドはプレインストールされているので、SAコアサーバー上ですぐに実行可能です。

ほかのUnixサーバーでcbtコマンドを使用するには、以下で説明する手順に従って手動でコマンドをインストールする必要があります。cbtコマンドは、ソフトウェアポリシーを作成すると、Unixベースの管理対象サーバーに自動的にインストールされます。

cbtコマンドは、SAメッシュへのネットワークアクセスが可能なUnixコンピューターで実行できます。Windowsプラットフォームではcbtコマンドはサポートされませんが、Windowsコンテンツのインポートとエクスポートは可能です。

SAコア以外のUnixサーバーにcbtコマンドをインストールするには、次の手順を実行します。

1 cbtコマンドのインストール先となるUnixサーバーにrootでログインします。このサーバーには、SAメッシュへのアクセスが必要です。

2 cbtアーカイブ (cbt-<バージョン>.zip) は、HPSAインストールメディアまたはSAコアに保存されています。

a SAコアでは、次のフォルダーにあるSAライブラリにcbtアーカイブが保存されています。Opsware/Tools/CBT

b HP Server Automationインストールメディアでは、packagesサブディレクトリにcbt-<バージョン>.zipファイルが保存されています。

3 cbtアーカイブを、インストール先ディレクトリにコピーします。

4 アーカイブを解凍します。

5 Java Runtime Environment (JRE) を構成します。

a このファイルがない場合は、JRE 1.4.xまたはJDK 1.6.x以降をwww.oracle.comからダウンロードし、ログインしたサーバーにインストールします。

b JAVA_HOME環境変数を変更し、Javaインストールディレクトリをポイントします。たとえば、cshで次のコマンドを発行します。

```
% setenv JAVA_HOME <javaインストール>
```

c オプションで、PATH環境変数にcbtインストールディレクトリを含める設定を行うことができます。

```
<cbtインストールディレクトリ>/bin
```

6 cbtコマンドを実行可能であることを確認するには、次のコマンドを入力します。

```
% cd <cbt インストールディレクトリ >/bin
% ./cbt -v
```

-vオプションを指定すると、コマンドのバージョン文字列が表示されます。

cbtコマンドを実行すると、次のエラーが発生することがあります。

```
Error occurred during initialization of VM
Could not reserve enough space for object heap
```

このエラーが発生したら、cbtスクリプトのjargsにある-Xmxオプションの値を小さい値 (-Xmx512mなど) に変更します。

cbtコマンドの設定

ここでは、cbtコマンドを構成する方法について説明します。cbtコマンドは、SAコアサーバーの次のディレクトリにインストールされています。

```
/opt/opsware/cbt/bin
```

SAコアの外部からのcbtコマンドの実行については、SAコア以外のUsNIXホスト上でのcbtコマンドの実行 (50ページ) を参照してください。

cbtコマンドは、SAメッシュへのネットワークアクセスが可能なUnixコンピューターで実行できます。Windowsプラットフォームではcbtコマンドはサポートされませんが、Windowsコンテンツのインポートとエクスポートは可能です。

cbtコマンドはJavaで記述され、スキーマ定義と永続的なストアにOWLおよびRDFを使用します。SAコンテンツのインポートとエクスポートでは、SA APIを使用して構成とサイズの大きなバイナリコンテンツ (パッケージやスクリプトなど) の両方を抽出します。

1 cbtコマンドを実行可能であることを確認するには、次のコマンドを入力します。

```
% cd <cbtインストールディレクトリ>/bin
% ./cbt -v
```

-vオプションを指定すると、コマンドのバージョン文字列が表示されます。

cbtコマンドを実行すると、次のエラーが発生することがあります。

```
Error occurred during initialization of VM
Could not reserve enough space for object heap
```

このエラーが発生したら、cbtスクリプトのjargsにある-Xmxオプションの値を小さい値 (-Xmx512mなど) に変更します。

2 インポート先またはエクスポート元となるメッシュごとに、次の手順を実行します。

a opsware-ca.crt信用証明書のコピーを /var/opt/opsware/crypto/twist/opsware-ca.crtから取得し、cbtコマンドでアクセス可能な場所に保存します。この手順は、SAコマンドセンターコアコンポーネントがインストールされているサーバーからcbtを実行する場合に、オプションで実行します。

b spog.pkcs8クライアントと証明書のコピーを/var/opt/opsware/crypto/twist/spog.pkcs8から取得し、cbtコマンドでアクセス可能な場所に保存します。この手順は、SAコマンドセンターコアコンポーネントがインストールされているサーバーからcbtを実行する場合に、オプションで実行します。

c Webサービスデータアクセスエンジン（twist）のユーザー名とパスワードをSA管理者から取得します。これは、Webサービスデータアクセスエンジン (twist) のインストール時に設定されます。

d ターゲットメッシュの構成ファイルを作成します。ファイルでは、SAメッシュコンポーネントへのアクセスに必要な場所とIDの情報を指定します。詳細については、次の説明を参照してください。

SAコア以外のUsNIXホスト上でのcbtコマンドの実行

SAコア以外のUNIXホストからcbtコマンドを実行するには、NSS_FIPS_ENABLED環境変数の設定が必要になります。SAコアをFIPS準拠モードで実行している場合、NSS_FIPS_ENABLED環境変数を1に設定します。

UNIXホストからのcbtコマンドの実行例を以下に示します。

```
export NSS_FIPS_ENABLED=1 && <cbtインストールディレクトリ>/bin/cbt -i <コンテンツディレクトリ> -p <ポリシー> -cf <ターゲットコア構成>
```

ターゲットメッシュの構成ファイルの作成

ターゲットメッシュの構成ファイルを作成すると、DETを簡単に操作できるようになります。デフォルトの構成ファイルのサンプルは、DETのインストール時に次の場所に保存されています。

```
cbt/cfg/default.properties
```

メッシュ構成ファイルは、キーと値のペアを等号で結んだテキストファイルであり、通常はDETコマンドラインで指定するSAコンポーネントのアクセス情報が保存されています。DETメッシュ構成ファイルのパラメーターを指定する場合は、このファイルのコピーを作成し、任意の場所に保存してください。

構成ファイルにはユーザー名とパスワードが保存されているので、セキュリティに注意してください。

表38では、DET構成に関連するすべてのプロパティをまとめています。このプロパティは、DETコマンドラインで指定するか、構成ファイルで指定することができます。

表38で示す構成プロパティのデフォルト値は、コマンドセンターコアコンポーネントを実行するSAメッシュでDETを実行する場合の値です(したがって、.hostプロパティはlocalhostになっています)。また、twist.certpaths、ssl.trustcerts、ssl.keypairsはコマンドセンターサーバー上のパスが想定されています。

メッシュ構成ファイルで指定しない構成関連プロパティについては、次の構成プロパティの表で示すデフォルト値が適用されます。

表38　　構成プロパティ

プロパティ名	デフォルト値	説明
cbt.numthreads	1	エクスポートで使用する同時スレッドの数。 コンテンツのエクスポートでは、任意の数のスレッドを指定できます。 ただし、インポートでDETがサポートできるスレッド数は1つのみです。
spike.enabled	true	XML-RPCベースのすべてのサーバーでの認証と承認にSpikeを使用します。
spike.host	way	Spikeのホスト名またはIP。
spike.path	wayrpc.py	SpikeのベースURLパス。
spike.password	<デフォルトなし>	Spike認証で使用するユーザーパスワード。これはOCCユーザーパスワードであり、メッシュのインストール時に設定されます。パスワードの取得については、SA管理者(またはメッシュのインストール担当者)にお問い合わせください。
spike.port	1018	Spikeのリスナーポート。
spike.protocol	https	Spikeのリスナープロトコル。通常はHTTPSです。
spike.username	admin	Spike認証で使用するユーザー名。cbtpermツールでアクセス権を割り当てたユーザー。 このユーザー名は、オブジェクトの作成または変更を実行できる管理者アカウントでなければなりません。DETのデフォルト構成によってspike.usernameはadminに設定されます。
spin.host	spin	データアクセスエンジンのホスト名またはIP。
spin.path	spinrpc.py	データアクセスエンジンのベースURLパス。
spin.port	1004	データアクセスエンジンのリスナーポート。

表38　構成プロパティ（続き）

プロパティ名	デフォルト値	説明
spin.protocol	http	データアクセスエンジンのリスナープロトコル。DETがSAコマンドセンターと同じサーバー上にあり、マルチサーバーメッシュでクリアテキストSpinを実行している場合は　HTTP、これ以外の構成ではHTTPSです。
ssl.keyPairs	/var/opt/opsware/crypto/twist/spog.pkcs8	カンマ区切りのクライアント証明書リストであり、XML-RPCベースサーバーとの通信で使用されます。
ssl.trustCerts	/var/opt/opsware/crypto/twist/opsware-ca.crt	カンマ区切りの信頼証明書ファイルのリストであり、XML-RPCベースサーバーとの通信で使用されます。
ssl.useHttpClient	true	JDK付属のHTTPクライアントではなくHTTPクライアントライブラリを使用します。
twist.certPaths	/var/opt/opsware/crypto/twist/opsware-ca.crt	カンマ区切りの信頼証明書リストであり、Webサービスデータアクセスエンジンとの通信で使用されます。
twist.host	localhost	Webサービスデータアクセスエンジンのホスト名またはIP。
twist.password	<デフォルトなし>	Webサービスデータアクセスエンジンのパスワード。このパスワードは、メッシュのインストール時に設定されます。パスワードの取得については、SA管理者（またはメッシュのインストール担当者）にお問い合わせください。
twist.port	1032	Webサービスデータアクセスエンジンのリスニングポート。
twist.protocol	t3s	Webサービスデータアクセスエンジンのプロトコル。t3またはt3sです。
twist.username	detuser	Webサービスデータアクセスエンジンのユーザー名。“detuser”を使用します。このアカウントはシステムアカウントであり、パスワードはメッシュのインストール時に設定されます。
way.host	way	コマンドエンジンのホスト名またはIP。
way.path	wayrpc.py	コマンドエンジンのベースURLパス。
way.port	1018	コマンドエンジンのリスナーポート。
way.protocol	https	コマンドエンジンのリスナープロトコル。通常はHTTPSです。

表38　構成プロパティ（続き）

プロパティ名	デフォルト値	説明
word.host	word	ソフトウェアリポジトリのホスト名またはIP。SA 7.80では、ソフトウェアリポジトリはSliceコンポーネントバンドルに含まれます。
word.path	wordbot-new.py	ソフトウェアリポジトリのベースURLパス。
word.port	1003	ソフトウェアリポジトリのリスナーポート。
word.protocol	https	ソフトウェアリポジトリのリスナープロトコル。これはHTTPSです。
mail.transport.protocol	smtp	メールサーバーで使用するメール転送プロトコル。
mail.smtp.host	smtp	メールサーバーのホスト名。
mail.smtp.port	25	メールサーバーが使用するポート番号。
mail.from	<currentuser>@<currenthost>	通知の電子メールの[差出人]フィールドで使用するアドレス。

次にターゲットメッシュの構成ファイルの例を示します。このファイルでは、重要なメッシュ構成情報のみが指定されています。

```
twist.host=twist.c07.dev.opsware.com
twist.port=1032
twist.protocol=t3s
twist.username=<detuser>
twist.password=<twist_password>
twist.certPaths=<opsware-ca.crtの絶対パス>

spike.username=<OCCユーザー>
spike.password=<OCCユーザーパスワード>
spike.host=way.c07.dev.opsware.com
way.host=way.c07.dev.opsware.com
spin.host=spin.c07.dev.opsware.com
word.host=theword.c07.dev.opsware.com

ssl.keyPairs=<spog.pkcs8の絶対パス>
ssl.trustCerts=<opsware-ca.crtの絶対パス>

mail.transport.protocol=smtp
mail.smtp.host=mail
mail.smtp.port=44
mail.from=joe_user@yourcompany.com
```

第3章 cbtコマンドリファレンス

ここでは、cbtコマンドとそのオプションについて説明します。cbtコマンドは、SAコアサーバーの次のディレクトリにあります。

/opt/opsware/cbt/bin

エクスポートオプション (-e)

エクスポートオプションは次の構文を使用します。

```
cbt -e <content_dir> [<options>]
```

表39 エクスポートオプション

短いオプション	長いオプション	description
-e <content_dir>	--export <content_dir>	SA コアから SA データをエクスポートし、指定されたコンテンツディレクトリに保存します。
-f <filter_file>	--filter <filter_file>	初回のエクスポート時は、エクスポートするデータを記述したフィルターファイルを指定する必要があります。2回目以降は、フィルターが指定されていなければ、その前に使用したコンテンツディレクトリ内のフィルターが使用されます。DET フィルターの詳細については、「例：エクスポートフィルターファイル (14 ページ)」を参照してください。
-b <content_dir>	--baseline <content_dir>	実行するエクスポートの比較対象となるベースラインエクスポートを指定します。このオプションを指定するには、エクスポート時に、 --incremental または --delete を指定する必要があります。
-incr	--incremental	増分エクスポートを実行します。フィルターファイルで指定するコンテンツの中で、ベースライン以降に追加または変更されたコンテンツのみをエクスポートします。このオプションを指定しない場合、フィルターファイルで指定したすべてのコンテンツがエクスポートされます。

表39　エクスポートオプション（続き）

短いオプション	長いオプション	description
-cf <file>	--config <file>	DET構成ファイルを指定します。詳細については、ターゲットメッシュの構成ファイルの作成 (50 ページ) を参照してください。
-c	--clean	-e で指定されたコンテンツディレクトリから、以前エクスポートされたデータを削除します。
-d	--debug	詳細なデバッグ情報を表示します。
-del	--delete	フィルターファイルに指定のないベースラインコンテンツを、「削除済み」とマークして、エクスポートに含めます。このオプションを指定しないと、「削除」とマークされているコンテンツはエクスポートされません。 このオプションを指定するには、 --baseline でベースラインエクスポートを指定する必要があります。
-np	--noprogress	コンソールに進行状況を表示しません。
-nd	--nodownload	ソフトウェアリポジトリ (word) からユニットをダウンロードしないようにします。重要: このオプションを使用して行ったエクスポートはインポートできません。
-lx	--logxml	XML形式のログファイルを作成します。
-em <addrs>	--email <addrs>	カンマで区切ったアドレスのリストに、エクスポートサマリーを電子メールで送ります。このオプションを有効にするには、DET構成ファイルに電子メール通知のパラメーターを追加している必要があります。
（なし）	--emaillog	電子メールにログファイル全体を含めます。

インポートオプション (-i)

インポートオプションは次の構文を使用します。

```
cbt -i <content_dir> [<options>]
```

表40　インポートオプション

短いオプション	長いオプション	description
-i <content_dir>	--import <content_dir>	指定されたコンテンツディレクトリから SA データをインポートします。
-p overwrite|duplicate|skip	--policy overwrite|duplicate|skip	インポートポリシーです。デフォルトは "overwrite" です。 "overwrite" は、ターゲット Server Automation 上の同じ名前空間のオブジェクトを、オブジェクト ID を変更せずに上書きします。 “duplicate” は、ターゲット Server Automation で同一のファイルが見つかった場合に、オブジェクトを複製して、それに統合的な名前を付けます。 “skip” は、最も保守的なポリシーです。ターゲット Server Automationに同じオブジェクトが見つかった場合、インポートしません。 コンテンツタイプごとのインポートポリシーの詳細については、「各コンテンツタイプでのインポートポリシー (37ページ)」を参照してください。
-del	--delete	エクスポートによって「削除済み」としてマークされたオブジェクトを削除します。つまり、このオプションが有効なのは、エクスポート時に -del オプションを指定した場合だけです。このオプションを指定しない場合、削除済みオブジェクトの名前が変更されます。

表40　インポートオプション（続き）

短いオプション	長いオプション	description
-fa	--folderacls	インポートしたフォルダーを既存のユーザーグループに関連付けます。 このオプションを指定しない場合、ターゲットメッシュの親フォルダーの ACL がインポートしたフォルダーに継承されます。 このオプションを指定した場合は、フォルダーをインポートする際にACLのインポートを試みます。ACLがインポートされるのは、ソースメッシュのユーザーグループと同じ名前のユーザーグループが、すでに存在しているか、または現在実行中の DET でインポート済みである場合だけです。インポートされた ACL は、同じ名前の既存のユーザーグループに関連付けられます。フォルダーの挿入時は、ターゲットメッシュの親フォルダーから継承された ACL はインポートしたACLで置き換えられます。フォルダーの更新時は、インポートしたACLが既存のACLを上書きします。
-n	--noop	「ドライラン」モードでインポートを実行します。このモードでは、データは変更されません。代わりに、普通に実行した場合に発生する変更をサマリーとして出力します。
-cf <file>	--config <file>	指定したファイルから構成を読み込みます。
-d	--debug	詳細なデバッグ情報を表示します。
-np	--noprogress	コンソールに進行状況を表示しません。
-nu	--noupload	変更されていないパッケージをソフトウェアリポジトリ (word) にアップロードしないようにします。 パッケージが上書きされたと表示されますが、実際はパッケージに変更は加えられていません。ユニットレコードが更新されただけです。
-lx	--logxml	XML 形式のログファイルを作成します。

表40　インポートオプション（続き）

短いオプション	長いオプション	description
-em <addrs>	--email <addrs>	カンマで区切ったアドレス宛てに、インポートサマリーを電子メールで送ります。このオプションを有効にするには、DET 構成ファイルに電子メール通知のパラメーターを追加している必要があります。
（なし）	--emaillog	電子メールにログファイル全体を含めます。

エクスポートステータス表示オプション (-t)

エクスポートステータス表示オプションは次の構文を使用します。

```
cbt -t <content_dir>
```

表41　エクスポートステータスの表示オプション

短いオプション	長いオプション	description
-t	--showstatus	指定されたコンテンツディレクトリのエクスポートステータスを表示します。

構成ファイルOption (-s)

構成ファイルオプションは次の構文を使用します。

```
cbt -s [-cf <file>]
```

表42　構成ファイルオプション

短いオプション	長いオプション	description
-s	--showconfig	現在の構成値を表示します。
-cf <file>	--confi <file>	指定したファイルから構成を読み込みます。

バージョン表示オプション (-v)

バージョン表示オプションは次の構文を使用します。

```
cbt -v
```

表43　バージョン表示オプション

短いオプション	長いオプション	description
-v	--version	DETツールのバージョンを表示します。

ヘルプ表示オプション (-h)

ヘルプ表示オプションは次の構文を使用します。

```
cbt -h
```

表44　ヘルプ表示オプション

短いオプション	長いオプション	description
-h	--help	ヘルプメッセージを表示します。

DETアクセス権コマンド、cbtperm

cbtpermコマンドは、DETの使用に関するアクセス権を設定します。cbtpermコマンドは次の構文を使用します。

```
cbtperm -u [user] -a [spike.username] -p [spike.port] -s [spike.host] -c
[ssl.trustCerts] -k [ssl.keyPairs]
```

表45　DETアクセス権コマンドオプション

短いオプション	長いオプション	description
-u	該当しない	DCML Exchange Tool の使用権限を与えるユーザーです。
-a	--spike.username	Spike認証用のユーザー名です (例: SA Administrator)。
-p	--spike.port	Spikeのリスナーポートです。
-s	--spike.host	Spikeのホスト名またはIPです。
-c	--ssl.trustCerts	XML-RPCサーバーとの通信に使用する、カンマで区切られた信頼できる証明書ファイルのリストです。
-k	--ssl.keyPairs	XML-RPCサーバーとの通信に使用する、カンマで区切られたクライアント証明書のリストです。

第4章 IDKの概要

IDKとISMの概要

Server Automationには、インテリジェントソフトウェアモジュール (ISM) 開発キット (IDK) が付属します。IDKには、ISMの作成、ビルド、アップロードを行うコマンドラインツールとライブラリが含まれています。ISMとは、アプリケーションビット、インストールスクリプト、コントロールスクリプトを含むファイルおよびディレクトリ群です。ISMは、ローカルファイルシステムでビルドし、Server Automationアプリケーションポリシーにアップロードします。アップロードが完了したら、HP Server Automationクライアントを使用してISMのアプリケーションを管理対象サーバーにインストールできます。

IDKの利点

IDKには、次のような利点があります。

- ソフトウェア製品の管理に関するベストプラクティスをまとめることにより、複雑なデータセンター環境で一貫性のある安定したソフトウェアビルドを配信し、変更を管理することができます。
- モジュールをServer Automationにアップロードすると、すぐに管理対象サーバーにインストール可能になります。
- アプリケーションのインストールスクリプトとコントロールスクリプトを、インストールするアプリケーションと分離できます。スクリプトの更新時に、アプリケーションを再インストールする必要がありません。
- Server Automationでカスタム属性のクエリを実行することにより、動的な構成が可能です。
- バイナリアーカイブからネイティブパッケージ (RPMなど) を自動的にビルドします。
- Unixプラットフォームで、共通の仕様形式に基づくソースコードからのビルドをサポートします。
- シェル環境でパッケージのビルドやインストールスクリプト記述を行う開発者と管理者向けのコマンドラインツールを提供します。

IDKツールと環境

IDKでは、次のツールが提供されています。

- ISMTool: ISMの作成、ビルド、アップロードを行うコマンドラインツール。
- ISMUserTool: ISMのアップロードを許可するユーザーを指定するコマンドラインツール。
- 環境変数: ISMToolからアクセス可能なシェル環境変数。
- 実行時ライブラリ: IDKツールをサポートするServer Automationルーチン。

サポート対象のパッケージタイプ

IDKでは、次のタイプのパッケージを作成できます。

- AIX LPP

- HP-UXデポ
- RPM
- Solarisパッケージ
- Windows MSI
- ZIP (WindowsおよびUnix)

IDKのインストール

IDKのインストールと実行は、管理対象サーバー(サーバーエージェントを実行するサーバー)で行うことをお勧めします。手順については、管理対象サーバーでのIDKのインストール (62ページ)を参照してください。サーバーエージェントの詳細については、『SAユーザーガイド: Server Automation』を参照してください。

IDKはコアサーバーにもインストールできますが、注意が必要です。コアコンポーネントは、CRYPTO_PATH環境変数をIDKツールと共有します。したがって、CRYPTO_PATH環境変数を誤って設定してしまうと、コアコンポーネントが稼働できなくなる可能性があります。

IDKは非管理対象サーバー(コアコンポーネントやエージェントを実行していないサーバー)にもインストールできますが、IDK機能に制限が発生します。たとえば、ISMのビルドは可能ですが、CRYPTO_PATH環境変数を設定しないとアップロードはできません。この環境変数の詳細については、CRYPTO_PATH (108ページ)を参照してください。非管理対象サーバーでのIDKのインストール (63ページ)を参照してください。

管理対象サーバーでのIDKのインストール

IDKとISMToolを管理対象サーバーにインストールするには、次の手順を実行します。

1. IDKを実行する管理対象サーバーを選択します。
2. IDKのインストール先となるホストと、ISMアプリケーションのインストール先となる管理対象サーバーで、同じバージョンのオペレーティングシステムが稼働していることを確認します。

   たとえば、Redhat Linux 7.3 管理対象サーバー上にインストールするISMアプリケーションを作成する場合、Redhat Linux 7.3システムにIDKをインストールします。
3. Redhat Linux Application Server、Enterprise Server、Workstationのいずれかに IDKをインストールする場合、`rpm-build`パッケージがインストールされていることを確認してください。このパッケージの有無を確認するには、次のコマンドを入力します。

   ```
   rpm -qa | grep rpm-build
   ```
4. SolarisゾーンにIDKをインストールする場合、`/usr/local`ディレクトリが存在し、書き込みアクセスが割り当てられていることを確認します(このディレクトリは、スパースルートゾーンには存在しないことがあります)。この確認作業は、Server Automationまたは次の`zonecfg`コマンドで実行できます。pathにはグローバルゾーン上のファイルシステムを指定します。

   ```
   zonecfg:ゾーン名:fs> add fs
   zonecfg:ゾーン名:fs> set dir=/usr/local
   zonecfg:ゾーン名:fs> set special=path
   zonecfg:ゾーン名:fs> set type=lofs
   ```
5. SAクライアントでは、名前に“ismtool”が含まれているソフトウェアポリシーを検索します。
6. ソフトウェアポリシーのリストが表示されたら、IDKを実行するプラットフォームのポリシーを右クリックし、[サーバーのアタッチ]を選択します。
7. [サーバーのアタッチ]ウィンドウで、IDKを実行する管理対象サーバーを選択します。

8 [サーバーをただちに修復] チェックボックスがオンになっていることを確認します。

9 [アタッチ] をクリックします。

10 Unixの場合: ターミナルウィンドウで、IDKのインストール先となるホストにrootでログインし、PATH環境変数を次の値に設定します。

```
/usr/local/ismtool/bin
```

(Windowsの場合、PATHは自動設定されますが、設定を有効にするには再度ログインが必要です)。

11 ターミナルウィンドウで次のコマンドを入力し、IDKのインストールをチェックします。

```
ismtool --myversion
```

非管理対象サーバーでのIDKのインストール

IDKのインストールと実行は、管理対象サーバー(サーバーエージェントを実行するサーバー)で行うことをお勧めします。手順については、管理対象サーバーでのIDKのインストール (62ページ) を参照してください。

IDKは非管理対象サーバー(コアコンポーネントやエージェントを実行していないサーバー)にもインストールできますが、IDK機能に制限が発生します。たとえば、ISMのビルドは可能ですが、CRYPTO_PATH環境変数を設定しないとアップロードはできません。この環境変数の詳細については、CRYPTO_PATH (108ページ) を参照してください。

IDKとISMToolを非管理対象サーバーにインストールするには、次の手順を実行します。

1 IDKを実行する管理対象サーバーを選択します。

2 IDKのインストール先となるホストと、ISMアプリケーションのインストール先となる管理対象サーバーで、同じバージョンのオペレーティングシステムが稼働していることを確認します。

たとえば、Redhat Linux 7.3 管理対象サーバー上にインストールするISMアプリケーションを作成する場合、Redhat Linux 7.3システムにIDKをインストールします。

3 Windows管理対象サーバーにIDKをインストールする場合には、CRYPTO_PATH (108ページ) の手順に従ってCRYPTO_PATH環境変数を設定してください。

4 Redhat Linux Application Server、Enterprise Server、Workstationのいずれかに IDKをインストールする場合、rpm-buildパッケージがインストールされていることを確認してください。このパッケージの有無を確認するには、次のコマンドを入力します。

```
rpm -qa | grep rpm-build
```

5 SolarisゾーンにIDKをインストールする場合、/usr/localディレクトリが存在し、書き込みアクセスが割り当てられていることを確認します(このディレクトリは、スパースルートゾーンには存在しないことがあります)。この確認作業は、Server Automationまたは次のzonecfgコマンドで実行できます。pathにはグローバルゾーン上のファイルシステムを指定します。

```
zonecfg:ゾーン名:fs> add fs
zonecfg:ゾーン名:fs> set dir=/usr/local
zonecfg:ゾーン名:fs> set special=path
zonecfg:ゾーン名:fs> set type=lofs
```

6 SAクライアントでは、名前に“ismtool”が含まれているソフトウェアポリシーを検索します。

7 ターゲットサーバープラットフォームに一致するポリシーを検索し、開きます。

8 [ポリシーアイテム] ビューを選択し、ポリシーに含まれるパッケージをすべて表示します。

9 ターゲットサーバーのOSとアーキテクチャーに一致するパッケージを検索し、開きます。

10 [**アクション**] > [**ソフトウェアのエクスポート**] メニューから、ターゲットサーバーにパッケージをダウンロードします。

11 ターゲットサーバーにパッケージを手作業でインストールします。

12 Unixの場合: ターミナルウィンドウで、IDKのインストール先となるホストにrootでログインし、PATH環境変数を次の値に設定します。

```
/usr/local/ismtool/bin
```

(Windowsの場合、PATHは自動設定されますが、設定を有効にするには再度ログインが必要です)。

13 ターミナルウィンドウで次のコマンドを入力し、IDKのインストールをチェックします。

```
ismtool --myversion
```

IDKクイックスタート

ここでは、簡単なISMを作成、ビルド、アップロードする方法について説明します。アップロードが完了すると、SAクライアントを実行し、アップロードしたISMを含むソフトウェアポリシーをチェックできます。

IDKをインストールしたホストのターミナルウィンドウで、次の手順を実行します。コマンドは、特に記載がない場合には、UnixとWindowsの両方で実行できます。

1 Unixの場合: IDKをインストールしたサーバーにrootでログインします。

rootでログインできない場合には、別のUnixユーザーでログインし、CRYPTO_PATH (108ページ) の手順に従ってCRYPTO_PATH環境変数を設定します。

2 Windowsの場合: ターミナルウィンドウを開き、CRYPTO_PATH環境変数が設定されていることを確認します。

3 ismusertool コマンドを実行して、ISM のアップロードを許可するアクセス権をユーザーに割り当てます。次に例を示します。

```
ismusertool --addUser johndoe
```

このコマンドを入力すると、次に示すように、エージェントゲートウェイ経由でコアと通信することを確認するメッセージが表示されます。

```
Using an agent gateway to reach an Opsware Core.
Is this correct?[y/n]: y
```

次に、Opswareのadminユーザー名とパスワードの入力プロンプトが開きます。

```
Enter Opsware Admin Username: admin
Enter admin's Opsware Password:
```

詳細については、ISMUserTool (110ページ) を参照してください。

4 ISMを新規作成します。

たとえば、fooという名前のISMを作成するには、次のコマンドを入力します。

```
ismtool --new foo
```

このコマンドを実行すると、現在のディレクトリ階層にfooという名前のディレクトリが作成されます。ISMは、fooディレクトリのコンテンツで構成されています。これ以降に実行するismtoolコマンドでは、ISMにfooを指定します。

5 アプリケーションファイルをISMに追加します。

アプリケーションファイルを追加する方法として、アーカイブをbarサブディレクトリにコピーする方法があります。たとえば、mytest.zipという名前のファイルにアプリケーションビットがある場合、次のコマンドを実行してISMに追加することができます。

Unixの場合:

```
cp /tmp/mytest.zip foo/bar
```

Windowsの場合:

```
copy c:\temp\mytest.zip foo\bar
```

6 後でISMをアップロードする際に、アップロード先となるソフトウェアポリシーへのパスを設定します。

注: ソフトウェアポリシーを格納するフォルダーに対して、フォルダー内のオブジェクトの書き込み権限が必要です。フォルダーのアクセス権は、SAクライアントの [フォルダーのプロパティ] ウィンドウで設定できます。

次のismtoolコマンドを実行すると、Quote Policyという名前のソフトウェアポリシーへのパスが設定されます。

Unixの場合:

```
ismtool  --opswpath '/My Kit/Service/Quote Policy' foo
```

Windowsの場合:

```
ismtool  --opswpath "/My Kit/Service/Quote Policy" foo
```

Unixではパスを一重引用符で囲みますが、Windowsでは二重引用符で囲みます。UnixとWindowsのいずれの場合も、パスにはスラッシュを使用します。

7 次のコマンドを入力して、ISM内でパッケージをビルドします。

```
ismtool --build foo
```

このコマンドを実行すると、foo/pkgサブディレクトリにパッケージが3つ作成されます。Linuxシステムでは、次のパッケージが作成されます。

```
foo-1.0.0-1.i386.rpm
foo-ism-1.0.0-1.i386.rpm
ismruntime-rpm-3.0.0-1.i386.rpm
```

foo-1.0.0-1.i386.rpmパッケージにはアプリケーションビットが含まれます。この例では、手順5でfoo/barサブディレクトリにコピーされています。foo-ism-1.0.0-1.i386.rpmパッケージには、インストールフックとコントロールスクリプトが含まれます(ここでは簡単な例を使用しているので、コントロールスクリプトはありません)。ismruntime-rpm-3.0.0-1.i386.rpmパッケージには、管理対象サーバーにパッケージをインストールする際にサーバーエージェントが使用するSA共有ランタイムが含まれます。

パッケージタイプ (RPM) は、Linuxシステムのネイティブのパッケージエンジンに対応しています。Windowsでは、--buildコマンドを実行するとfoo\pkgサブディレクトリに次のMSIパッケージが作成されます。

```
foo-1.0.0-1.msi
foo-ism-1.0.0-1.msi
ismruntime-msi-3.0.0-1.msi
```

8 次のコマンドを入力し、ISMをソフトウェアポリシーにアップロードします。

```
ismtool --upload foo
```

このコマンドを実行すると、いくつかプロンプトが開きます。最初のプロンプトでは、ISMのアップロード先となるコアを確認します。

```
Using the following Opsware Core:

  Data Access Engine : d02      192.168.198.91:1004
  Software Repository: d02      192.168.198.91:1003
  Command Engine     : d02      192.168.198.91:1018

  Is this correct? [y/n]: y
```

次のプロンプトでは、Opswareのユーザー名とパスワードを入力します。

```
  Enter Opsware Username: johndoe
  Enter johndoe's Opsware Password:
.. .
Success!
```

9 SAクライアントでソフトウェアポリシーを開き、上記の手順でアップロードしたISMが保存されていることを確認します。

プラットフォーム間の相違点

一般的に、IDKは異なるプラットフォーム (オペレーティングシステム) のパッケージで同じ機能を実行できます。ただし、プラットフォーム間でいくつか相違点があるので、ここではその内容について説明します。

Solaris

Solarisパッケージの名前は、最大9文字です。大文字の後に、アプリケーションを識別する小文字を続ける表記法を使用します。また、名前の最後に特殊な意味を持つ文字をオプションで付加することができます。この形式は表記法であり、必須条件ではありません。次に、Solarisパッケージ名の例をいくつか示します。

```
SPROcc
SPROcmpl
SPROcodmg
SUNWgssx
SUNWgzip
SUNWhea
SUNWhiu8x
SUNWhmd
SUNWhmdu
SUNWhmdx
```

ISMToolでSolarisパッケージを作成する場合、最大9文字でパッケージ名を使用する必要があります。ISMToolで生成されるパッケージ名は、`ISM`で始まり、その後にISM名の最初の5文字が続きます。さらに、`c` (コントロールパッケージの場合)、`0` (アプリケーションパッケージの最初の部分)、`1` (アプリケーションパッケージの2番目の部分) などが続きます。たとえば、ISM名が`foobar`の場合、パッケージ名は次のようになります。

```
ISMfooba0
ISMfoobac
```

切り詰めが発生した場合は、ISMToolで警告メッセージが表示されるので、競合しないようにISMの名前を変更してください。パッケージ名を表示するには、Solarisの`pkginfo`コマンドを実行します。

Solarisパススルーパッケージのアップロードでは、応答ファイルはアップロードされません。応答ファイルは、手動でアップロードする必要があります。

Windows

Windowsでは、ISMToolでアプリケーションとコントロールWindowsインストーラー (MSI) パッケージを作成する際、`ProductName`と`ProductVersion`は次のようにエンコードされます。

```
ProductName:     <名前>-<バージョン>
ProductVersion:  0.0.<app|ctl>リリース>
```

<名前>、<バージョン>、<リリース>は、ISMの内部情報を示し、ISMToolの`--info`コマンドで参照できます。このエンコーディングは設計上のスキーマであり、修復プロセスを実行する上で必要になります。

第5章 IDKビルド環境

ISMファイルシステムの構造

ISMToolの--buildコマンドと--uploadコマンドは、ISMディレクトリ上で動作します。このディレクトリは、--unpackコマンドまたは--newコマンドで作成されます。--unpackコマンドは、--packで圧縮したファイル(ISMディレクトリコンテンツが保存されています)を解凍します。--newコマンドは、新しいISMディレクトリを作成します。たとえば、次のコマンドを実行すると、ntp-4.1.2という名前のディレクトリが新規作成されます。

```
ismtool --new ntp-4.1.2
```

このコマンドにより、ntp-4.1.2ディレクトリの下に次のサブディレクトリが作成されます。

- bar: バイナリアーカイブが格納されています。バイナリアーカイブとは、アプリケーションパッケージの作成に使用するコンテンツです。
- doc: ISMのビルドで自動生成されるドキュメント(HTML)が格納されます。このディレクトリには、他のドキュメントも作成できます。
- ism: ISMのコントロールパッケージの作成に必要なすべてのファイルが格納されています。ismディレクトリでは、デフォルトのパッケージフック(インストール前、インストール後、アンインストール前、アンインストール後)の編集と、コントロールスクリプトをism/controlに追加する作業を実行できます。
- log: ソース変換(コンパイルやローカルインストール)の出力、ネイティブパッケージエンジン(msi、rpm、pkgtrans、swpackage)からの出力、SAアップロードなどの追跡に使用するファイルが格納されています。
- pad: ISMToolの--addPkgPropオプションで指定するインストールスクリプト(インストール前、インストール後、アンインストール前、アンインストール後)が格納されています。
- pkg: アプリケーション、コントロール、共有ランタイムパッケージなど、-buildによって生成される項目が格納されています。また、このサブディレクトリにはパススルーパッケージのコピーも格納されています。
- tmp: ISMToolのスクラッチ領域として使用されます。
- src: ソースをバイナリアーカイブにコンパイルする処理を制御するファイルが格納されています(オプション)。

次のコマンドを実行すると、次のようなISMサブディレクトリが作成されます。

```
ismtool --build ntp-4.1.2
```

ソースビルドによってバイナリアーカイブディレクトリが出力され、__ntp-4.1.2_src_ntp.spec.cpioという名前が割り当てられています。ビルドによって、log、pkg、tmpの各サブディレクトリと、ファイル名が2つの下線で始まるファイルが作成されます。

```
ntp-4.1.2/
            src/
                ntp-4.1.2.tar.gz
                ntp.spec
            bar/
                __ntp-4.1.2_src_ntp.spec.cpio
                __ntp-4.1.2_src_ntp.spec.cpio.meta
            pkg/
               ntp-4.1.2-3.i386.rpm
               ntp-ism-4.1.2-7.i386.rpm
```

```
   ismruntime-rpm-2.0.rpm
log/
    . . .
doc/
    index.html
    index/
        ntp-4.1.2-3.i386.rpm.html
        ntp-ism-4.1.2-7.i386.rpm.html
tmp/
   . . .
ism/
    ism.conf
    bin/
        ismget
        parameters
        platform
        python
    env/
        ism.sh
        ism.py
        ism.pl
    pkg/
        ism_check_install
        ism_post_install
        ism_post_uninstall
        ism_pre_install
        ism_pre_uninstall
    control/
pad/
    ismruntime-rpm-2.0.0.i386.rpm
     . . .
    ntp-4.1.2-3.i386.rpm/
                          pkg.conf
                          scripts/
    ntp-ism-4.1.2-7.i386.rpm/
   . . .
```

ビルドプロセス

ここでは、次の内容について説明します。

- -buildコマンドを実行するタイミング
- 複数のコマンドラインオプション
- -buildコマンドで実行されるアクション
- -buildコマンドで作成されるパッケージ

--buildコマンドを実行するタイミング

ISMToolの--buildコマンドは、--newの後、--uploadの前に実行します。オプション指定でISMを変更した場合には、変更内容を有効にするために--uploadの前に--buildを実行する必要があります。たとえば、--opswpathを指定する場合、--buildを実行して新しいソフトウェアポリシーパスを適用してから、ISMをアップロードしてください。

複数のコマンドラインオプション

ISMToolオプションは、コマンドラインで複数指定する方法と1つずつ指定する方法があります。次のUnixの例では、apacheという名前のISMディレクトリについて、ネイティブパッケージエンジンをrpm3、バージョンを2.0.47b、デフォルトのインストールユーザーをroot、デフォルトのインストールグループをrootに変更しています。

```
ismtool --pkgengine rpm3 --version 2.0.47b --user root --group root apache
```

上記のコマンドは、以下のコマンドをすべて実行した場合に相当します。

```
ismtool --pkgengine rpm3     apache
ismtool --version   2.0.47b  apache
ismtool --user      root     apache
ismtool --group     root     apache
```

入力されたコマンドは、論理的に適切な実行順序になるようにISMToolによって自動的にソートされます。次のコマンドを実行すると、ビルドの前にapacheのバージョンが3.0に変更されます。

```
ismtool --build --version 3.0 apache
```

--buildコマンドで実行されるアクション

ISMToolの--buildコマンドは、次の手順で実行されます。

1. ビルド前のクリーンアップとして、ビルドによって副次的に生成されたものをすべて削除します。ただし、これまで実行したビルドでビルドキャッシュとして生成されたcpioアーカイブは削除されません。
2. オプションスクリプトであるism/build/ism_cleanを実行します。ism/buildサブディレクトリ内のスクリプトは、ビルドプロセスにフックされます。このスクリプトを使用するには、手動で作成する必要があります。
3. アプリケーションソースでチェックサムを実行します。現在のチェックサムと以前のチェックサムが一致しない場合、アプリケーションリリース番号をインクリメントします。
4. コントロールソース(ismサブディレクトリのコンテンツ)でチェックサムを実行します。現在のチェックサムと以前のチェックサムが一致しない場合、アプリケーションリリース番号をインクリメントします。
5. オプションスクリプトであるism/build/ism_preを実行します。
6. ソースビルドの場合、.specファイルをsrcサブディレクトリ内で再帰的に検索し、コンパイルして実行します。
7. 共有ランタイムパッケージを作成します。
8. コントロールパッケージを作成します。
9. アプリケーションパッケージを作成します。
10. 自動のHTMLドキュメントdoc/index/index.htmlを生成します。
11. オプションスクリプトであるism/buid/ism-postを実行します。

--buildコマンドで作成されるパッケージ

--buildコマンドを実行すると、pkgサブディレクトリに次のパッケージが作成されます。

- アプリケーションパッケージ: bar (バイナリアーカイブ) サブディレクトリのコンテンツから作成されます。このパッケージには、アプリケーションビットが含まれています。アプリケーションアーカイブをbarサブディレクトリにコピーしてから、--buildコマンドを実行します。アプリケーションパッケージのファイル名は、次の形式で指定されます。<バージョン>はISM, 全体に適用されるバージョン、<リリース>はアプリケーションパッケージごとのリリースです。詳細については、ISMの名前、バージョン番号、リリース番号 (74ページ) を参照してください。

  <名前>-<バージョン>-<リリース>.<パッケージの拡張子>

- コントロールパッケージ: このパッケージには、ismサブディレクトリのコントロールスクリプトとインストールスクリプトが格納されています。コントロールパッケージのファイル名は、次の形式で指定されます。

  <名前>-ism-<バージョン>-<リリース>.<パッケージの拡張子>

- 共有ランタイムパッケージ: このパッケージには、サーバーエージェント (インストール時) とコントロールスクリプトが実行する共有実行時ルーチンが格納されています。この実行時ルーチンは、アプリケーション自体ではなくServer Automationが使用します。共有ランタイムパッケージのファイル名は、次の形式で指定されます (<CTLプレフィックス>が付加されるのは、--ctlprefixオプションでデフォルト以外の値を指定した場合のみです)。

  ismruntime-<CTLプレフィックス>-<パッケージタイプ>-<IDKバージョン>.<パッケージの拡張子>

- パススルーパッケージ: --addPassthruPkgオプションを指定すると、pkgサブディレクトリにそのままコピーされます。

ISMのアプリケーションファイルの指定

ここでは、アプリケーションファイルをISMに提供する方法について説明します。

- アーカイブをbarサブディレクトリに配置
- パススルーパッケージの指定
- ソースのコンパイル (Unixのみ)

アーカイブをbarサブディレクトリに配置

--buildの実行前に、ファイルアーカイブをISMのbar (バイナリアーカイブ) サブディレクトリに手動でコピーすることができます。または、barサブディレクトリ内のアーカイブは、specファイルの%filesセクションのディレクティブによって、cpioファイルとして生成されます。詳細については、ソースのコンパイル (Unixのみ) (71ページ) も参照してください。

--buildコマンドを実行すると、barサブディレクトリ内にあるアーカイブが再パッケージされ、pkgサブディレクトリのアプリケーションパッケージが作成されます。barサブディレクトリには、次のようなタイプのアーカイブが格納されています。

表46　バイナリアーカイブの有効なタイプ

ファイル拡張子	アーカイブタイプ
.cpio	Unix CPIOアーカイブ
.msi	Microsoftインストーラー
.rpm	RPM Package Manager

表46　バイナリアーカイブの有効なタイプ

ファイル拡張子	アーカイブタイプ
.tar	テープアーカイブ
.tar.bz2	bzip2圧縮テープアーカイブ
.tar.gz	gzip圧縮テープアーカイブ
.zip	Info-Zip互換Zip

パススルーパッケージの指定

barサブディレクトリ内のアーカイブとは異なり、パススルーパッケージの抽出や再パッケージは行われません。--addPassthruPkgコマンドを実行すると、パススルーパッケージがそのままpkgサブディレクトリにコピーされます。--addPassthruPkgで指定したパッケージは、ISMディレクトリに保存できません。次の例では、パススルーパッケージをISMに追加し、ソフトウェアポリシーへの追加を指定しています。

```
ismtool --addPassthruPkg /tmp/bos.rte.libs.5.1.0.50.U --pkgType lpp ISMNAME
ismtool --attachPkg bos.rte.libs-5.1.0.50 --attachValue true ISMNAME
```

ソースのコンパイル (Unixのみ)

--buildコマンドは、srcサブディレクトリを再帰的に検索し、specファイル (ファイル名の末尾がspec) を検出します。検出されたspecファイルはBourne Shellにコンパイルされ、実行されます。specファイルは、RPM specファイル言語から派生した簡易版の言語で記述されています。ISMToolのspecファイルライクな言語コンパイラーにより、既存のRPM specファイルに最小限の変更を加えるだけで使用できます。

specファイル言語の詳細については、次のURLにあるMaximum RPMのドキュメントを参照してください。

http://www.rpm.org/max-rpm/index.html

specファイルの例

NTP 4.1.2のISM specファイルの例を示します。

```
################################################
# Common Preamble
################################################

%define ismname %(../ism/bin/ismget name)
%define version %(../ism/bin/ismget version)
%define prefix %(../ism/bin/ismget prefix)

Name:%{ismname}
Version:%{version}

################################################
# prep, build, install, files
################################################

Source: http://www.eecis.udel.edu/~ntp/ntp_spool/ntp4/ntp-4.1.2.tar.gz

%prep
```

```
%setup -n ntp-4.1.2

%build

%ifos Solaris2.7
echo ''do something Solaris2.7 specific''
%endif

%ifos Linux
echo ''do something Linux specific''
%endif

./configure --prefix=%prefix
make

%install
/bin/rm -rf $ISM_BUILD_ROOT
make install prefix=$ISM_BUILD_ROOT/%{prefix}

%files
%defattr(-,root,root)
%prefix
```

specファイルのプリアンブル

プリアンブルでは、ismgetプログラムでISMから取得する情報を指定します。次の3行は、ISMの名前、バージョン、プレフィックスを取得します。

```
%define ismname    %(../ism/bin/ismget name)
%define version    %(../ism/bin/ismget version)
%define prefix     %(../ism/bin/ismget prefix)
```

取得した情報は、ソースの設定とコンパイルで使用できます。ただし、%defineコマンドはオプションであり、プリアンブルでの必須タグはNameとVersionのみです。

%prep

%prepセクションは、コンパイル用にソースの準備を行うセクションであり、ソースの展開と解凍を行います。Sourceタグで、ソースファイルを1つ指定します。ソースを複数指定する場合は、Source0、Source1、のようにタグを並べて指定します。...同様に、パッチはPatchタグを1つ、またはPatch0、Patch1、のように並べて指定します。...ISMToolは、マクロ機能を複製します(『Maximum RPM』を参照してください)。%setupマクロは、ソースをアンパックする方法を制御します。また、%prepセクションでは%patchマクロを使用したパッチ管理も可能です。

%build

%buildセクションのシェルスクリプトコマンドは、ソースをバイナリに変換します。ソースのコンパイルには、通常は./configure -prefix=%{prefix}とmakeを使用します。構成は、プラットフォーム(オペレーティングシステム)に応じて切り替えることができます。プラットフォームタグは、実際のインストール環境にあるRPMで下位互換性を維持するために使用します。次に、プラットフォームの切り替えに使用できるISMToolのプラットフォーム文字列の例を示します。

```
Linux
RedHat
RedHat-Linux-7.2
```

```
RedHat-Linux-AS2.1
Solaris
Solaris2.8
Solaris-2.8
SunOS
SunOS5.7
SunOS-5.7
hpux
hpux11.00
hpux-11.00
HPUX
HPUX11.00
HPUX-11.00
aix
aix4.3
aix-4.3
AIX
AIX4.3
AIX-4.3
```

%install

%installセクションでは、ビルドから仮想インストールディレクトリにファイルをコピーする指定を行います。たとえば、%prefixを/usr/localに設定すると、次のコマンド行を実行することでNTPが/usr/local/binにインストールされます。

```
make install prefix=$ISM_BUILD_ROOT/%{prefix}
```

$ISM_BUILD_ROOT変数 ($RPM_BUILD_ROOTに相当) は、ISMのtmpディレクトリ内にある一時ディレクトリを示します。この一時ディレクトリは仮想インストールのルートとして使用され、%filesセクションのディレクティブが適用されます。

また、%installセクションでは、バイナリインストールのファイルを抽出する場所も指定します。バイナリインストールから開発用サーバー上に抽出されたファイルは、パッケージして仮想インストールのディレクトリに保存することが可能です。ただし、この処理を実行できない場合、バイナリインストーラーをエンドシステムに転送し、インストール後のISMフックを使用してインストールすることが可能です。この場合、インストーラーのバイナリアーカイブを作成し、ISMのbarサブディレクトリにコピーします。

%files

ソース変換フェーズで出力されるファイル群は、specファイルの%filesセクションにあるディレクティブで指定されます。このファイルは、ISMのbarサブディレクトリにあるcpioにアーカイブされます。

ソース変換の最後のフェーズでは、$ISM_BUILD_ROOTにインストールするファイルを選択します。%filesセクションのディレクティブは、『Maximum RPM』に記載されている選択機能のサブセットです。このディレクティブでは、ファイルまたはディレクトリのリスト (サブディレクトリも含む) を$ISM_BUILD_ROOTを基準に指定します。上記の例のインストールパスは$ISM_BUILD_ROOT/%{prefix}です。このファイルをパッケージ対象として選択する場合は、パッケージディレクトリに%prefixを指定します。

ファイルやディレクトリの名前を指定して選択する方法の他に、メタ情報を指定することもできます。%defattr(-,root,root) を指定すると、アーカイブエンジンは、ファイルシステム内で検出されたモードを使用しますが、アーカイブ作成時にはファイルシステム内で検出したファイルの所有権をroot,rootで置換します。%defattr() と%attr() の詳しい説明は、『Maximum RPM』を参照してください。

ISMの名前、バージョン番号、リリース番号

ここでは、次の内容について説明します。

- ISMの名前、バージョン、リリースの初期値
- ISMのバージョンとリリース番号の比較
- ISMバージョンのアップグレード

ISMの名前、バージョン、リリースの初期値

--newコマンドは、新しいISM用のディレクトリを作成し、ISM.の内部ベース名を指定します。たとえば、次のコマンドを実行すると、ファイルシステム内にmystuffディレクトリが作成され、内部ベース名がmystuff、バージョン番号が1.0.0に設定されます。

```
ismtool --new mystuff
```

ほとんどの場合、バージョン番号は--newで指定します。次のコマンドを実行すると、ntp-1.4.2という名前のディレクトリが作成され、内部ベース名がntp、バージョン番号が1.4.2に設定されます。

```
ismtool --new ntp-1.4.2
```

内部ベース名、バージョン番号、リリース番号の表示には、--infoコマンドを使用します。

```
ismtool --info ntp-1.4.2
```

このコマンドを実行すると、次の内容が出力されます。

```
. . .
name:                ntp
version:             4.2.1
appRelease:          0
. . .
ctlRelease:          0
. . .
```

ISMのバージョンとリリース番号の比較

ISMのバージョン番号とリリース番号には、いくつか相違点があります。バージョン番号の指定には、--newコマンドまたは--versionコマンドを使用します。リリース番号はISMToolが自動生成するので、ユーザー指定はできません。バージョン番号は、ISM全体に適用される番号です。アプリケーションパッケージとコントロールパッケージには、それぞれ個別のリリース番号が割り当てられます。--buildコマンドを実行すると、パッケージが再生成されるたびにリリース番号はインクリメントされます。アプリケーションパッケージとコントロールパッケージは別々にビルドできるので、パッケージのリリース番号は異なる場合があります。

アプリケーションパッケージとコントロールパッケージの名前は、内部ベース名、バージョン番号、リリース番号で構成されます。たとえば、アプリケーションパッケージの名前がntp-4.1.2-3.i386.rpmの場合、4.1.2はバージョン番号、3はリリース番号を示しています。詳細については、-buildコマンドで作成されるパッケージ (70ページ)も参照してください。

IDK (ISMではありません) のバージョン番号を表示するには、次のコマンドを実行します。

```
ismtool --myversion
```

ISMバージョンのアップグレード

内部ベース名の変更 (`--name`) とバージョン番号の変更 (`--version`) は、ユーザーが実行することは可能ですが、ディレクトリ名は自動的に変更されないのでお勧めしません。内部ベース名やバージョン番号を変更する場合は、混乱を避けるため、ISM内のディレクトリ名も変更してください。

内部ベース名、バージョン番号、ディレクトリ名、ソフトウェアポリシーパスは、同じ設定にしておくことをお勧めします。たとえば、`foo-1.2.7`を`foo-1.2.8`にアップグレードする手順を示します。

1. `foo-1.2.7`と同じレベルに、ISMディレクトリを新規作成します。

   ```
   ismtool --new foo-1.2.8
   ```

2. アーカイブを`foo-1.2.8/bar`ディレクトリにコピーするか、パススルーパッケージを指定します。
3. 旧バージョンと同じレベルに、ソフトウェアポリシーのパスを設定します。

   Unixの場合:

   ```
   ismtool --opswpath 'MyFolder/{$NAME}/{$VERSION}'
   ```

   Windowsの場合:

   ```
   ismtool --opswpath "MyFolder/{$NAME}/{$VERSION}"
   ```

   `--opswpath`コマンドを実行すると、`NAME`変数に`foo`、`VERSION`変数に1.2.8が設定されます。現在の変数値を表示するには、`--info`コマンドを実行します。変数値の設定の詳細については、ISMTool変数 (100ページ) を参照してください。
4. ISMToolを使用して、`foo-1.2.8` ISMをビルドおよびアップロードします。
5. SAクライアントで、管理対象サーバーから`foo-1.2.7`ポリシーをデタッチします。
6. (オプション) `foo-1.2.7`ポリシーを削除します。
7. 管理対象サーバーに新しいソフトウェアポリシーを適用します。

第6章 IDKスクリプト

ISMスクリプトの概要

ISMスクリプトとは、Unixシェルスクリプトまたは Windowsコマンドラインスクリプトであり、ISMディレクトリに格納されています。ここでは、次のタイプのISMスクリプトについて説明します。

- **インストールフック**: ISMTool の `--build` コマンドによって ISM のコンロールパッケージにバンドルされます。管理対象サーバー上で、ネイティブのパッケージエンジン (`rpm` など) によって実行されます。インストールフックは、コントロールスクリプトを呼び出すことができます。
- **コントロールフック**: ISMのコントロールパッケージにバンドルされます。ソフトウェアサーバーの起動など、日常的に行うアプリケーションタスクを実行します。
- **インストールスクリプト**: コントロールパッケージではなく、ソフトウェアリポジトリに格納されます。このスクリプトは、SAクライアントのパッケージのプロパティで表示できます。

インストールフックとコントロールスクリプトの開発と実行は、全体的に次のようなプロセスで行われます。

1. ISMToolの`--new`コマンドを実行します。これにより、デフォルトのインストールフックが作成されます。
2. テキストエディターで、コントロールスクリプトを作成します。
3. テキストエディターで、デフォルトのインストールフックを変更します。インストールフックではコントロールスクリプトを呼び出すことも可能です。
4. ISMToolを使用してISMをビルドし、アップロードします。
5. SAクライアントで、ISMに含まれているアプリケーションを管理対象サーバーにインストールします。インストールでは、インストール前のフックとインストール後のフックが管理対象サーバー上で実行されます。
6. アプリケーションの運用ライフサイクルでは、コントロールスクリプトの実行またはスケジュール設定を行います。
7. アプリケーションのライフサイクルが終了すると、アプリケーションはアンインストールされます。アンインストールでは、アンインストール前のフックとアンインストール後のフックが管理対象サーバー上で実行されます。

インストールスクリプトは、インストールフックやコントロールスクリプトとは異なるプロセスで実行されます。詳細については、インストールスクリプト (87ページ) を参照してください。

ISMスクリプトでは、スクリプトに関連付けられたパッケージをロックするプログラム (`rpm`や`pkgadd`など) の呼び出しはできません。

インストールフック

インストールフックは、`ism/pkg`サブディレクトリに格納されているスクリプトです(インストールフックは「パッケージングスクリプト」と呼ばれることもあります)。インストールフックは、管理対象サーバーでアプリケーションをインストールまたはアンインストールするプロセスの中で実行されます。

インストールフックの作成

ISMToolの--newコマンドを実行すると、次のインストールフックが作成されます。

Unixの場合:

```
ism/pkg/
       ism_check_install
       ism_post_install
       ism_post_uninstall
       ism_pre_install
       ism_pre_uninstall
```

Windowsの場合:

```
ism\pkg\
       ism_post_install.cmd
       ism_post_uninstall.cmd
       ism_pre_install.cmd
       ism_pre_uninstall.cmd
```

インストールフックをカスタマイズするには、テキストエディターで変更します。インストールフックの内容は編集可能ですが、ファイル名の変更はできません。

デフォルトのism_pre_installフックとism_post_uninstallフックは単なるスタブであり、アクションは実行しません。デフォルトのism_post_installフックは、ism_configureとism_startの各コントロールスクリプトを呼び出します。また、デフォルトのism_pre_uninstallフックは、ism_stopコントロールスクリプトを呼び出します。コントロールスクリプトはISMToolによって自動生成されないので、テキストエディターで作成する必要があります(コントロールスクリプト (82ページ)を参照してください)。

--buildコマンドで自動生成されるデフォルトのインストールフックの内容については、次の項で説明します。

- Unix向けのデフォルトのインストールフック (80ページ)
- Windows向けのデフォルトのインストールフック (81ページ)

インストールフックの確認

ネイティブパッケージエンジンは、ism_check_installフックを直接サポートするものもありますが、ism_pre_installフックを使用して暗黙的にサポートするものもあります。ISMToolは、ネイティブパッケージエンジンにcheck_installの機能をマッピングします。check_installスクリプトが何らかのコードを返した場合、インストールは中断します。

インストールフックの呼び出し

ISMのアプリケーションを管理対象サーバーにインストール(またはアンインストール)すると、サーバー上のネイティブパッケージエンジンはインストールフックを呼び出します(ユーザーがインストールフックを直接実行することはありません)。たとえばLinuxシステムの場合、rpmユーティリティはism_pre_installを呼び出し、その後すぐにアプリケーションビットをインストールします。また、アプリケーションビットをアンインストールした後、すぐにism_post_uninstallを呼び出します。

詳細については、インストールスクリプトとインストールフックの呼び出し (88ページ)も参照してください。

インストールフックとZIPパッケージ

Server Automationで使用するZIPパッケージエンジンは、他のパッケージエンジンと異なり、インストールフックをサポートしません。ZIPパッケージエンジンが指定され、インストールフックファイルが空でない場合、ISMToolは警告を表示し、インストールフックファイルを無視します。

ZIPパッケージとインストールディレクトリ

IDKによって生成されるZIPパッケージは、再配置できません。つまり、1つのZIPパッケージを使用して、1つの管理対象サーバー上にある複数のディレクトリに複数のアプリケーションインスタンスをインストールすることはできません。したがって、SAクライアントでエンドユーザーがZIPパッケージの[インストールパス]フィールドを変更すると、パッケージのインストールは失敗します。インストールディレクトリを変更するには、ISM開発者が`--prefix`オプションまたは`--ctlprefix`オプションで新しいパスを指定し、ISMを新しくビルドしてから、これをコアにアップロードする必要があります(Windows NT4では上記のオプションの指定が必要であり、変数は指定できません)。

ZIPパッケージでのベストプラクティスとして、ISM開発者は、ISMの説明に次のような内容の警告を指定することをお勧めします。“WARNING: このパッケージのインストールパスは変更しないでください。”

インストールフック関数

インストールフックは、カスタマイズによって、次の表で示すアクションを実行できます。

表47　　インストールフック関数

インストールフック	関数
`ism_pre_install`	必要なディレクトリの作成、ユーザーの作成、ディレクトリアクセス権の設定
`ism_post_install`	`ism_configure`コントロールスクリプトの呼び出し、`ism_start`コントロールスクリプトの呼び出し(Webサーバーの起動など)
`ism_pre_uninstall`	`ism_stop`コントロールスクリプトの呼び出し(サーバーの停止)
`ism_post_uninstall`	必要なクリーンアップを実行

コントロール専用ISMのスクリプト

`--skipApplicationPkg`オプションを指定すると、ISMToolはアプリケーションパッケージをビルドしないので、コントロール専用ISMの作成が可能になります。この機能を使用すると、ISMToolでインストールまたはパッケージを行わないアプリケーション向けにコントローラーをビルドできます。例として、オペレーティングシステムのコア機能、実行中のためパッケージングできないアプリケーション、専用ハードウェアなどのコントローラーがあります。

コントロール専用ISMのインストールとアンインストールには、`ism_ctl_post_install`と`ism_ctl_pre_uninstall`の各スクリプトが実行されます(このスクリプトはすべてのISMで実行されますが、通常はコントロール専用ISMのみで使用するように指定します)。このスクリプトはISMToolによって生成されないので、ユーザーが作成してから`--build`コマンドを実行する必要があります。次に、スクリプトで必要な名前と場所を示します。

Unixの場合:

```
ism/pkg/
       . . .
       ism_ctl_post_install
       ism_ctl_pre_uninstall
```

Windowsの場合:

```
ism\pkg\
       . . .
       ism_ctl_post_install.cmd
       ism_ctl_pre_uninstall.cmd
```

管理対象サーバーでのインストールフックの場所

開発システムで--buildコマンドを実行し、ISMのコントロールパッケージにインストールフックをバンドルします。管理対象サーバーでは、コントロールパッケージのコンテンツはISMのctlprefixで指定したディレクトリにインストールされます。インストールフックは、デフォルトで次の場所にインストールされます。

Unixの場合:

/var/opt/OPSWism/<ISM名>/pkg

Windowsの場合:

%ProgramFiles%\OPSWism\ISM名\pkg

インストールフックのデフォルトディレクトリを変更するには、ISMのビルドとアップロードの前に--ctlprefixオプションを指定します。たとえば次のようにctlprefixを指定すると、インストールフックは/usr/local/ntp-4.1.2/pkgにインストールされます。

```
ismtool --ctlprefix /usr/local ntp-4.1.2
```

Unix向けのデフォルトのインストールフック

デフォルトのism_pre_installフック:

```
#!/bin/sh
#
# ISM Pre Install Script
#
. `dirname $0`/../env/ism.sh
```

デフォルトのism_post_installフック:

```
#!/bin/sh
#
# ISM Post Install Script
#
. `dirname $0`/../env/ism.sh
if [ -x ${ISMDIR}/control/ism_configure ]; then
${ISMDIR}/control/ism_configure
fi
if [ -x ${ISMDIR}/control/ism_start ]; then
${ISMDIR}/control/ism_start
fi
```

デフォルトのism_pre_uninstallフック:

```
#!/bin/sh
#
# ISM Pre Uninstall Script
#
. `dirname $0`/../env/ism.sh
if [ -x ${ISMDIR}/control/ism_stop ]; then
${ISMDIR}/control/ism_stop
fi
```

デフォルトのism_post_unininstallフック:

```
#!/bin/sh
#
```

```
# ISM Post Uninstall Script
#
. `dirname $0`/../env/ism.sh
```

Windows向けのデフォルトのインストールフック

デフォルトのism_pre_install.cmdフック:

```
@echo off
REM
REM ISM Pre Install Hook
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
call %1
ENDLOCAL
```

デフォルトのism_post_install.cmdフック:

```
@echo off
REM
REM ISM Post Install Script
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
call %1
REM
REM Call the ISM's configure script
REM
IF EXIST "%ISMDIR%\control\ism_configure.cmd"
call "%ISMDIR%\control\ism_configure.cmd"
REM
REM Call the ISM's start script
REM


IF EXIST "%ISMDIR%\control\ism_start.cmd"
call "%ISMDIR%\control\ism_start.cmd"
ENDLOCAL
```

デフォルトのism_pre_uninstall.cmdフック:

```
@echo off
REM
REM ISM Pre Uninstall Hook
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
```

```
call %1
REM
REM Call the ISM's stop script
REM
IF EXIST "%ISMDIR%\control\ism_stop.cmd"
call "%ISMDIR%\control\ism_stop.cmd"
ENDLOCAL
```

デフォルトのism_post_unininstall.cmdフック:

```
@echo off
REM
REM ISM Post Uninstall Script
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
call %1
```

コントロールスクリプト

ISMコントロールスクリプトはism/controlディレクトリに格納されています。コントロールスクリプトは、インストールされたアプリケーションのハウスキーピングとメンテナンスを行います。

インストールフックは、コントロールスクリプトを実行することができます。インストール(またはアンインストール)中に実行するタスクを定期的に実行したい場合は、コントロールスクリプトとして記述する必要があります。たとえば、ism_post_installフックは、ism_startコントロールスクリプトを呼び出すことによって、インストールの直後にアプリケーションを起動することができます。また、ism_pre_uninstallフックはism_stopコントロールスクリプトを呼び出すことにより、アプリケーションをシャットダウンできます。

コントロールスクリプトの実行には、SA クライアントの ISM コントロールウィンドウを使用します。また上級ユーザーは、Global Shellのコマンドラインからコントロールスクリプトを実行することも可能です。

コントロールスクリプトの作成

コントロールスクリプトは、インストールフックとは異なり、ISMToolによって自動生成されないので、テキストエディターで作成する必要があります 作成したコントロールスクリプトは、ism/controlサブディレクトリに追加できます。コントロールスクリプトのファイル名には、次の命名規則が適用されます。

Unixの場合:

```
ism/control/
          ism_start
          ism_stop
          ism_configure
          ism_reconfigure
```

Windowsの場合:

```
ism\control\
          ism_start.cmd
          ism_stop.cmd
          ism_configure.cmd
```

ism_reconfigure.cmd

コントロールスクリプトの名前は、SAクライアントの[ISMコントロール]ウィンドウでは別の形式で表示されることがあります。[ISMコントロール]ウィンドウの[アクション]フィールドにはコントロールスクリプトの名前が表示されますが、先頭のism_やファイルタイプ拡張子が表示されない場合があります。たとえば、ism_start.cmdという名前のコントロールスクリプトは、[アクション]フィールドでは「start」と表示されることがあります。[アクション]フィールドには、コントロールスクリプト名の最初の25文字のみが表示されます。したがって、最初の25文字の部分が重複しないようにしてください。UnixとWindowsのいずれも、先頭のism_は小文字で指定します。大文字を使用すると、[アクション]フィールドにプレフィックスが表示されます。

Unixでは、ism/controlにあるコントロールスクリプトが実行可能であることを確認してください。これ以外のディレクトリにあるコントロールスクリプトはSAクライアントで表示されません。

コントロールスクリプトの関数

コントロールスクリプトは、アプリケーションの管理で反復的に行うタスクを実行します。次の表では、コントロールスクリプトの一般的な用途とまとめています。

表48 コントロールスクリプトの関数

コントロールスクリプト	関数
ism_start	コンパニオンサーバーまたは依存関係のあるサーバーの通知、アプリケーションの起動
ism_stop	コンパニオンサーバーまたは依存関係のあるサーバーの通知、アプリケーションの停止
ism_configure	構成の実行
ism_reconfigure	to ism_configure類似していますが、まずism_stopを呼び出してからism_startを呼び出す

管理対象サーバーでのコントロールスクリプトの場所

コントロールスクリプトは、インストールフックと同様に、--buildコマンドでコントロールパッケージにバンドルされます。管理対象サーバーでは、コントロールスクリプトはISMのctlprefix値で指定されたディレクトリに格納されます。デフォルトでは、コントロールスクリプトは管理対象サーバー上の次のディレクトリにインストールされます。

Unixの場合:

/var/opt/OPSWism/<ISM名>/control

Windowsの場合 (NT4を除く):

%ProgramFiles%\OPSWism\<ISM名>\control

デフォルトディレクトリを変更するには、ISMToolで--ctlprefixオプションを指定します。Windows NT4では、--ctlprefixを指定します。変数は指定できません。

ISMパラメーターを使った動的構成

ISMのparameterユーティリティによって、コントロールスクリプトとインストールフックはSAのカスタム属性の値にアクセス可能になります。ISMパラメーターのキーは、対応するカスタム属性の名前と同じです。カスタム属性の値に基づいて、パラメーターの値が決まります。カスタム属性の値は、SAオブジェクト (ファシリティ、カスタマー、サーバー、デバイスグループ) から取得されます。

カスタム属性は、SAクライアントで設定される名前と値のペアであり、構成情報が格納されています。たとえば、Apache Webサーバーのポート番号を指定する場合、APACHE_1.3_PORTという名前のカスタム属性に80という値が設定されているとします。ISMにAPACHE_1.3_PORTという名前のパラメーターがある場合、コントロールスクリプトでカスタム属性の現在の値を取得できます。

エンドユーザーはSAクライアントの[ISMコントロール]ウィンドウから、パラメーターのソース(SAオブジェクト)の表示、パラメーター値の表示、パラメーター値の上書きを実行できます。

ISMパラメーターの開発プロセス

ISMパラメーターの開発と使用は、次のようなプロセスで行います。

1. ISMToolを使用して、新しいパラメーターを追加します。
2. テキストエディターを使用して、パラメーターへのアクセスに使用するコントロールスクリプトを記述(またはインストールフックを変更)します。
3. ISMToolを使用してISMをビルドし、アップロードします。
4. SAクライアントで、ISMに含まれているアプリケーションを管理対象サーバーにインストールします。
5. SAクライアントで、パラメーターと同じ名前のカスタム属性を作成します。
6. SAクライアントを使用して、管理対象サーバーでコントロールスクリプトを実行します。実行時に、スクリプトはServer Automationからパラメーター(コントロール属性)の値を取得します。

ISMパラメーターの追加、表示、削除

新しいパラメーターを作成するには、ISMToolの--addParamコマンドを使用します。作成されたパラメーターは、ISMのスクリプトで取得可能です。パラメーターには4つのフィールドがあり、それぞれISMToolオプションで指定されます。次の表では、フィールドとそれに対応するオプションをまとめます。

表49　ISMパラメーターのフィールド

パラメーターのフィールド	ISMToolオプション	説明
名前	--addParam	ISMパラメーターの名前。カスタム属性の名前と同じ名前を指定します。
デフォルト値	--paramValue	パラメーターのデフォルト値。スクリプトがカスタム属性を検出できない場合、このデフォルト値を使用します。
タイプ	--paramType	パラメーターのデータ型。次の値を指定できます。 '文字列' 'テンプレート'
説明	--paramDesc	パラメーターの説明(テキスト)。

次のUnixコマンドを実行すると、NTP_SERVERという名前のパラメーターがntp-4.2.1 ISMに追加されます。

```
ismtool --addParam NTP_SERVER \
     --paramValue 127.0.0.1 \
     --paramType 'String' \
     --paramDesc 'NTP server, default to loopback' ntp-4.2.1
```

ntp-4.2.1 ISMに追加されたパラメーターを表示するには、次のコマンドを入力します。

```
ismtool --showParams ntp-4.2.1
```

上記で追加したパラメーターを削除するには、次のコマンドを入力します。

```
ismtool --removeParam NTP_SERVER ntp-4.2.1
```

スクリプトでのパラメーターへのアクセス

ISMTool で追加したパラメーターにアクセスするは、ISM コントロールスクリプトを記述します。次のスクリプト言語がサポートされます。

- Bourne Shell
- Korn Shell
- Windowsコマンドシェル
- Python
- Perl

パラメーターへのアクセスには、シェルスクリプトは環境変数、Python スクリプトは辞書、Perl スクリプトはハッシュテーブルを使用します。

ISM parametersユーティリティ

コントロールスクリプトは、parametersユーティリティを実行してパラメーターを取得します。このユーティリティはISM共有ランタイムパッケージに含まれています。取得可能なパラメーターは、--addParamコマンドで定義されているパラメーターのみです。

次にparametersユーティリティの構文を示します。

```
parameters [options]
--scope <scope> ; server|servergroup|customer|facility|
                ; servicelevel|os|custapps|webserver|appserver|
                ; dbserver|systemutilities|osextras|install|
                ; default (default is all)
-s/--sh                 ; Bourne Shell syntax
-k/--ksh                ; Korn-Shell syntax
-p/--python             ; Python repr'ed dictionary
-l/--perl               ; PERL map
-c/--cmd                ; Windows Cmd syntax
-b/--vbscript           ; Windows VBScript syntax
-h/--help               ; Help
-v/--version            ; Version
```

--scopeオプションは、カスタム属性の検索範囲を Server Automation の指定範囲に限定します。たとえば、ファシリティとカスタマーの両方にカスタム属性が定義されている状態で、--scope facilityと指定すると、カスタマーのカスタム属性は対象から除外されます。参照情報: カスタム属性の検索順序 (86ページ)

parametersユーティリティは、処理中にエラーが発生すると、_OPSW_ISMERRという名前の特殊なパラメーターを返します。このパラメーターの値は、発生したエラーの簡単な説明です。

サンプルスクリプト

次の Bourne Shell のサンプルは、Unix で NTP タイムサービスを設定するコントロールスクリプトの例です。parametersユーティリティは、NTP_CONF_TEMPLATEとNTP_SERVERという2つのパラメーターを取得します。このパラメーターはISM向けに設定済みです。

```
#!/bin/sh
. `dirname $0`/../env/ism.sh
```

```
eval `${ISMDIR}/bin/parameters`
echo $NTP_CONF_TEMPLATE | \
sed "s/NTP_SERVER_TAG/$NTP_SERVER/" > /etc/ntp.conf
```

次のコントロールスクリプトもNTPを構成します。Pythonで記述されています。

```
#!/usr/bin/env python
import os
import sys
import string
ismdir=os.path.split(sys.argv[0])[0]
cmd = '%s --python' % (os.path.join(ismdir,'bin','parameters'))
params = eval(os.popen(cmd,'r').read())
template = params['NTP_CONF_TEMPLATE']
value = params['NTP_SERVER']
conf = string.replace(template,'NTP_SERVER_TAG',value)
fd=open('/etc/ntp.conf','w')
fd.write(conf)
fd.close()
```

次の例は、Windows向けの構成コントロールスクリプトです。この例では、32ビット版Windowsオペレーティングシステム向けに、各パラメーターを「名前=値」(1行に1ペア)の形式で出力します。

Windows FORコマンドは、各パラメーターを環境変数として設定します(この例では、FORコマンドは2行に分割されていますが、FORコマンドは1行で記述する必要があります)。最後に、パラメーターはWindowsNTPConfigureScript.cmdという名前のNTP構成スクリプトに渡されます。

```
@echo off
SETLOCAL
cd /d %ISMDIR%
for /f "delims== tokens=1,2" %%i in ('""bin\parameters.cmd""') do set
%%i=%%j WindowsNTPConfigureScript.cmd %NTP_CONF_TEMPLATE% %NTP_SERVER%
ENDLOCAL
```

カスタム属性の検索順序

SAクライアントでは、カスタム属性を複数の場所で設定できます。たとえば、foo.hp.comという名前の管理対象サーバーで、APACHE_1.3_PORTという名前のカスタム属性を8085に設定し、foo.hp.comサーバーに関連づけられているWidget Corp.という名前のカスタマーで、同じカスタム属性を80に設定するとします。実行時、foo.hp.com上でコントロールスクリプトがAPACHE_1.3_PORTパラメーターにアクセスする場合、サーバーのカスタム属性が最初に検索されるので、8085という値が取得されます。

カスタム属性が見つからない場合には、スクリプトはISMToolの--paramValueオプションで設定したデフォルトのパラメーター値を使用します。

デフォルトの検索順序

デフォルトでは、カスタム属性は次の順序で検索されます。

1. サーバー
2. デバイスグループ
3. カスタマー
4. レルム

5 ファシリティ

6 オペレーティングシステム

7 ISM (アップロードでソフトウェアポリシー内に作成)

8 パッチポリシー

9 ソフトウェアポリシー

デバイスグループとサービスレベルが複数ある場合は、アルファベット順に検索されます。たとえば、サーバーがABCグループとXYZグループに含まれる場合、カスタム属性の検索順序は、ABCグループの方がXYZグループより前になります。また、サブグループは、親グループのカスタム属性を継承しません。

インストールスクリプト

インストールスクリプトは、padサブディレクトリに格納されています。インストールフックと同様に、インストールスクリプトは、管理対象サーバーにアプリケーションをインストールまたはアンインストールするプロセスの中で実行されます。

インストールスクリプトとインストールフックの相違点

インストールスクリプトとインストールフックには類似した目的がありますが、次の表で示すように相違点がいくつかあります。

表50 インストールスクリプトとインストールフックの相違点

インストールスクリプト	インストールフック
SAクライアントで、パッケージのプロパティごとに表示されます。	SAクライアントで、パッケージのコンテンツごとに表示されます(表示対象はRPMのみです)。
格納場所はpadサブディレクトリです。	格納場所はism/pkgサブディレクトリです。
モデルリポジトリに保存されます(アップロードの後)。	コントロールパッケージにバンドルされ、管理対象サーバー上の、ctlprefixで指定されたディレクトリにインストールされます。
サーバーエージェントが実行します。	ネイティブパッケージエンジンが実行します。
ISMのパッケージごとに定義できます。	ISM全体として定義します。

インストールスクリプトの作成

ISMToolは、padサブディレクトリ構造を作成しますが、デフォルトのインストールスクリプトは作成しません。ISMToolは、--buildで作成または--addPassthruPkgで追加したパッケージごとに、次のサブディレクトリを作成します。

pad/<パッケージ名>/scripts

たとえばLinuxの場合、--buildコマンドを実行すると、ntp-1.4.2という名前のISM用に次のサブディレクトリが作成されます。

```
pad/ismruntime-rpm-2.0.0-1.i386.rpm/scripts
pad/ntp-ism-4.2.1-1.i386.rpm/scripts
pad/ntp-4.2.1-1.i386.rpm/scripts
```

テキストエディターで、scriptsサブディレクトリにインストールスクリプトを作成します。たとえば、ntp-4.2.1-1.i386.rpmパッケージ用に、次のようなインストールスクリプトを作成します。

```
pad/ntp-4.2.1-1.i386.rpm/scripts/
                                preinstallscript
                                pstinstallscript
                                preuninstallscript
                                pstuninstallscript
```

インストールスクリプトのファイル名には、上記の例で使用したものと同じ名前を指定する必要があります。たとえば、インストールのすぐ後に呼び出すスクリプトの名前は、pstinstallscriptとなります。

インストールスクリプトとインストールフックの呼び出し

ISMでインストールスクリプトとインストールフックが両方作成されている状態で、アプリケーションを管理対象サーバーにインストールする場合、Server Automationは次の順序で次のようなタスクを実行します。

1. ISMランタイムパッケージをインストールします。
2. ISMコントロールパッケージをインストールします。
3. preinstallscript(インストールスクリプト)を実行します。
4. ism_pre_install(インストールフック)を実行します。
5. アプリケーションパッケージ(アプリケーションビット)をインストールします。
6. ism_post_install(インストールフック)を実行します。
7. ism_configure(コントロールスクリプト)を実行します。
8. ism_start(コントロールスクリプト)を実行します。
9. pstinstallscript(インストールスクリプト)を実行します。

管理対象サーバー上のアプリケーションをアンインストールする際、Server Automationは次の順序で次のようなタスクを実行します。

1. preuninstallscript(アンインストールスクリプト)を実行します。
2. ism_pre_uninstall(アンインストールフック)を実行します。
3. ism_stop(コントロールスクリプト)を実行します。
4. アプリケーションパッケージ(アプリケーションビット)をアンインストールします。
5. ism_post_uninstall(アンインストールフック)を実行します。
6. pstuninstallscript(アンインストールスクリプト)を実行します。
7. ISMコントロールパッケージをアンインストールします。
8. ISMランタイムパッケージをアンインストールします。

第7章 IDKコマンド

ISMTool引数タイプ

表51は、この章の後ほど説明するISMToolコマンドで使用する引数タイプの一覧です。たとえば、引数タイプISMNAMEは、ISMTool --newコマンドの構文で指定します。

表51 ISMTool引数タイプ

引数タイプ	説明	例
PATH	ファイルシステムの絶対パスです。	/foo/bar
STRING	スペースを含まないテキスト文字列です。	foobar
TEXT	引用符で囲まれた任意のテキストです。Unixの場合は一重引用符で囲み、Windowsの場合は二重引用符で囲みます。	'This is some text'
BOOL	ブール値です。	trueまたはfalse
ISMFILE	ファイルシステム内の有効な.ismファイルへのパスです。このファイルはISMDIR内に解凍されます。	/foo/bar/name.ism
ISMDIR	解凍済みの有効なISMFILE、または新規作成したISMへのパスです。	xyz /home/sam/xyz
ISMNAME	新しく作成するISMの名前です。ISMNAMEは、STRINGまたはSTRING-VERSIONの形式で指定できます。	ntp ntp-4.1.2
VERSION	ISMのバージョンを表す文字列です。VERSIONは、スペースを含まない、ネイティブパッケージングエンジンの有効なバージョン文字列である必要があります。	1.2.3 4.13 0.9.7b
HOST[:PORT]	ホストと、オプションで指定できるポートです。	www.foo.com www.foo.com:8000 192.168.1.2:8000
BYTES	バイト数を表す整数値です。	42
SECONDS	秒数を表す整数値です。	300
PARAMTYPE	予期されるパラメーターデータのタイプです。指定できる値は、定数'String'および定数'Template'です。Unixの場合は一重引用符で囲み、Windowsの場合は二重引用符で囲みます。	'文字列' 'テンプレート'

ISMTool情報提供コマンド

この項では、ビルド環境の情報を提供するISMToolコマンドについて説明します。

--help

ISMToolのコマンドラインヘルプを表示します。

--env

環境内にあるシステムレベルツールの位置を表示します。このコマンドは、ビルドに関する問題の調査や、環境変数ISMTOOLBINPATHが正しく設定されているかどうかを確認するのに役立ちます。たとえば、Unixシステムで--envを使用すると以下のような結果が表示されます。

```
% ismtool --env
bzip2:/usr/local/ismtool/lib/tools/bin/bzip2
cpio:/usr/local/ismtool/lib/tools/bin/cpio
gzip:/usr/local/ismtool/lib/tools/bin/gzip
install:/usr/local/ismtool/lib/tools/bin/install
17
patch:/usr/local/ismtool/lib/tools/bin/patch
python:/usr/local/ismtool/lib/tools/bin/python
pythonlib:/usr/local/ismtool/lib/tools/lib/python1.5
rpm2cpio:/usr/bin/rpm2cpio
rpm:/bin/rpm
rpmbuild:/usr/bin/rpmbuild
tar:/usr/local/ismtool/lib/tools/bin/tar
unzip:/usr/local/ismtool/lib/tools/bin/unzip
wget:/usr/local/ismtool/lib/tools/bin/wget
zip:/usr/local/ismtool/lib/tools/bin/zip
zipinfo:/usr/local/ismtool/lib/tools/bin/zipinfo
pkgengines:['rpm4']
```

--myversion

ISMToolのバージョンを表示します。

--info ISMDIR

ISMDIRディレクトリにあるISMの内部情報を概略表示します。ビルド完了後は、より詳細な情報を次のURLで確認できます。

```
<ISMDIR>/doc/index/index.html
```

--showParams ISMDIR

各コントロールパラメーターの、名前、デフォルト値、タイプ、説明を表示します。

--showPkgs ISMNAME

ISMが管理しているすべてのパッケージを一覧表示します。一覧には、コントロールパッケージ、アプリケーションパッケージ、パススルーパッケージ、およびパススルーパッケージ内のすべての内部パッケージが含まれます。内部パッケージの例には、Solarisパッケージに含まれるSolarisパッケージインスタンスや、AIX LPPパッケージに含まれる更新ファイルセットがあります。各管理対象パッケージについて、パッケージ名、タイプ、アタッチされたステータス、および設定可能なすべてのメタデータが表示されます。

--showOrder ISMNAME

ISMが管理しているアタッチされたパッケージの現在のインストール順序を表示します。

--showPathProps ISMNAME

このオプションは、Server Automation 6.0では使用できません。

ソフトウェアポリシーメタデータに設定されている値を表示します。

ISMTool作成コマンド

この項では、ISMディレクトリ構造を生成するISMToolコマンドについて説明します。

--new ISMNAME

新しいISM (サブディレクトリとファイルを含むディレクトリ) を作成します。ISMNAMEの値に、新しく作成するISMディレクトリの名前を指定します。ISMの内部名は、ISNAMEの形式によって変わります。

たとえば、次のコマンドはfoobarという名前のISMディレクトリを作成します。ISMの内部名はfoobarで、ISMの初期バージョンはデフォルトで1.0.0になります。

```
% ismtool --new foobar
```

次のコマンドは、ntp-4.1.2という名前のISMディレクトリを作成します。ISMの内部名はntpで、ISMの初期バージョンはデフォルトで4.1.2になります。ISMの内部名に-VERSIONが含まれないことに注意してください。

```
% ismtool --new ntp-4.1.2
```

ISMディレクトリの名前は、ISMの内部名とは独立しています。たとえば、開発者がディレクトリntp-4.1.2をmyntpという名前に変更しても、ISMの内部名は引き続きntpで、バージョンも4.1.2のままです。

--pack ISMDIR

ISMDIR内のISMのZIPアーカイブを作成します。アーカイブ名は、<ismname-version.ism>になります。ただし、ISMDIRの容量は2GB未満である必要があります。(容量が2GBを超える場合は、zipやtarユーティリティを使用してください。)--packの使用例を次に示します。

Unixの場合:

```
% ismtool --new tick
```

```
% ismtool --version 3.14 tick
% ls
tick/
% mv tick spooon
% ls
spooon/
% ismtool --pack spooon
% ls
spooon/ tick-3.14.ism
```

Windowsの場合:

```
% ismtool --new tick
% ismtool --version 3.14 tick
% dir
11/21/2003 10:17a <DIR> tick
% move tick spoon
% dir
11/21/2003 10:17a <DIR> spoon
% ismtool --pack spoon
% dir
11/21/2003 10:17a <DIR> spoon
11/21/2003 10:17a 1,927,339 tick-3.14.ism
```

--unpack ISMFILE

ISMFILEという名前のZIPファイルに含まれるISMを解凍します。ISMは、ISMFILE作成時に--packコマンドで指定したISMDIRに解凍されます。次の例では、--packの使用例で作成したISMFILEを使用しています。

Unixの場合:

```
% ls
spooon/ tick-3.14.ism
% rm -rf spooon
% ls
tick-3.14.ism
% ismtool --unpack tick-3.14.ism
% ls
spooon/ tick-3.14.ism
```

Windowsの場合:

```
% dir
11/21/2003 10:17a <DIR> spoon
11/21/2003 10:17a 1,927,339 tick-3.14.ism
% rmdir /s /q spoon
% dir
11/21/2003 10:17a 1,927,339 tick-3.14.ism
% ismtool --unpack tick-3.14.ism
% dir
11/21/2003 10:17a <DIR> spoon
11/21/2003 10:17a 1,927,339 tick-3.14.ism
```

ISMToolビルドコマンド

この項では、ISMのビルドと変更を行うISMToolコマンドについて説明します。

--verbose

詳細なデバッグ情報を表示します。

--banner

出力バナーの表示を無効化します。

--clean

ビルドによって生成されたすべてのファイルをクリーンアップします。一時ファイルおよびすべてのビルド成果物を削除します。

--build

ISMをビルドし、`pkg`サブディレクトリにパッケージを作成します。

buildコマンドの主な目的は、ISMに含まれるパッケージの作成です。あるいは、ソースのコンパイル、インストール前スクリプトおよびインストール後スクリプトを実行させることもできます。

--upgrade

現在インストールされているISMToolのバージョンに合わせてISMをアップグレードします。

新しくリリースされたISMToolでは、不具合が修正されていたり、解凍先のISMDIRでの動作が変更されていたりする場合があります。現在インストールされているISMToolのバージョンがISMを作成したISMToolのバージョンと異なる場合、特定のアクションを実行しなければならないことがあります。メジャー、マイナーを問わずダウングレードは許可されていないことに注意してください。たとえば、ISMToolバージョン2.0.0で作成したISMは、ISMToolバージョン1.0.0では処理できません。表52は、現在インストールされているISMToolのバージョンと前のバージョンが異なる場合に必要な開発者によるアクションのまとめです。

表52 ISMToolアップグレードアクション

ISMTool バージョン (現在イントールされているもの)	ISMTool バージョン (ISMの作成に使用したもの)	開発者によるアクション
1.0.1	1.0.0	パッチのインクリメント。開発者によるアクションは必要ありません。上位互換および下位互換のある単純な自動アップグレードとみなされます。
1.0.0	1.0.1	パッチのデクリメント。自動的なダウングレード。アクションは必要ありません。
1.1.0	1.0.0	マイナーインクリメント。ISMに--upgradeコマンドを適用する必要があります。多少の動作変更や機能向上がある場合があります。警告:この操作は取り消すことができません。マイナーアップグレードは影響を最小限に留めるようデザインされています。
2.0.0	1.0.0	メジャーインクリメント。ISMに--upgradeコマンドを適用する必要があります。大きな動作変更がある場合があります。リリースノートに示されている他のアクションを実行しなければならない可能性があります。
1.0.0	2.0.0 または1.1.0	メジャーまたはマイナーデクリメント。このダウングレードパスは禁止されています。インストールされているバージョンのISMToolで、このISMを処理することはできません。

--name STRING

ISMの内部名をSTRINGに変更します。解凍したISMのトップレベルディレクトリであるISMDIRは、ISMの内部名と異なる名前にすることができます。両方の名前を変えるには、ISMTool --nameコマンドを使用して内部名を変更し、ファイルシステムコマンドを使用してディレクトリ名を変更します。STRINGの形式がネイティブパッケージングエンジンの有効な形式ではない場合でも、--buildコマンドを実行してパッケージングエンジンがエラーをスローするまで問題は発覚しません。

--version STRING

ISM の内部バージョンフィールドを変更します。STRING にスペースを含めることはできません。--versionコマンドは、STRINGの形式についてそれ以外のチェックを行いません。STRINGの形式がネイティブパッケージングエンジンの有効な形式ではない場合でも、--buildコマンドを実行してパッケージングエンジンがエラーをスローするまで問題は発覚しません。

--prefix PATH

ISMのインストールプレフィックスを変更します。PATHは、ISMToolの「ソースからビルドする」機能や、パッケージングエンジンのドライバーによって使用されます。管理対象サーバー上にインストールする際、ISMにパッケージされているアプリケーションファイルは、PATHを基準とした場所にインストールされます。SAクライアントでは、PATHは、パッケージのプロパティの [インストールパス] フィールドに表示されます。以下のUnixの例では、次の.tarファイルを使用します。

```
% tar tvf ntp/bar/ntp.tar
```

```
-rw-r--r-- root/root      1808 2002-11-22 09:20:36 etc/ntp.conf
drwxr-xr-x ntp/ntp           0 2003-07-08 16:22:38 etc/ntp/
-rw-r--r-- root/root        22 2002-11-22 09:22:08 etc/ntp/step-tickers
-rw-r--r-- ntp/ntp           7 2003-07-08 16:22:38 etc/ntp/drift
-rw------- root/root       266 2001-09-05 03:54:42 etc/ntp/keys
-rwxr-xr-x root/root    252044 2001-09-05 03:54:43 usr/sbin/ntpd
-rwxr-xr-x root/root     40460 2001-09-05 03:54:43 usr/sbin/ntpdate
-rwxr-xr-x root/root     70284 2001-09-05 03:54:43 usr/sbin/ntpdc
-rwxr-xr-x root/root     40908 2001-09-05 03:54:43 usr/sbin/ntp-genkeys
-rwxr-xr-x root/root     66892 2001-09-05 03:54:43 usr/sbin/ntpq
-rwxr-xr-x root/root     12012 2001-09-05 03:54:43 usr/sbin/ntptime
-rwxr-xr-x root/root     40908 2001-09-05 03:54:43 usr/sbin/ntptimeset
-rwxr-xr-x root/root     19244 2001-09-05 03:54:43 usr/sbin/ntptrace
-rwxr-xr-x root/root      1019 2001-09-05 03:54:39 usr/sbin/ntp-wait
```

この例では、--prefixに '/'を指定して、上記のアプリケーションパッケージをファイルシステムのルートを基準にインストールするようビルドします。

```
% ismtool --build --prefix '/' --pkgengine rpm4 ntp
.
.
.
% rpm -qlpv ntp/pkg/ntp-1.0.0-1.i386.rpm
drwxr-xr-x   2 ntp   ntp        0 Jul  8 16:22 /etc/ntp
-rw-r--r--   1 root  root    1808 Nov 22  2002 /etc/ntp.conf
-rw-r--r--   1 ntp   ntp        7 Jul  8 16:22 /etc/ntp/drift
-rw-------   1 root  root     266 Sep  5  2001 /etc/ntp/keys
-rw-r--r--   1 root  root      22 Nov 22  2002 /etc/ntp/step-tickers
-rwxr-xr-x   1 root  root   40908 Sep  5  2001 /usr/sbin/ntp-genkeys
-rwxr-xr-x   1 root  root    1019 Sep  5  2001 /usr/sbin/ntp-wait
-rwxr-xr-x   1 root  root  252044 Sep  5  2001 /usr/sbin/ntpd
-rwxr-xr-x   1 root  root   40460 Sep  5  2001 /usr/sbin/ntpdate
-rwxr-xr-x   1 root  root   70284 Sep  5  2001 /usr/sbin/ntpdc
-rwxr-xr-x   1 root  root   66892 Sep  5  2001 /usr/sbin/ntpq
-rwxr-xr-x   1 root  root   12012 Sep  5  2001 /usr/sbin/ntptime
-rwxr-xr-x   1 root  root   40908 Sep  5  2001 /usr/sbin/ntptimeset
-rwxr-xr-x   1 root  root   19244 Sep  5  2001 /usr/sbin/ntptrace
```

インストールプレフィックスを '/usr/local'に変更するのは簡単です。次のようにします。

```
% ismtool --build --prefix '/usr/local' ntp
.
.
.
% rpm -qlpv ntp/pkg/ntp-1.0.0-2.i386.rpm
drwxr-xr-x   2 ntp   ntp        0 Jul  8 16:22 /usr/local/etc/ntp
-rw-r--r--   1 root  root    1808 Nov 22  2002 /usr/local/etc/ntp.conf
-rw-r--r--   1 ntp   ntp        7 Jul  8 16:22 /usr/local/etc/ntp/drift
-rw-------   1 root  root     266 Sep  5  2001 /usr/local/etc/ntp/keys
-rw-r--r--   1 root  root      22 Nov 22  2002 /usr/local/etc/ntp/step-
tickers
-rwxr-xr-x   1 root  root   40908 Sep  5  2001 /usr/local/usr/sbin/ntp-
genkeys
-rwxr-xr-x   1 root  root    1019 Sep  5  2001 /usr/local/usr/sbin/ntp-wait
-rwxr-xr-x   1 root  root  252044 Sep  5  2001 /usr/local/usr/sbin/ntpd
-rwxr-xr-x   1 root  root   40460 Sep  5  2001 /usr/local/usr/sbin/ntpdate
-rwxr-xr-x   1 root  root   70284 Sep  5  2001 /usr/local/usr/sbin/ntpdc
```

```
-rwxr-xr-x   1 root  root   66892 Sep  5  2001 /usr/local/usr/sbin/ntpq
-rwxr-xr-x   1 root  root   12012 Sep  5  2001 /usr/local/usr/sbin/ntptime
-rwxr-xr-x   1 root  root   40908 Sep  5  2001 /usr/local/usr/sbin/
ntptimeset
-rwxr-xr-x   1 root  root   19244 Sep  5  2001 /usr/local/usr/sbin/ntptrace
```

Windowsの場合は、MSIに対してインストール場所を指定する標準的な方法はありません。そのため、barディレクトリのMSIファイルからビルドされたアプリケーションパッケージの場合、--prefixの設定は無視されます。ただし、ZIPファイルからビルドしたWindows用アプリケーションパッケージの場合は、--prefixの設定が使用されます。Windowsでは、プレフィックスは以下の形式である必要があります。driveletter:\directoryname (たとえば、D:\mydir)。Windows NT4では、--prefixは必須であり、変数を含めることはできません。

Unixでは、PATHのデフォルト値は/usr/localです。

--ctlprefix PATH

コントロールファイルのインストールプレフィックスを変更します。このコマンドは推奨されていないため、代わりにデフォルト値を使用するようにします。管理対象サーバー上にインストールする際、ISMにパッケージされているコントロールファイルは、PATHを基準とした場所にインストールされます。SAクライアントでは、PATHは、パッケージのプロパティの[インストールパス]フィールドに表示されます。Windowsでは、プレフィックスは以下の形式である必要があります。driveletter:\directoryname (たとえば、D:\mydir)。Windows NT4では、--ctlprefixは必須であり、変数を含めることはできません。

PATHのデフォルト値は、以下のとおりです。

Unixの場合:

/var/opt/OPSWism

Windowsの場合:

%ProgramFiles%\OPSWism

Solarisで--ctlprefixを指定すると、共有実行時パッケージの名前を入力するよう求められます。

--user STRING (Unixのみ)

アプリケーションパッケージ内のファイルの所有ユーザーをSTRINGに変更します。パッケージ内のファイルが管理対象サーバーにインストールされると、ファイルの所有者は指定したUnixユーザーになります。

--group STRING (Unixのみ)

アプリケーションパッケージ内のファイルの所有グループをSTRINGに変更します。

--ctluser STRING (Unixのみ)

コントロールパッケージ内のファイルの所有ユーザーを STRING に変更します。デフォルト値は root です。パッケージ内のファイルが管理対象サーバーにインストールされると、ファイルの所有者は指定した Unix ユーザーになります。

--ctlgroup STRING (Unixのみ)

コントロールパッケージ内のファイルの所有グループをSTRINGに変更します。デフォルト値はbinです。

--pkgengine STRING (Unixのみ)

ネイティブパッケージングエンジンを変更します。複数のパッケージングエンジンを利用できるシステムの場合、このコマンドでエンジンを切り替えます。利用可能なエンジンを表示するには、--helpコマンドまたは--envコマンドを実行します。

ただし、ネイティブパッケージングエンジンを変更した場合、--uploadの実行時に一切のパッケージがソフトウェアポリシーに追加されなくなるので注意してください。

--ignoreAbsolutePaths BOOL (Unixのみ)

アーカイブ内の絶対パスを無視します。たとえば、次に挙げるのは絶対パスを含むバイナリアーカイブです。

```
% tar tvf test/bar/foo.tar
-rw-r--r-- root/root      1808 2002-11-22 09:20:36 /foo/bar/baz.conf
```

この場合、--prefixに/usr/localを指定するとインストールパスがあいまいになります。baz.confは、/foo/bar/baz.confにインストールされるべきでしょうか? それとも、/usr/local/foo/bar/baz.conf にインストールされるべきなのでしょうか?正解が/foo/bar/baz.confなら、ISMの--prefixに'/'を指定する必要があります。逆に、正解が/usr/local/foo/bar/baz.confの場合は、--ignoreAbsolutePathsコマンドを指定する必要があります。

--addCurrentPlatform (Unixのみ)

現在のプラットフォームを、ISMのサポート対象リストに追加します。注: このコマンドによってISMがクロスプラットフォームになるわけではありません。ISMは、SAをサポートする複数のプラットフォームで構築できます。プラットフォームとは、OSの種類とバージョンの組み合わせです。プラットフォームの例には以下のものがあります。Redhat-Linux-7.2、SunOS-5.9、Windows-2000。ISMが現在サポートしているプラットフォームを確認するには--infoコマンドを使用します。

--removeCurrentPlatform (Unixのみ)

現在のプラットフォームを、ISMのサポート対象リストから削除します。

--addPlatform TEXT (Unixのみ)

TEXTに指定したプラットフォームを、ISMのサポート対象リストに追加します。プラットフォームのサポートおよび識別は動的に行われるため、--addPlatformの実行時にエラーチェックは行われません。このため、--addPlatformではなく--addCurrentPlatformを使用することが推奨されます。

--removePlatform TEXT (Unixのみ)

TEXTに指定したプラットフォームを、ISMのサポート対象リストから削除します。

--target STRING (Unixのみ)

警告: 専門知識がない場合はこのコマンドを使用しないでください。

RPM パッケージングエンジンでクロスプラットフォームのアプリケーションパッケージ作成を有効にします。--targetコマンドは、--skipControlPkgコマンドとともに使用する必要があります。STRINGの形式は、<arch-os>です。たとえば、i686-linux、sparc-solaris2.7などがあります。

--skipControlPkg BOOL

コントロールパッケージをビルドしないようにします。このコマンドにより、構造化されたアプリケーションコントロールパッケージを必要としないファイルをISMToolでパッケージングできます。

--skipApplicationPkg BOOL

アプリケーションパッケージをビルドしないようにします。このコマンドにより、コントロールパッケージのみのISMパッケージをISMToolで作成できます。この機能を使用することで、インストールやパッケージングにISMToolを使用しないアプリケーションのためのコントローラーをビルドできます。例として、オペレーティングシステムのコア機能、実行中のためパッケージングできないアプリケーション、専用ハードウェアなどのコントローラーがあります。

--chunksize BYTES (Unixのみ)

アプリケーションパッケージの最大容量をバイトで指定します。(圧縮に関する諸要素はヒューリスティクスで決定されます。)バイナリアーカイブ (bar) ディレクトリには、アプリケーションパッケージの元になるアーカイブが多数格納されていることがあります。アーカイブの容量が指定したチャンクサイズを超過すると、アプリケーションアーカイブは複数のビンにグループ分けされ、各ビンがサブアプリケーションパッケージに転換されます。アルゴリズムは標準的なビンパッキングヒューリスティックです。移動可能な単位は、barディレクトリ内のバイナリアーカイブです。

たとえば、パッケージ出力形式がRPM で、5 つのバイナリアーカイブがあるとします。a.tgz (100M)、b.tgz(100M)、c.tgz (200M)、d.tgz (300M)、およびe.tgz (50M) です。チャンクサイズを314572800 (300M) に設定した場合、出力されるアプリケーションビンは以下のとおりになります。

```
part1( a.tgz, b.tgz, e.tgz ) == 250M
part2( c.tgz )               == 200M
part3( d.tgz )               == 300M
```

これにより3つのアプリケーションパッケージが作成されます。

```
foobar-part0-1.0.0.i386.rpm
foobar-part1-1.0.0.i386.rpm
foobar-part2-1.0.0.i386.rpm
```

通常、アプリケーションパッケージのサイズが1ギガバイト近くにならない限り、チャンクサイズが問題になることはありません。サイズが1ギガバイト近くになると、パッケージエンジンによっては動作に支障をきたすことがあります。デフォルトのチャンクサイズは1ギガバイト (2^30バイト) です。

--solpkgMangle BOOL (SunOSのみ)

アプリケーションパッケージ名がSolarisの要件に合わせて変更されないようにします。詳細については、Solaris (66ページ) を参照してください。

Solarisのパッケージを作成する場合、パッケージ名は9文字以内に制限されます。ただし、--buildプロセス中にパッケージ名がISMTool によって変更 (“マングリング”) されるのを防ぎたい場合もあります。--solpkgMangleにfalseを指定すると、ISMToolはISM名を使用してアプリケーションパッケージを作成します。ただし、コントロールパッケージの名前が変更されるのを防ぐことはできません。--solpkgMangleにfalseを指定する際は、ISM名を9文字以内にする必要があることと、複数のアプリケーションパッケージを含めることができない、という点に注意してください。

--embedPkgScripts BOOL

ISMパッケージングスクリプト (インストールフック) の内容をアプリケーションパッケージに埋め込みます。このオプションは、--skipControlPkgおよび--skipRunTimePkgとともに使用する必要があります。

デフォルトでは、アプリケーションパッケージはコントロールパッケージによってインストールされるISMパッケージングスクリプトを呼び出すように作成されます。--embedPkgScriptsオプションは、アプリケーションパッケージ内のism/pkgディレクトリにあるスクリプトの内容を埋め込むことでデフォルトの動作を上書きします。これらのスクリプトは、アプリケーションパッケージのインストールとアンインストールの前と後のフェーズに実行されます。

ism/pkgディレクトリのスクリプトの中に不要なものがある場合は、--buildプロセスを開始する前に削除してください。RPMおよびLPPパッケージングエンジンは、checkinstallフェーズがないため、RPM/LPPのビルド時にism_check_installファイルは無視されます。

--skipRuntimePkg BOOL

以後の--build操作で実行時パッケージをビルドするかどうかを指定します。

デフォルトでは、実行時パッケージはビルドされます。--skipRuntimePkgにtrueを指定すると、--skipRuntimePkgにfalseを指定するまで、実行時パッケージがビルドされなくなります。parametersインタフェースなどのISMユーティリティは、実行時パッケージが見つからないと実行できません。ISMをインストールしようとしている管理対象サーバーに実行時パッケージが存在している確証がない場合は、--skipRuntimePkgにtrueを指定しないでください。

ISMToolインタフェースコマンド

この項では、SAと対話するためのISMToolコマンドについて説明します。

--upload

ISMDIR内のISMを、
--opswpathで指定したソフトウェアポリシーにアップロードします。存在しないソフトウェアポリシーを指定した場合は、アップロードプロセスの最中にソフトウェアポリシーが自動的に作成されます。接続するSAコアを指定するには、コマンドライン引数 (--softwareRepositoryなど) または表53に示されている環境変数を使用します。

--uploadコマンドを使用すると、SAのユーザー名とパスワードの入力を求められます。アップロード操作を行う前に、ismusertoolを使用してそのユーザーにアクセス権を与える必要があります。また、ユーザーはソフトウェアポリシーを格納しているフォルダーの書き込み権限を持っている必要があります。

--noconfirm

yまたはnによる返答を求められる確認メッセージを表示しないようにします。ISMToolの確認メッセージには以下のようなものがあります。

```
Do you wish to proceed with upload? [y/n]:
```

--noconfirmを指定すると確認メッセージは表示されなくなり、すべての答えはyとみなされます。--noconfirmオプションが影響を与えるのは、現在起動中のISMToolだけです。

--opswpath STRING

アップロードされたISMを格納するソフトウェアポリシーのパスを指定します。パスには常にスラッシュが含まれます (Windowsでも)。

存在しないソフトウェアポリシーを指定した場合は、アップロードプロセスの最中にソフトウェアポリシーが自動的に作成されます。フォルダー (終端がポリシーでないパス) を指定した場合はエラーが発生します。ISMをフォルダーにアップロードすることはできないためです。

ISMTool はクロスプラットフォーム ISM の作成をサポートしています。クロスプラットフォーム ISM の例として、ネットワークタイムプロトコル (NTP) デーモンがあります。NTPはさまざまなプラットフォームでソースからビルド できます。ISMTool は、クロスプラットフォーム ISM のアップロード を容易にするために、`--opswpath STRING`内の変数展開をサポートしています。この変数は、ISM の内部設定を示します。表 53 は、ISMToolで認識される変数を示しています。

表53　　ISMTool変数

変数	例
${NAME}	ntp
${VERSION}	4.1.2
${APPRELEASE}	3
${CTLRELEASE}	7
${PLATFORM}	Redhat Linux 7.2
${OSTYPE}	Redhat Linux
${OSVERSION}	7.2

Unixの例:

```
% ismtool --opswpath '/System Utilities/${NAME}/${VERSION}/${PLATFORM}' ntp
```

展開例:

```
'/System Utilities/ntp/4.1.2/Redhat Linux 7.2'
```

Windowsの例:

```
% ismtool --opswpath "/System Utilities/${NAME}/${VERSION}/${PLATFORM}" ntp
```

展開例:

```
"/System Utilities/ntp/4.1.2/Windows 2000"
```

--commandCenter HOST[:PORT]

フォルダーへのアップロードに、`HOST[:PORT]` にあるOpswareコマンドセンターコアコンポーネントを使用します。

--dataAccessEngine HOST[:PORT]

アップロードに、`HOST[:PORT]` にあるSAデータアクセスエンジンを使用します。

--commandEngine HOST[:PORT]

アップロードに、HOST[:PORT]にあるSAコマンドエンジンを使用します。

--softwareRepository HOST[:PORT]

アップロードに、HOST[:PORT]にあるSAソフトウェアリポジトリを使用します。

--description TEXT

ISMの説明文を入力します。このテキストは、アップロード時にソフトウェアポリシーの[説明]フィールドにコピーされます。

--addParam STRING

STRINGという名前のパラメーターをISMに追加します。通常、--paramValue、--paramDesc、--paramTypeの各コマンドを併せて指定します。例:

```
% ismtool --addParam NTP_SERVER \
          --paramValue 127.0.0.1 \
          --paramType 'String' \
          --paramDesc 'NTP server, default to loopback' ntp

% ismtool --addParam NTP_CONF_TEMPLATE \
          --paramValue /some/path/ntp.conf.template \
          --paramType 'Template' \
          --paramDesc 'Template for the /etc/ntp.conf file' ntp
```

--paramValue TEXT

パラメーターのデフォルト値を設定します。--addParamコマンドを同時に指定する必要があります。パラメータータイプが'String'の場合、デフォルト値をTEXTで指定します。パラメータータイプが'Template'の場合は、TEXTは構成テンプレートファイルへのパスとして解釈されます。そして、テンプレートファイル内のデータがデフォルト値としてロードされます。--paramValueと--paramTypeを指定しないと、デフォルト値は空文字列になります。

--paramType PARAMTYPE

パラメータータイプを設定します。--addParamコマンドを同時に指定する必要があります。PARAMTYPEは、'String'または'Template'である必要があります。デフォルトのタイプは'String'です。

--paramDesc TEXT

パラメーターの説明文を設定します。--addParamコマンドを同時に指定する必要があります。デフォルト値は空文字列です。

--removeParam STRING

STRINGという名前のパラメーターを削除します。

--rebootOnInstall BOOL

アプリケーションパッケージに、SAパッケージ制御フラグreboot_on_installを付加します。--rebootOnInstallにtrueを指定すると、管理対象サーバーはパッケージのインストール後再起動します。ISMに複数のアプリケーションパッケージがある場合、リストの最後のパッケージにフラグが付けられます。

--rebootOnUninstall BOOL

アプリケーションパッケージに、SAパッケージ制御フラグreboot_on_uninstallを付加します。--rebootOnUninstallにtrueを指定すると、管理対象サーバーはパッケージのアンインストール後再起動します。ISMに複数のアプリケーションパッケージがある場合、リストの最初のパッケージにフラグが付けられます。

--registerAppScripts BOOL (Windowsのみ)

ISMパッケージングスクリプト (インストールフック) をアプリケーションパッケージに登録します。

デフォルトでは、ISMパッケージングスクリプトはインストール前、インストール後、アンインストール前、アンインストール後に実行されるようアプリケーションMSIにエンコードされます。--registerAppScriptsを指定すると、ISMパッケージングスクリプトはデフォルトの動作とは異なり、アップロード時にSAパッケージコントロールスクリプトとして登録されます。パッケージコントロールスクリプトはモデルリポジトリに登録され、HP Server Automationクライアントで確認できます。

MSIによるアプリケーションのインストールと競合するアクションがISMパッケージングスクリプトに含まれている場合は、--registerAppScriptsコマンドの指定が必要です。たとえば、インストール後スクリプトがmsiexec.exeを呼び出すと競合が起きる場合があります。Microsoftインストーラーが同時インストールを許容していないため、msiexec.exeを呼び出したスクリプトは正常に終了しません。ISMパッケージングスクリプトをSAパッケージコントロールスクリプトとして登録することで、MSIによるインストール/アンインストールの外側でスクリプトを呼び出すことができます。

--endOnPreIScriptFail BOOL (Windowsのみ)

アプリケーションパッケージのインストールを終了します。

--endOnPreIScriptFailと--registerAppScriptsの両方にtrueを指定すると、ISMインストール前スクリプトがゼロ以外の終了コードを返した場合にインストールが中止されます。

--endOnPstIScriptFail BOOL (Windowsのみ)

アプリケーションパッケージのインストールを終了します。

--endOnPstIScriptFailと--registerAppScriptsの両方にtrueを指定すると、ISMインストール後スクリプトがゼロ以外の終了コードを返した場合にインストールが中止されます。

--endOnPreUScriptFail BOOL (Windowsのみ)

アプリケーションパッケージのアンインストールを終了します。

--endOnPreUScriptFailと--registerAppScriptsの両方にtrueを指定すると、ISMアンインストール前スクリプトがゼロ以外の終了コードを返した場合にアンインストールが中止されます。

--endOnPstUScriptFail BOOL (Windowsのみ)

アプリケーションパッケージのアンインストールを終了します。

--endOnPstUScriptFailと--registerAppScriptsの両方にtrueを指定すると、ISMアンインストール後スクリプトがゼロ以外の終了コードを返した場合にアンインストールが中止されます。

--addPassthruPkg {PathToPkg} --pkgType {PkgType} ISMNAME

{PathToPkg}で指定したパッケージをパススルーパッケージとして扱います。サポートされるパッケージタイプ{PkgType}は、表54に示すようにプラットフォームによって異なります。

{PathToPkg}はパッケージへの完全パスまたは相対パスを指定します。ただし、--addPassthruPkgオプションを指定した時点でパッケージが存在している必要があります。{PathToPkg}に、現在のISMのディレクトリ構造に含まれるパッケージを指定することはできません。たとえば、コントロールパッケージ、アプリケーションパッケージ、またはbarディレクトリ内のパッケージをパススルーパッケージとして指定することはできません。

デフォルトでは、アップロード操作はパススルーパッケージ (--addPassthruPkgで指定したもの) をソフトウェアポリシーに追加しません。パススルーパッケージを追加するには、--attachPkgオプションを指定する必要があります。

Solarisパススルーパッケージのアップロードでは、応答ファイルはアップロードされません。応答ファイルは、手動でアップロードする必要があります。

各プラットフォームで使用できる{PkgType} (パッケージタイプ) の値を次の表に示します。

表54　　パススルーオプションがサポートするパッケージタイプ

プラットフォーム (OS)	{Pkgtype}に指定できる値
AIX	lpp rpm zip
HP-UX	depot zip
Linux	rpm zip
SunOS	rpm solcluster solpatch solpkg ips zip
Windows	hotfix msi sp zip

次の例は、ISMへのパススルーパッケージの追加とソフトウェアポリシーへのパッケージの追加を行う方法を示しています。

```
% ismtool --addPassthruPkg /tmp/bos.rte.libs.5.1.0.50.U --pkgType lpp ISMNAME
Inspecting specified package:...
bos.rte.libs.5.1.0.50.U (lpp)
  bos.rte.libs-5.1.0.50 (update fileset)
  IY42527 (apar)
Done.
% ismtool --attachPkg bos.rte.libs-5.1.0.50 --attachValue true ISMNAME
```

--removePassthruPkg {PassthruPkgFileName} ISMNAME

登録済みのパススルーパッケージがパススルーパッケージではなくなったことを指定します。

ISMToolは以下の操作を行います。

1. ISMのディレクトリ構造から{PassthruPkgFileName}を削除します。
2. {PassthruPkgFileName}がパススルーパッケージではなくなったことをism.confに記録します。
3. 次回以降のすべてのアップロードにおいて、そのパッケージが--opswpathソフトウェアポリシーに追加されたら、そのパッケージを削除します。

ISM はパススルーパッケージから削除されたパッケージをすべて記憶しています。SA クライアントまたは前回のアップロード操作でパッケージが追加された場合は、次回のアップロード操作でパッケージがポリシーから削除されます。

--attachPkg {PkgName} --attachValue BOOLEAN ISMNAME

ISMの管理対象パッケージを--opswpathで示すソフトウェアポリシーに追加するかどうかを指定します。

デフォルトでは、コントロールまたはアプリケーションタイプのパッケージは、作成されるとソフトウェアポリシーに追加するためのマークが付けられます。一方、パススルーパッケージと内部パッケージは、--attachPkgオプションを指定しないと追加のマークが付けられません。

{PkgName}は、--showPkgsコマンドで表示されるパッケージ名です。もし --attachValueがtrueなら、パッケージにソフトウェアポリシーに追加するためのマークが付けられます。もし --attachValueがfalseなら、パッケージはソフトウェアリポジトリにアップロードされますが、ソフトウェアポリシーには追加されません。--attachValue が false で、なおかつパッケージがすでにソフトウェアポリシーに入っている場合は、パッケージにはポリシーから削除するためのマークが付けられます。パッケージの追加と削除は--upload実行時に行われます。ソフトウェアポリシーに追加できるパッケージタイプを次の表に示します。

表55　　パッケージタイププロパティ

パッケージタイプ	スクリプトを含めることができるか?	ソフトウェアポリシーに追加できるか?
AIX LPP	いいえ	いいえ
AIXベースファイルセット	はい	はい
AIX更新ファイルセット	はい	はい
AIX APAR	いいえ	はい
HP-UXデポ	いいえ	いいえ
HP-UXファイルセット	はい	はい

表55　パッケージタイププロパティ（続き）

パッケージタイプ	スクリプトを含めることができるか?	ソフトウェアポリシーに追加できるか?
HP-UXパッチファイルセット	いいえ	いいえ
HP-UX製品	いいえ	はい
HP-UXパッチ製品	いいえ	はい
IPSパッケージ	いいえ	はい
RPM	はい	はい
Solarisパッケージ	いいえ	いいえ
Solarisパッケージインスタンス	はい	はい
Solarisパッチ	はい	はい
Solarisパッチクラスター	いいえ	はい
Windowsホットフィックス	はい	はい
Windows MSI	はい	はい
Windowsサービスパック	はい	はい
Windows ZIPファイル	はい	はい

--orderPkg {PkgName} --orderPos {OrderPos} ISMNAME

ISMが管理しているアタッチされたパッケージのインストール順序を変更します。

{OrderPos}は、{PkgName}で指定したパッケージの新しいインストール順序を示す整数値です。最初にインストールするパッケージの{OrderPos}は1です (0ではありません)。インストール順序を表示するには、`ismtool --showOrder`コマンドを使用します。

次の例では、インストール順序の表示と変更を行います。

```
% ismtool --showOrder ISMNAME
[1] test-ism-1.0.0-1.rpm
[2] test-1.0.0-1.rpm
[3] bos.rte.libs-5.1.0.50
[4] IY42527

% ismtool --orderPkg IY42527 --orderPos 1 ISMNAME
[1] IY42527
[2] test-ism-1.0.0-1.rpm
[3] test-1.0.0-1.rpm
[4] bos.rte.libs-5.1.0.50
```

--addPathProp {PathProp} --propValue {PropValue} ISMNAME

ソフトウェアポリシーのプロパティ (メタデータ) に値を設定します。

現在の値を表示するには、--showPathPropsコマンドを使用します。--addPathPropコマンドで使用できる値を次の表に示します。

表56　{PathProp}に設定できる値

{PathProp}に設定できる値	{PropValue}タイプ	例
description	TEXT	'This does something important'
非推奨: notes	TEXT	'And so does this'
非推奨: allowservers	BOOLEAN	false

次の例では、descriptionプロパティを設定するコマンドを示します。

```
% ismtool --addPathProp description --propValue 'This policy does something'
ISMNAME
% ismtool --showPathProps ISMNAME
description:   This policy does something
```

--editPkg {PkgName} --addPkgProp {PkgProp} --propValue {PropValue} ISMNAME

指定したパッケージのメタデータプロパティに値を設定します。

{PkgName}に更新するパッケージを指定します。--showPkgsコマンドで表示されるいずれかのパッケージ名を指定します。次の表に、{PkgProp}に設定できる値を示します。

表57　{PkgProp}に設定できる値

{PkgProp}に設定できる値	説明	{PropValue}タイプ
deprecated	パッケージの非推奨ステータス	BOOLEAN
description	パッケージの説明	TEXT
endonpreiscriptfail	インストール前スクリプトが失敗したら修復が終了する	BOOLEAN
endonpreuscriptfail	アンインストール前スクリプトが失敗したら修復が終了する	BOOLEAN
endonpstiscriptfail	インストール後スクリプトが失敗したら修復が終了する	BOOLEAN
endonpstuscriptfail	アンインストール後スクリプトが失敗したら修復が終了する	BOOLEAN
installflags	パッケージのインストールフラグ	TEXT
notes	パッケージに関する注	TEXT

表57　{PkgProp}に設定できる値

{PkgProp}に設定できる値	説明	{PropValue}タイプ
rebootoninstall	パッケージのインストール後に再起動が必要	BOOLEAN
rebootonuninstall	パッケージのアンインストール後に再起動が必要	BOOLEAN
uninstallflags	パッケージのアンインストールフラグ	TEXT

endonXXXscriptfailの値は、パッケージにインストール前/後スクリプトまたはアンインストール前/後スクリプトが定義されている場合にだけ設定できます。これらのスクリプトは、ISMNAME/padサブディレクトリにあります。

すべてのパッケージタイプが、前表のすべての{PkgProp}の値をサポートしているわけではないことに注意してください。各パッケージがサポートする{PkgProp}の値を確認するには、SAクライアントでパッケージプロパティの詳細情報を表示します。また、次の表にも各パッケージタイプがサポートする{PkgProp}の値を示します。

表58　各パッケージタイプで設定できる{PkgProp}の値

{PkgProp}に設定できる値	パッケージタイプ	説明	{PropValue}
upgradeable	RPM	パッケージがアップグレードできる	BOOLEAN
productname	Windows MSI	MSIの製品名	STRING
productversion	Windows MSI	MSIのバージョン番号	STRING
servicepacklevel	Windows OS サービスパック	サービスパックのバージョン番号	INTEGER
installdir	Windows ZIP	インストールディレクトリ	STRING
postinstallscriptfilename	Windows ZIP	インストール後スクリプトのファイル名	STRING
postinstallscriptfilenamefail	Windows ZIP	インストール後スクリプトが失敗したら修復が終了する	BOOLEAN
preuninstallscriptfilename	Windows ZIP	アンインストール前スクリプトのファイル名	STRING
preuninstallscriptfilenamefail	Windows ZIP	アンインストール前スクリプトが失敗したら修復が終了する	BOOLEAN

productversion、productname、およびservicepacklevelは、--upload操作を行う前に設定する必要があります。productname と productversion は、1 度 --upload 操作を行うと変更できなくなります。productnameまたはproductversionを変更後、再度--upload操作を行っても変更は適用されません。

次の例は、パッケージの説明文を設定する方法を示します。

```
% ismtool --editPkg bos.rte.libs.5.1.0.50 --addPkgProp description --
propValue 'This is a fileset' ISMNAME
```

ISMTool環境変数

ISMToolはこの項で説明するシェルの環境変数を参照します。

CRYPTO_PATH

agent/agent.srvファイルとagent/opsware-ca.crtファイルを格納するディレクトリを示す環境変数です。

CRYPTO_PATH、agent.srv、opsware-ca.crtが必要になるのは、SAによって管理されていないサーバー(つまりサーバーエージェントがないサーバー)からISMをアップロードする場合だけです。

ISMのアップロード中にISMToolがSAコアに接続するには、HP Server Automationのインストール中に生成されたクライアント証明書(agent.srvファイルとopsware-ca.crtファイル)が必要です。

ISMToolでこれらのクライアント証明書を使用する場合、SAクライアントとは異なるセキュリティメカニズムが呼び出されるので注意が必要です。その結果、より大きなアクセス権を得たり、アクセス権を失ったりすることがあります。通常ではアクセスできないはずの顧客のサーバーにアクセスできる場合もあります。

また、SAクライアントでは不可能な操作を実行できるかもしれません。そのため、このような状況になったら、セキュリティ権限の変更によって意図せぬ結果が生じないように注意深くISMToolを使用する必要があります。

agent.srvとopsware-ca.crt ファイルを取得して、環境変数CRYPTO_PATHを設定するには次の手順を実行します。

1. rootとしてSAコアサーバーにログインし、次のファイルを見つけます。

   ```
   /var/opt/opsware/crypto/agent/agent.srv
   /var/opt/opsware/crypto/agent/opsware-ca.crt
   ```

2. agent.srvとopsware-ca.crtを、IDKをインストールしたサーバーの次のディレクトリにコピーします。

   ```
   <some-path>/agent
   ```

   ディレクトリパスの<some-path>の部分は任意で構いませんが、agent.srvとopsware-ca.crtがあるサブディレクトリはagentでなければなりません。

3. 環境変数CRYPTO_PATHは<some-path>に設定します。

   たとえば、Unixサーバーでagent.srvとopsware-ca.crtのフルパス名が次のとおりだとします。

   ```
   /home/jdoe/dev/
   ```

   cshの場合、次のようにして環境変数を設定します。

   ```
   setenv CRYPTO_PATH /home/jdoe/dev/
   ```

   Windowsで、agent\agent.srvとagent\opsware-ca.crtが次の場所にあるとします。

   ```
   C:\jdoe\dev\
   ```

   この場合、次のように環境変数を設定します。

   ```
   set CRYPTO_PATH=C:\jdoe\dev\
   ```

ISMTOOLBINPATH

この環境変数には、ISMToolがシステムレベルのツール (tarやcpioなど) を検索するディレクトリ名をコロンで区切って指定します。検索は以下の方法で行われます。

1. 環境変数ISMTOOLBINPATHのパスを検索します。
2. /usr/local/ismtool/lib/tools/bin内のコンパイル済みバイナリを検索します (もしあれば)。
3. ユーザーのパスを検索します。

ISMTOOLCC

ISMToolがフォルダーへのアップロード時に使用するOpswareコマンドセンターコアコンポーネントのHOST[:PORT] を示す環境変数です。

ISMTOOLCE

ISMToolが使用するSAコマンドエンジンのHOST[:PORT]を示す環境変数です。

ISMTOOLDA

ISMToolが使用するSAデータアクセスエンジンのHOST[:PORT]を示す環境変数です。

ISMTOOLPASSWORD

この環境変数は、ISMToolでアップロードする際のSAパスワードを示す文字列です。

ISMTOOLSITEPATH

“site” ディレクトリのパスを示す環境変数です。

ISMToolには、新しいISMを作成するときに参照するデフォルトのスクリプトおよび属性値 (インストールプレフィックスなど) が含まれています。開発者は、サイトディレクトリを使用することで、デフォルトのスクリプトや属性値をオーバーライドできます。

defaults.confファイル

サイトディレクトリの中には、上書きする属性値を記入したdefaults.conf ファイルを作成できます。defaults.conf内の行は次の形式です。<タグ>:<値>。#という文字で始まる行はコメントです。次の例は、defaults.confで設定できる値を示します。

Unixの場合:

```
prefix:    /usr/local
ctlprefix: /var/opt/OPSWism
opswpath:  /System Utilities/${NAME}/${VERSION}/${PLATFORM}
version:   1.0.0
ctluser:   root
ctlgroup:  bin
```

Windowsの場合:

```
prefix:    ???
ctlprefix: ???
opswpath:  /System Utilities/${NAME}/${VERSION}/${PLATFORM}
version:   1.0.0
```

templatesサブディレクトリ

ISMTOOLSITEPATHのtemplatesサブディレクトリに独自のコピーを置くことで、/usr/local/ismtool/lib/ismtoollib/templatesディレクトリ内のファイルをオーバーライドできます。たとえば、WindowsまたはUnixのデフォルトのパッケージフックファイルをオーバーライドできます。

controlサブディレクトリ

時として、共通で使用するツールをISMのcontrolディレクトリにインストールしなければならない場合があります。ISMToolはそのような要件にも対応しています。ISMTOOLSITEPATHのcontrolサブディレクトリにあるすべてのファイルをISMのcontrolディレクトリにコピーすることでそれを実現します。ISMのcontrolディレクトリにすでに同じファイルがある場合は上書きされません。

ISMTOOLSR

ISMToolが使用するSAソフトウェアリポジトリのHOST[:PORT]を示す環境変数です。

ISMTOOLUSERNAME

この環境変数は、ISMToolでアップロードする際のSAユーザー名を示す文字列です。

ISMUserTool

ISMToolの--uploadコマンドを使用すると、SAユーザー名の入力を求められます。SAユーザーがアップロードを実行できるようにするには、ISMUsertoolを実行して権限を割り当てます。

アップロード権限のあるユーザーを表示するには以下のコマンドを実行します。

```
% ismusertool --showUsers
```

ユーザーにアップロード権限を与えるには以下のコマンドを実行します。

```
% ismusertool --addUser johndoe
```

アップロード権限を取り消すには以下のコマンドを実行します。

```
% ismusertool --removeUser johndoe
```

ISMUsertoolはコマンドラインで複数のオプションを指定できます。コマンドについての詳細を表示するには、--helpオプションを指定します。

```
% ismusertool --help
```

デフォルトでは、Opswareadminユーザーにアップロード権限が与えられており、その権限は取り消すことができません。

フォルダーは、Server Automationバージョン6.0の新機能です。ISMをフォルダーにアップロードするには、ユーザーがフォルダー権限を持っている必要があります。デフォルトでは、adminユーザーはフォルダー権限を持っていません。本番環境の場合、adminユーザーにフォルダー権限を与えるのはふさわしくないので、アップロードにadminを使用しないでください。