HP Server Automation

HP-UX、IBM AIX、Red Hat Enterprise Linux、Solaris、SUSE Linux Enterprise Server、VMware、Windows®オペレーティングシステム向け

ソフトウェアバージョン: 9.10

プラットフォーム開発者ガイド

ドキュメントリリース日: 2011年6月 (英語版)
ソフトウェアリリース日: 2011年6月

invent

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。ここでの記載で追加保証を意図するものは一切ありません。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

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。

Java™は、Sun Microsystems, Incの米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。

ドキュメントの更新情報

このマニュアルの表紙には、以下の識別情報が記載されています。

- ソフトウェアバージョンの番号は、ソフトウェアのバージョンを示します。
- ドキュメントリリース日は、ドキュメントが更新されるたびに変更されます。
- ソフトウェアリリース日は、このバージョンのソフトウェアのリリース期日を表します。

更新状況、およびご使用のドキュメントが最新版かどうかは、次のサイトで確認できます。

http://support.openview.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、次のWebサイトから行なうことができます。

http://h20229.www2.hp.com/passport-registration.html (英語サイト)

または、HP Passportのログインページの [**New users - please register**] リンクをクリックします。

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。

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 (英語サイト)

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

目次

第 1 章　概要 11
Server Automationプラットフォームの概要 11
Server Automationプラットフォームのコンポーネント 11
自動化アプリケーション 13
SAランタイム環境: 13
SAプラットフォームリソース 14
SA管理ネットワーク 16
SA管理対象デバイス 16
SAプラットフォームの利点 17
強力なセキュリティ 17
多彩なサービス 17
さまざまなプログラマーから容易に利用可能 18
SAプラットフォームAPIの設計 18
サービス 18
APIのオブジェクト 19
例外 20
イベントキャッシュ 20
検索 20
セキュリティ 21
API ドキュメントとTwister 21
定数のフィールド値 22
サポートされるクライアント 22
第 2 章　SA CLIメソッド 23
SA CLIメソッドの概要 23
メソッドの呼び出し 23
セキュリティ 24
APIとSA CLIメソッドの間のマッピング 24
SA CLIメソッドとUnixコマンドの違い 24
SA CLIメソッドのチュートリアル 25
形式指定子 29
形式指定子の位置 30
デフォルトの形式指定子 31
ID形式指定子の例 31
構造形式指定子の構文 31
構造形式指定子の例 32
ディレクトリ形式指定子の例 34
値の表現 34
OGFS内のSAオブジェクト 34

プリミティブ値 . . . 35
配列 . . . 37
SA CLIメソッドのパラメーターと戻り値 . . . 38
メソッドのコンテキストとselfパラメーター . . . 38
コマンドラインでの引数渡し . . . 38
パラメーターの型の指定 . . . 39
パラメーターとしての複合オブジェクトと配列 . . . 39
オーバーロードされたメソッド . . . 39
戻り値 . . . 40
終了ステータス . . . 40
検索フィルターとSA CLIメソッド . . . 41
検索の構文 . . . 41
検索の例 . . . 41
検索可能な属性と有効な演算子 . . . 43
スクリプトの例 . . . 44
create_custom_field.sh . . . 44
create_device_group.sh . . . 45
create_folder.sh . . . 46
remediate_policy.sh . . . 46
remove_custom_field.sh . . . 48
schedule_audit_task.sh . . . 49
SA CLIメソッドの使用法情報の取得 . . . 49
サービスのリストの表示 . . . 49
APIドキュメントでのサービスの検索 . . . 50
サービスのメソッドのリストの表示 . . . 50
メソッドのパラメーターのリストの表示 . . . 50
値オブジェクトに関する情報の取得 . . . 50
属性が変更可能かどうかの判定 . . . 51
属性がフィルタークエリで使用可能かどうかの判定 . . . 51
第 3 章　PytwistによるPython APIアクセス . . . 53
Pytwistの概要 . . . 53
Pytwistのセットアップ . . . 53
Pytwistでサポートされるプラットフォーム . . . 53
Pytwistのアクセス要件 . . . 53
管理対象サーバーへのPytwistのインストール . . . 53
Pytwistの例 . . . 54
get_server_info.py . . . 55
create_folder.py . . . 56
remediate_policy.py . . . 56
Pytwistの詳細 . . . 59
認証モード . . . 59
TwistServerメソッドの構文 . . . 59
エラー処理 . . . 59
Javaパッケージ名およびデータ型のPytwistへのマッピング . . . 60

第 4 章　自動化プラットフォーム拡張 (APX) の作成 61
APXの作成 61
Program APX 63
Web APX 63
APXユーザーの役割 64
APXのアクセス権 64
アクセス権のエスカレーション 65
APXの構造 66
ファイル構造 66
OGFS統合 66
APXインタフェース - APX拡張のカテゴリの定義 67
RightClickToRunインタフェース 69
インタフェースAPIの使用 69
apxtoolコマンド 70
apxtoolの構文 70
短いコマンドオプションと長いコマンドオプションの使用 70
新しいAPXの作成 - apxtool new 71
APXの削除 - apxtool delete 72
SAからのAPXのエクスポート - apxtool export 73
APXのSAへのインポート - apxtool import 74
APX情報のクエリ - apxtool query 75
APXの現在のバージョンの設定 - apxtool setcurrent 77
エラー処理 78
APXファイル 79
APX構成ファイル - apx.cfg 79
APXアクセス権エスカレーション構成ファイル - apx.perm 80
APXの進行状況の表示 81
apxprogressコマンド 81
apxprogressを使用するサンプルシェルスクリプト 82
APXの進行状況の表示 83
チュートリアル: WebアプリケーションAPXの作成 83
チュートリアルの前提条件 83
1. アクセス権の設定とチュートリアルフォルダーの作成 84
2. 新規Webアプリケーションの作成 84
3. 新規WebアプリケーションのSAへのインポート 86
4. 新規Webアプリケーションの実行 86
5. Webアプリケーションの変更 87
6. 変更したWebアプリケーションの実行 88
チュートリアル: Program APXの作成 89
チュートリアルの前提条件 89
1. アクセス権の設定とチュートリアルフォルダーの作成 89
2. 新規Program APXの作成 90
3. 新規APXのSAへのインポート 92
4. 新規APXの実行 92
5. APXの変更 92
6. 変更したAPXの実行 93

7. TwisterインタフェースへのAPXの進行状況の表示 . . . 93
第 5 章　エージェントツール . . . 97
エージェントツールの概要 . . . 97
インストール要件 . . . 98
オペレーティングシステムのサポート . . . 98
セキュリティ、アクセス制御、認証 . . . 98
その他の要件 . . . 98
インストール: . . . 98
エージェントツールの手動インストール . . . 99
エージェントのインストール時のエージェントツールのインストール . . . 99
エージェントツールのアップグレード . . . 99
エージェントツールのスクリプト . . . 100
使用法 . . . 100
エージェントツールのスクリプトの例 . . . 102
Unix/Linux . . . 102
Windows . . . 102
第 6 章　Microsoft Windows PowerShell/SA統合 . . . 103
Microsoft Windows PowerShellの概要 . . . 103
Windows PowerShellとSAとの統合 . . . 103
統合PowerShell/SAコマンドレット . . . 104
インストール要件 . . . 104
オペレーティングシステムのサポート . . . 104
インストール: . . . 104
Microsoft PowerShell/SA統合の機能 . . . 105
管理対象サーバーへのリモートアクセス . . . 105
監査とスナップショットのルール . . . 105
DSEスクリプト統合 . . . 105
サンプルセッション . . . 105
シナリオ1 . . . 106
シナリオ2 . . . 109
シナリオ3 . . . 111
シナリオ4 . . . 113
第 7 章　Java RMIクライアント . . . 117
Java RMIクライアントの概要 . . . 117
Java RMIクライアントの設定 . . . 117
Java RMIの例 . . . 118
GetServerInfoの例のコンパイルと実行 . . . 118
第 8 章　Webサービスクライアント . . . 121
Webサービスクライアントの概要 . . . 121
このリリースで提供されているプログラム言語のバインド . . . 121
サービスの場所とWSDLのURL . . . 121
Webサービスクライアントのセキュリティ . . . 122
オーバーロードされた操作 . . . 122
Javaインタフェースのサポート . . . 122

サポートされないデータ型 . . . 122
VOの作成または更新の際のsetDirtyAtrributesの呼び出し . . . 123
SA WebサービスAPI 2.2との互換性 . . . 123
Perl Webサービスクライアント . . . 124
Perlクライアントに必要なソフトウェア . . . 124
Perlデモプログラムの実行 . . . 124
Perlサンプルコード . . . 125
Webサービス用のPerlオブジェクトの構築 . . . 128
C# Webサービスクライアント . . . 131
C#クライアントに必要なソフトウェア . . . 131
C#クライアントスタブの入手方法 . . . 131
C#スタブドキュメントへのアクセス . . . 131
C#デモプログラムのビルド . . . 131
C#デモプログラムの実行 . . . 132
C#サンプルコード . . . 133
C#でのパスワードセキュリティ . . . 135
第 9 章　プラグ可能チェック . . . 137
プラグ可能チェックの概要 . . . 137
プラグ可能チェックの設定 . . . 137
プラグ可能チェックのチュートリアル . . . 137
監査と修復の概要 . . . 144
プラグ可能チェックの作成 . . . 145
プラグ可能チェックのガイドライン . . . 146
プラグ可能チェックの開発プロセス . . . 147
プラグ可能チェック構成 (config.xml) . . . 148
監査 (get) スクリプト . . . 149
修復 (set) スクリプト . . . 150
プラグ可能チェックのその他のコード . . . 151
プラグ可能チェックのZIP化 . . . 151
プラグ可能チェックのインポート . . . 152
監査ポリシーの作成 . . . 152
監査ポリシーの作成 . . . 152
監査ポリシーのエクスポート . . . 153
config.xmlファイルの文書型定義 (DTD) . . . 153
付録 A　検索フィルターの構文 . . . 161
フィルターの文法 . . . 161
使用に関する注 . . . 162
第 B 章　Apache HTTPサーバーおよびPHPのリビルド . . . 163
APX HTTP環境の拡張 . . . 163
PHPのリビルド . . . 163
Apacheのリビルド . . . 164
索引 . . . 167

第 1 章　概要

Server Automationプラットフォームの概要

Server Automationプラットフォームは、SAの統合と拡張を容易にするためのAPIとランタイム環境のセットです。Server Automationプラットフォーム APIは、監査コンプライアンス、Windowsパッチ管理、OSプロビジョニングなどのコアサービスを公開します。ランタイム環境は、Global File System (OGFS) にアクセス可能なGlobal Shellスクリプトを実行します。

Server Automationプラットフォームを使用すると、次の作業を実行できます。

- 新しい自動化アプリケーションの構築やSAの拡張を通じて、ITの生産性向上やITポリシーへの適合を促進できます。
- 既存の監視、トラブルのチケット発行、課金、仮想化テクノロジーといった情報を他のITシステムと交換できます。
- SAモデルリポジトリを使用して、オペレーション、環境、資産といった重要なIT情報を記憶して組織化できます。
- さまざまなアプリケーションやオペレーティングシステムの管理を自動化できます。
- 既存のUnixおよびWindowsスクリプトをSAに組み込み、セキュアな監査対象の環境でスクリプトを実行することができます。

Server Automationプラットフォームのコンポーネント

図1に、Server Automationプラットフォームの主な要素を示します。

図 1　　Server Automationプラットフォームのコンポーネント

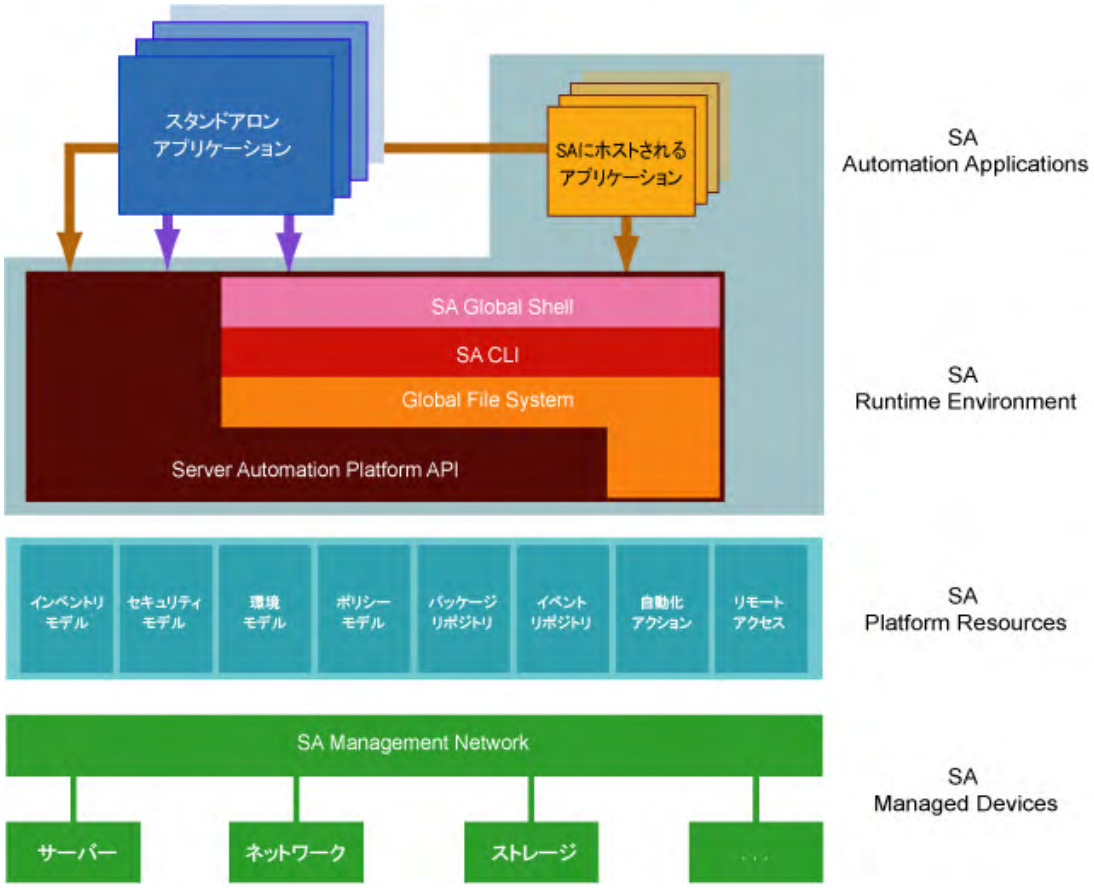


図1に示すように、プラットフォームには次の5つの主要な要素があります。これらの要素については、この後の部分で詳しく説明します。

- **自動化アプリケーション:** ユーザーがプラットフォーム上に作成するアプリケーション。これらのアプリケーションは、SAにホストされたアプリケーション (実行中のSAのコンテキストで動作) またはスタンドアロンのアプリケーション (既存のビジネスおよび管理システムのコンテキストで動作) です。
- **ランタイム環境:** すぐに使用できる強力なランタイムサービスのセットと、それに対応する言語に依存しないプログラミングモデルを提供します。これらは、スクリプト作成者からWeb開発者、熟練した企業Javaプログラマーまで、さまざまなプログラマーから容易に利用できるように設計されています。
- **プラットフォームリソース:** プラットフォームのさまざまなデータオブジェクト、自動化アクション (パッチ適用、プロビジョニング、監査など)、機能 (各管理対象サーバーのランタイム環境へのリモートアクセスなど) を開発者が容易に利用できるようにします。
- **SA管理ネットワーク:** 強力な接続機能、セキュリティ機能、キャッシングテクノロジーのセットで、プラットフォームが場所、IPアドレス空間、利用可能帯域幅などの制約と無関係にデバイスに到達できるようにします。
- **SA管理対象デバイス:** SA管理ネットワークによってプラットフォームに接続される管理対象サーバーおよびネットワークデバイス。

自動化アプリケーション

図1に示すように、自動化アプリケーションはスタックの最上部にあります。これらは、ユーザーがプラットフォーム上に作成するアプリケーションです。

自動化アプリケーションは、SAにホストされたアプリケーション (SAランタイム環境で動作) またはスタンドアロンのアプリケーション (完全に独立したコンテキストで動作) です。スタンドアロンのアプリケーションは、Webサービス呼び出しを通じてプラットフォームにリモートアクセスします。

単純なアプリケーションなら、Unixシェルスクリプトを使ってごく短時間で作成できます。もっと複雑なアプリケーション、たとえば既存のソース管理システムやチケット発行システムとの統合といったものは、もう少し時間がかかり、Python、Microsoft .NET、Javaなどによるコーディングが必要になる可能性があります。いずれの場合でも、プラットフォームは言語に依存しないシステムとして設計されているので、さまざまな開発者が容易に採用できます。

SAランタイム環境:

プラットフォームスタックでその下にあるのがSAランタイム環境です。これは、すぐに使用できる強力なランタイムサービスのセットと、それに対応する言語に依存しないプログラミングモデルを提供します。SAにホストされるアプリケーションは、SAランタイム環境で動作します。

ランタイム環境のコアは、Global ShellとGlobal File Systemの2つのコンポーネントから構成されます。これら2つのコンポーネントの組み合わせにより、すべての管理対象デバイスが、なじみ深いLinux/Unixシェルのファイル/ディレクトリ構造で整理され、アクセスできるようになります。

Global Shell

Global Shellは、Global File System (OGFS) に対するコマンドラインインタフェースです。コマンドラインインタフェースは、ターミナルウィンドウで動作するbashなどのLinuxシェルを通じて使用できます。OGFSは、SAデータモデルと管理対象サーバーの内容 (ファイルなど) を1つの仮想ファイルシステムに統合します。

Global File System

OGFSは、プラットフォームデータモデルのオブジェクト (ファシリティ、カスタマー、デバイスグループなど) と、プラットフォームの管理対象デバイスで使用可能な情報 (管理対象ネットワークデバイスの構成設定や管理対象サーバーのファイルシステムなど) を、ファイルディレクトリとテキストファイルの階層構造で表現します。たとえば、OGFSの/opsw/Customerディレクトリにはカスタマーオブジェクトの詳細が、/opsw/Serverディレクトリには管理対象サーバーに関する情報が記録されています。また、/opsw/Serverディレクトリには、管理対象サーバーの内容 (ファイルシステムやレジストリ) を反映するサブディレクトリも含まれています。

このファイル/ディレクトリ構造を使用することで、シェルスクリプトになじんだ管理者は、サーバーを表すディレクトリを反復処理することで同じ作業を複数のサーバーに対して実行するスクリプトを容易に作成できます。Global File Systemは、スクリプトのロジックを各管理対象サーバーに対してセキュアに配布して実行します。

デバイスの内容にアクセスするには、SAとNetwork Automation (NA) で管理されるすべてのデバイスを表す仮想ファイルシステムであるGlobal File Systemを使用します。エンドユーザーと自動化アプリケーションの両方が、必要なセキュリティ承認を経て、OGFSを通じてリモートサーバーのファイルシステムにアクセスできます。Windowsサーバーでは、管理者はレジストリ、IIメタベース、COM+オブジェクトにもアクセスできます。

SAコマンドラインインタフェース

SAコマンドラインインタフェース (CLI) は、システム管理者やプラットフォーム自動化アプリケーションが、ソフトウェアのプロビジョニング、デバイスへのパッチ適用、監査の実行といった自動化タスクをコマンドラインから起動するために使用できます。高機能の構文により、さまざまなオブジェクトタイプを、CLI呼び出しの入力に使用したり、出力として受け取ったりすることができます。

実際には、CLI自体は、次の項で説明するプラットフォームAPIの上に、プログラムによって生成されます。この方法の利点は、開発者によって新しいAPIがプラットフォームAPIに追加されたときに、それに対応するCLIメソッドが自動的に使用可能になることです。すなわち、製品で新機能が使用可能になってから、対応するCLIメソッドがプラットフォームで使用可能になるまでに、遅れが発生しないということです。

SAプラットフォームAPI

SAプラットフォームAPIは、SAのWin32 APIです。値の取得と設定や、アクションの実行のためのアプリケーションプログラミングインタフェースが定義されています。SAクライアントやSAコマンドラインインタフェース (CLI) といったSAのユーザーインタフェースは、すべてSAプラットフォームAPIの上に構築されています。APIには、Java RMIクライアント向けのライブラリと、SOAPベースのWebサービスクライアント向けのWSDLが含まれています。Webサービスのサポートにより、プログラマーはPerl、C#、Pythonといった一般的な言語でクライアントを作成できます。

SAプラットフォームリソース

SAプラットフォームリソースは、SAランタイム環境の下にあって、開発者がさまざまなオブジェクトやアクションのセットにアクセスし、自分のアプリケーションで再使用したり操作したりするために使用されます。

インベントリモデル

インベントリモデルは、各管理対象デバイスに対してSAが収集するすべての情報 (メーカー、製造者、CPU、オペレーティングシステム、インストール済みソフトウェアなど) を提供します。インベントリ情報は、SA APIを通じて入手できるとともに、Global File Systemのファイル (`attr`サブディレクトリの下) としても表示されます。インベントリモデルには、サーバーやネットワークデバイスなどのオブジェクトが含まれます。

管理者は、インベントリオブジェクトに関連付けられたデータを拡張できます。たとえば、ユーザーがデバイスの画像や、リース有効期限や、デバイスが接続されているUPSのIDなどを保存したい場合、このような属性を各デバイスレコードに容易に追加できます。ユーザーは、これらの属性の追加、削除、操作を、標準の属性の場合と同様に行うことができます。

セキュリティモデル

セキュリティモデルを使うことにより、開発者は、SAが備える認証と承認のセキュリティシステムを利用できます。

プラットフォームのすべてのクライアント (管理アプリケーション、スクリプト、SAのエンドユーザーインタフェース) は、同じセキュリティフレームワークで管理されます。

ユーザーの役割を作成し、アクセス権を付与するのは、開発者でなくセキュリティ管理者です。開発者は、自分のアプリケーションのコンテキスト内で、これらすべてのユーザーの役割とアクセス権を再使用できます。たとえば、ネットワーク管理者がシェルスクリプトを作成し、他のネットワーク管理者と共有する場合に、共有相手のネットワーク管理者は、自分が管理する権限を持つネットワークデバイス以外に対してそのスクリプトを実行できないことが保証されます。

承認メカニズムは、いくつかのレベルでアクセスを制御します。レベルとしては、ユーザーが実行できるタスク、タスクからアクセスされるサーバーとネットワークデバイス、SAオブジェクト (ソフトウェアポリシーなど) があります。

環境モデル

環境モデルは、デバイスが存在するビジネスコンテキスト全体を定義します。一般的に、デバイスは1つまたは複数のカスタマーに属し、特定のファシリティに存在し、1つまたは複数のグループに属します。プラットフォームは、これらのオブジェクト (カスタマー、ファシリティ、デバイスグループなど) をアプリケーション開発者から使用可能にします。

インベントリオブジェクトと同様、環境オブジェクトも容易に拡張できます。これにより、たとえば、特定のデータセンターで用いられるSNMPトラップレシーバー、特定のファシリティでのみ使用可能なプリンター、特定のビジネスユニットのみで使用されるApache構成といった属性の定義が容易になります。

ポリシーモデル

ポリシーモデルを使用することで、開発者はSAで定義されているすべてのベストプラクティスにアクセスできます。ポリシーは、サーバーまたはネットワークデバイスの適切な状態を記述します。たとえば、パッチポリシーはサーバー上に存在すべきパッチを、ソフトウェアポリシーはサーバー上に存在すべきソフトウェアを記述します。

これらのポリシーは当該分野の専門家が定義し、承認されたシステム管理者は、これらのポリシーを使用してデバイスを監査することによって、デバイス上に実際に存在するものが存在すべきものと異なっているかどうかを判定できます。プログラマーは、すべてのポリシーのライブラリを自分のアプリケーションから利用できます。

ソフトウェアポリシーはフォルダーによって整理され、これによってセキュリティ境界が定義されます。すなわち、アプリケーションは、ユーザーアクセス権に基づいてアクセスが許可されているソフトウェアポリシーだけにアクセスできます。

パッケージリポジトリ

パッケージリポジトリを使用することで、開発者はSAに記憶されているすべてのソフトウェアとパッチにアクセスできます。これには、オペレーティングシステムビルド、オペレーティングシステムパッチ、ミドルウェア、エージェント、およびユーザーがSAにアップロードしたすべてのソフトウェアが含まれます。

イベントリポジトリ

イベントリポジトリは、アクションが (ユーザーインタフェースを通じて、またはプログラムからプラットフォームによって) 実行されたときにSAが生成するデジタル署名付き監査証跡を格納します。他のプラットフォームオブジェクトと同様、これらのイベントはプログラムから取得できます。

自動化アクション

自動化アクションを使用することで、開発者は、SA が管理対象デバイスに対して実行できるすべてのアクションをプログラムから起動できます。これには、監査の実行、ソフトウェアのプロビジョニング、最新のOSパッチの適用などが含まれます。

プラットフォームは、エンドユーザーがSAクライアントで使用できるのと同じ機能へのアクセスを提供します。これには、パッチのインストール、オペレーティングシステムのプロビジョニング、ソフトウェアポリシーのインストールと削除などの作業が含まれます。実際には、SA クライアントは SA ランタイム環境を通じてプログラムから利用できるのと同じAPIを使用しています。

リモートアクセス

リモートアクセスを使用することで、開発者は、管理対象デバイスのファイルシステム (サーバーの場合) および実行環境 (すべてのデバイスの場合) にプログラムからアクセスできます。開発者は、ファイルまたは特定のソフトウェアパッケージの存在をチェックしたり、オペレーティングシステムコマンドを実行してディスクの使用状況をチェックしたり、システムスクリプトを実行して日常のメンテナンス作業を実行したりするアプリケーションを容易に作成できます。

SA管理ネットワーク

管理ネットワークは、開発者が管理対象の任意のデバイスにセキュアにアクセスできるようにするための強力なテクノロジーの組み合わせです。管理ネットワークが提供する主なサービスは次のとおりです。

- **接続**: プラットフォーム (およびそれを使用する自動化アプリケーション) から任意の管理対象デバイスに到達できるようにします。
- **セキュリティ** : SSL/TLSベースの暗号化、認証、メッセージの完全性を含みます。
- **アドレス空間の仮想化**: 重なり合う複数のIPアドレス空間内でプラットフォームがサーバーを発見できるようにします。複雑な企業ネットワークのほとんどには、複数のプライベートIPアドレス空間が存在します。
- **可用性**: システムアーキテクチャーで、管理対象デバイスへの冗長経路を定義して、どれかのネットワーク経路に障害が起きてもデバイスに到達できるようにすることができます。
- **キャッシング**: サーバーが遠くのサーバーでなく近くのサーバーからソフトウェアやパッチをダウンロードできるようにすることで、時間とネットワーク接続のコストを節約します。
- **帯域幅スロットリング**: システムアーキテクチャーで、SAとSAアプリケーションがネットワーク経由で特定のデバイスと通信するときに消費できる帯域幅を決定できるようにします。
- **最小コストルーティング**: システムデザイナーが、特定のデバイスに到達するために使用する経路を決定するルールを設定することで、ネットワーク接続のコストを最小化できるようにします。

SA管理対象デバイス

プラットフォームスタックの最下部にあるのは、実際の管理対象デバイスです。プラットフォームは、65種類以上のサーバーOSバージョンと、35社以上のネットワークデバイスベンダーを管理し、標準でサポートされるデバイスのモデル/バージョン数は数千に及びます。

サポートされるデバイスのリストは常に更新されています。このリストは、プラットフォーム開発者やスクリプト作成者にとって直接的な利益があります。使い慣れた同じプラットフォームプログラミング環境を使いながら、自動化アプリケーションから到達できる管理対象デバイスを常に増やし続けることができるからです。

SAプラットフォームの利点

SAプラットフォームの主な利点を次に示します。

強力なセキュリティ

次のような包括的なセキュリティメカニズムがプラットフォームから提供されるので、アプリケーションで独自にセキュリティを実現する手間がかかりません。

- **セキュアな通信チャネル**: 自動化アプリケーションから管理対象デバイスへのエンドツーエンドの通信は暗号化され、認証されます。
- **役割ベースのアクセス制御**: プラットフォームはSAに備わっている役割ベースのアクセス制御に従うので、開発者がアプリケーションを共有する場合、管理者がアクセス権を付与されているデバイスに対してだけそのアプリケーションが実行可能であることが保証されます。
- **デジタル署名付き監査証跡**: 自動化アプリケーションが実行された後で、プラットフォームは、アプリケーションの実行者、実行時刻、対象デバイスを記録したデジタル署名付き監査証跡を生成します。
- **包括的な到達範囲**: プラットフォームはすべてのデバイスに対する包括的な到達範囲を実現するので、システム管理者や開発者は、デバイスへの到達方法について悩む必要はありません。
- **業界最大の対象プラットフォーム範囲**: サポートされるデバイスには、65種類以上のサーバーOSバージョンと、1,000種類以上のネットワークデバイスが含まれます。
- **任意の物理的場所**: デバイスは、大規模データセンター、小売店、サテライトオフィスなど、世界中のどこに存在していてもかまいません。
- **任意のIPアドレス空間**: プラットフォームは重なり合う複数のIPアドレス空間をサポートするので、デバイスはどのIPアドレス空間に属していてもかまいません。
- **DMZ**: デバイスがDMZなどのアクセス困難なネットワーク空間に存在する場合でも、開発者やシステム管理者は、デバイスへの到達方法の詳細 (要塞ホスト経由など) について悩む必要はありません。

多彩なサービス

プラットフォームからは、元になる自動化システムのほぼすべての関連データおよびアクションを利用できます。

- **多彩な標準データ**: 開発者は、プラットフォーム自体から生成されるデータ (デバイスインベントリデータやファシリティ情報など) と、プラットフォームを使用するユーザーによって生成されたデータ (デバイスグループカスタマー、ベストプラクティスポリシー、アップロードされたソフトウェア、パッチ、スクリプトなど) を含む多彩なデータに容易にアクセスできます。開発者は、このデータを読み書きするアプリケーションを容易に作成できます。
- **拡張可能なデータストア**: 開発者は、ネイティブプラットフォームオブジェクトを容易に拡張して、独自のデータを追加することができます。デバイスインベントリモデルを拡張することにより、プラットフォームでネイティブに検出されない属性を含めることができます。カスタマーオブジェクトやファシリティオブジェクトを拡張して、そのカスタマーに関連するデバイスのプロビジョニングや監査の参考となる属性を含めることができます。
- **自動化タスク**: プラットフォームは、元になる自動化システムのほぼすべての機能を開発者に公開しています。これには、パッチ適用、プロビジョニング、監査などが含まれます。これにより、複数のシステムにまたがる複雑なワークフローを作成する場合でも、これらのアクションを自動化アプリケーションのコンテキストから簡単に呼び出すことができます。

さまざまなプログラマーから容易に利用可能

プラットフォームは、Unixシェルスクリプトや Visual Basicスクリプトの作成者、PerlやPythonのプログラマー、企業の.NETまたはJavaプログラマーなど、広範囲の開発者に便利なように設計されています。プラットフォームのランタイムサービスレイヤーでは、ほとんどのプラットフォームオブジェクトがファイル/ディレクトリ構造で利用でき、ほとんどのプラットフォームサービスがコマンドラインインタフェース (SA CLI) を通じて使用できます。このため、シェルスクリプトの使用に慣れたシステム管理者は、新しいプログラミング言語やツールを習得しなくても、すぐにプラットフォームを利用できます。使い慣れたテキストエディターとUnixシェルを使って、すぐにスクリプトを開発できます。

もっと複雑なアプリケーションや既存のシステムとの統合が必要な場合、システムプログラマーは、Webサービスバインディングを備えた任意のプログラミングツールや言語を使用できます。

SAプラットフォームAPIの設計

プラットフォームAPIは、Javaインタフェースで定義され、Javaパッケージに組織化されています。さまざまなクライアント言語やリモートアクセスプロトコルをサポートするため、API は関数方式の値呼び出しモデルを採用しています。

サービス

プラットフォーム API では、関連する関数のセットが1つのサービスにまとめられています。各サービスは、名前の末尾が `Service` である Java インタフェースによって指定されます。例としては、`ServerService`、`FolderService`、`JobService`などがあります。

サービスは、APIへのエントリポイントです。APIにアクセスするには、クライアントはサーバーインタフェースで定義されたメソッドを呼び出します。たとえば、管理対象サーバー上にインストールされているソフトウェアのリストを取得するには、クライアントは`ServerService`インタフェースの`getInstalledSoftware`メソッドを呼び出します。他の`ServerService`メソッドの例としては、`checkDuplex`、`setPrimaryInterface`、`changeCustomer`があります。

SAプラットフォームAPIには70以上のサービスが含まれているので、ここではすべてを記述できません。表1に示すのは、最初に試してみることをお勧めするいくつかのサービスです。すべてのサービスの一覧については、APIドキュメントとTwister (21ページ) に示すURLにブラウザーでアクセスしてください。

表 1　　SAAPIのサービスの一部

サービス名	サービスで提供される操作の一部
`AuditTaskService`	監査タスクの作成、取得、実行。
`ConfigurationService`	アプリケーション構成の作成、アプリケーション構成を使用するソフトウェアポリシーの取得。
`DeviceGroupService`	デバイスグループの作成、グループへのデバイスの割り当て、グループのメンバーの取得、動的ルールの設定。
`EventCacheService`	値オブジェクトのクライアント側キャッシュの更新などのアクションのトリガー。詳細についてはイベントキャッシュ (20ページ) を参照してください。

表 1　SAAPIのサービスの一部 (続き)

サービス名	サービスで提供される操作の一部
FolderService	フォルダーの作成、フォルダーの子の取得、フォルダーのカスタマーの設定、フォルダーの移動。
InstallProfileService	OSインストールプロファイルの作成、取得、更新。
JobService	ジョブの進行状況と結果の取得、ジョブのキャンセル、ジョブスケジュールの更新。
NasConnectionService	NAサーバーのホスト名の取得、NAサーバーに対するコマンドの実行。
NetworkDeviceService	指定した検索フィルターに基づくファミリー、名前、モデル、タイプなどの情報の取得。
SequenceService	サーバーにオペレーティングシステムをインストールするためのOSシーケンスの作成、取得、実行。
ServerService	サーバーに関する情報の取得、サーバー上でのポリシーの調整 (修復) (ソフトウェアのインストール)、カスタムフィールドおよび属性の取得と設定、OS シーケンスの実行 (OSのインストール)。
SoftwarePolicyService	ソフトウェアポリシーの作成、サーバーへのポリシーの割り当て、ポリシーの内容の取得、サーバーに対するポリシーの修復 (調整)。
SolPatchService	Solaris パッチのインストールとアンインストール、ポリシーオーバーライドの追加。
VirtualColumnService	カスタムフィールドおよびカスタム属性の管理。
WindowsPatchService	Windowsパッチのインストールとアンインストール、ポリシーオーバーライドの追加。

APIのオブジェクト

SAプラットフォームAPIは関数に基づいていますが、クライアントがオブジェクト指向のライブラリを作成できるように設計されています。SA のデータモデルには、サーバー、フォルダー、カスタマーなどのオブジェクトが含まれます。これらは永続的なオブジェクトです。すなわち、モデルリポジトリに格納されます。APIには、これらのオブジェクトに対応する次のアイテムがあります。

- オブジェクトの動作を定義するサービス。たとえば、ServerServiceのメソッドは、管理対象サーバーオブジェクトの動作を指定します。
- 永続的オブジェクトのインスタンスを表すオブジェクト (ID) 参照。たとえば、ServerRefは管理対象サーバーを一意に識別する参照です。ServerServiceのほとんどのメソッドの第1引数は、そのメソッドの操作対象の管理対象サーバーを識別するServerRefです。ServerRefのId属性は、モデルリポジトリに格納されているサーバーオブジェクトの主キーです。
- 永続的オブジェクトのデータメンバー (属性、フィールド) を表す1つ以上の値オブジェクト (VO)。たとえば、ServerVOにはagentVersionやloopbackIPなどの属性が含まれます。ServerHardwareVOの属性には、manufacturer、model、assetTagが含まれます。ほとんどの属性は、クライアントアプリケーションからは変更できません。属性が変更可能な場合、setterメソッドのAPIドキュメントに「フィールドはクライアントから設定可能」と記されています。

パフォーマンス上の理由で、永続的オブジェクトの更新操作は大きい単位で行われます。たとえば、ServerServiceのupdateメソッドは、個々の属性でなくServerVO全体を引数に取ります。

例外

SA固有のAPI例外は、すべて次のいずれかの例外から派生します。

- OpswareException - アプリケーションレベルのエラーが起きたときに発生します。たとえば、エンドユーザーが無効な値をメソッドに渡したときなどです。クライアントアプリケーションは通常、このタイプの例外からは回復できます。OpswareExceptionから派生する例外の例としては、NotFoundException、NotInFolderException、JobNotScheduledExceptionがあります。
- OpswareSystemException - SA内部でエラーが起きたときに発生します。通常、クライアントアプリケーションが実行可能になるには、SA管理者が問題を解決する必要があります。

次の例外はセキュリティに関連しています。

- AuthenticationException - 無効なSAユーザー名またはパスワードが指定されたときに発生します。
- AuthorizationException - ユーザーに操作の実行またはオブジェクトへのアクセスの権限がないときに発生します。アクセス権の詳細については、『SA 管理ガイド』を参照してください。

イベントキャッシュ

一部のクライアントアプリケーションは、SAオブジェクトのローカルコピーを保持する必要があります。クライアントからEventCacheServiceを通じてアクセスされるキャッシュには、SAオブジェクトへの最近の変更を記述するイベントが記録されています。クライアントは、定期的にキャッシュをポーリングすることで、オブジェクトの作成、更新、削除を検出できます。イベントは、設定された一定の時間だけキャッシュに保持されます。デフォルトでは、最近2時間分のイベントが保持されます。保持時間を変更するには、『SA管理ガイド』の説明に従って、Webサービスデータアクセスエンジンの構成ファイルを編集します。

検索

SAプラットフォームAPIの検索メカニズムは、値オブジェクトの属性 (フィールド) に基づいてオブジェクト参照を取得します。たとえば、getServerRefsメソッドはServerVO値オブジェクトの属性によって検索を行います。getServerRefsメソッドのシグネチャは次のとおりです。

```
public ServerRef[] getServerRefs(Filter filter)...
```

get*Refsメソッドは、filterパラメーターで検索基準を指定するオブジェクトを受け取ります。単純な式によるfilterパラメーターの構文を次に示します。

値オブジェクト.属性 演算子 値

(この構文は単純化されています。完全な定義については、フィルターの文法 (161 ページ) を参照してください)。

次の例は、getServerRefsメソッドのfilterパラメーターを示します。

```
ServerVO.hostName = "d04.example.com"
ServerVO.model BEGINS_WITH "POWER"
ServerVO.use IN "UNKNOWN" "PRODUCTION"
```

次のような複合式も使用できます。

```
(ServerVO.model BEGINS_WITH "POWER") AND (ServerVO.use = "UNKNOWN")
```

値オブジェクトの属性の中には、filter パラメーターに指定できないものがあります。たとえば、ServerVO.stateはfilterパラメーターで使用できますが、ServerVO.OsFlavorは使用できません。使用可能な属性を知るには、API ドキュメントの値オブジェクトの項目に「フィールドはフィルタークエリに使用可能」というコメントがあるかどうかを確認します。

セキュリティ

SAプラットフォームのユーザーは、SA自動化プラットフォームAPIのメソッドを呼び出すために認証され、承認される必要があります。SAに接続する際に、クライアントはSAユーザー名とパスワードを送信します(認証)。メソッドを呼び出すためには、SAユーザーは必要なアクセス権を持つグループに属する必要があります (承認)。アクセス権は、ユーザーが実行できる操作のタイプを制限するだけでなく、操作で使用されるサーバーおよびネットワークデバイスへのアクセスも制限します。

アプリケーションクライアントがプラットフォーム上で動作するためには、SA管理者が必要なユーザーおよびアクセス権をコマンドセンターで指定する必要があります。詳細については、『SA 管理ガイド』の「ユーザーとグループの設定」の章を参照してください。セキュリティ関連の例外の詳細については、例外 (20ページ) を参照してください。

クライアントとSAの間の通信は暗号化されます。Webサービスクライアントの場合、要求と応答のSOAPメッセージ (操作呼び出しを実装するもの) はSSL over HTTP (HTTPS) で暗号化されます。

APIドキュメントとTwister

SAには、SAプラットフォームAPIに関して記述しているAPIドキュメント (Javadocs) が付属しています。APIドキュメントにアクセスするには、ブラウザーに次のURLを指定します。

```
https://<SAコアホスト>/twister
```

<SAコアホスト>は、コマンドセンターコンポーネントを実行しているSAコアサーバーのIPアドレスまたはホスト名です。

Twisterは、ブラウザー内部からAPIメソッドを一度に1つずつ呼び出すためのプログラムです。たとえば、ServerService.getServerVOメソッドを呼び出すには、次の手順を実行します。

1. ブラウザーでAPIドキュメントを開きます。
2. [すべてのクラス] ペインでcom.opsware.serverを選択します。
3. com.opsware.serverペインでServerServiceを選択します。
4. メインペインで、下にスクロールしてgetServerVOメソッドを表示します。
5. getServerVOメソッドに対して [**試行**] をクリックします。
6. SAユーザー名とパスワードを入力します。
7. ServerService.getServerVOのTwisterペインで、[oid] フィールドに管理対象サーバーのIDを入力します。

8 [**実行**] をクリックします。Twisterペインに、返されたServerVOオブジェクトの属性が表示されます。

定数のフィールド値

API のいくつかの値オブジェクト (VO) では、フィールドの値が定数で定義されています。たとえば、JobInfoVOのstatusフィールドは、STATUS_ACTIVEやSTATUS_PENDINGなどの定数で定義される値を取ります。APIでは定数がJavaのstatic finalフィールドとして指定されていますが、APIから生成されたWSDLでは定数が定義されていません。定数の定義を見るには、API ドキュメントで [定数のフィールド値] ページを開きます。

```
https://<SAコアホスト>/twister/docs/constant-values.html
```

たとえば、[定数のフィールド値] ページではSTATUS_ACTIVEが整数1と定義されています。

サポートされるクライアント

SAプラットフォームは、シェルスクリプトを作成するシステム管理者から、最新のツールやテクノロジーに精通した.NETやJavaのプログラマーまで、さまざまなスキルレベルのプログラマーをサポートします。サポートされるすべてのクライアントは同じメソッドのセットを呼び出します。メソッドはSAプラットフォームのサービスによって分類されています。開発者は、SAプラットフォームAPIのメソッドを呼び出す次のタイプのクライアントを作成できます。

- **SA コマンドラインインタフェース (CLI)**: Global Shellセッションから起動されるシェルスクリプトは、OGFSの実行可能プログラムであるCLIメソッドを呼び出すことで、SAプラットフォームAPIにアクセスできます。各CLIメソッドは、APIの1つのメソッドに対応します。
- **Webサービス**: これらのクライアントは、SOAP over HTTPSを使用して、SAに要求を送信し、応答を受信します。Webサービスの操作 (WSDLで定義) は、APIのメソッドに対応します。開発者は、PerlやC#といった一般的な言語でWebサービスクライアントを作成できます。
- **Java RMI**: これらのクライアントは、他のJava仮想マシンからリモートJavaオブジェクトを呼び出します。
- **Pytwist**: これらのPythonプログラムは、SAコアサーバーまたは管理対象サーバー上で動作します。

Web サービスクライアントと Java RMI クライアントは、SA コアサーバーまたは管理対象サーバーと異なるサーバー上で動作することができます。CLIメソッドは、OGFSがインストールされているコアサーバー上のGlobal Shellセッションで実行されます。

第 2 章　SA CLIメソッド

SA CLIメソッドの概要

エンドユーザーは、SAクライアントを通じてSAにアクセスします。場合によっては、上級ユーザーが、複数のサーバーに対するバルク操作や繰り返し作業を実行するために、コマンドライン環境でSAにアクセスすることが必要になります。SAのコマンドライン環境は、Global Shell (OGSH)、Global File System (OGFS)、SAコマンドラインインタフェース (CLI) メソッドから構成されます。

SAの操作をコマンドラインから実行するには、OGSHセッション内部からSA CLIメソッドを呼び出します。SA CLIメソッドは、SA APIのメソッドに対応する、OGFS内の実行可能ファイルです。SA CLIメソッドを実行すると、対応するAPIメソッドが呼び出されます。

この章を理解するには、OGSHとOGFSに関する知識が必要です。詳細については、『SAユーザーガイド: Server Automation』の「OGSH」を参照してください。

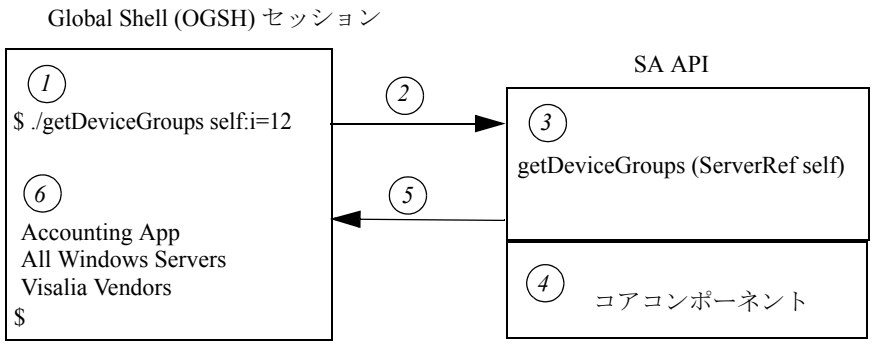
`ouploadおよびodownload`コマンドの詳細については、『SAユーザーガイド: Server Automation』の「OCLI 1.0」を参照してください。

メソッドの呼び出し

図2に示すように、OGSHセッションでSA CLIメソッドを呼び出すと、次の処理が行われます。

1. 入力されたコマンドとパラメーターをOGSHが解析して、APIメソッドを判定します。
2. OGSHが対応するAPIメソッドを呼び出します。
3. この操作を実行するアクセス権がユーザーにあるかどうかが検証されます。その後、SAが操作を実行します。
4. APIメソッドが結果をSA CLIメソッドに返します。
5. SA CLIメソッドが戻り値をOGSHセッションの`stdout`に書き出します。例外が発生した場合、SA CLIメソッドは0以外のステータスを返します。

図 2　　SA CLIメソッドの呼び出しの概要



セキュリティ

SA CLIメソッドは、SAクライアントと同じ認証と承認のメカニズムを使用します。OGSHセッションを開始すると、SAはSAユーザーを認証します。SA CLIメソッドを実行すると、承認が実行されます。SA CLIメソッドを正常に実行するには、必要なアクセス権を持つグループにSAユーザーが属する必要があります。セキュリティの詳細については、『SA 管理ガイド』を参照してください。

APIとSA CLIメソッドの間のマッピング

OGFSは、SAオブジェクトをディレクトリ構造で、オブジェクト属性をテキストファイルで、APIメソッドを実行可能ファイルで表現しています。これらの実行可能ファイルがSA CLIメソッドです。各SA CLIメソッドは、1つのAPIメソッドに対応します。メソッド名、パラメーター、戻り値は、どちらのタイプのメソッドでも同じです。

たとえば、`setCustomer` APIメソッドは次のJavaシグネチャを持ちます。

```
public void setCustomer(ServerRef self,
                        CustomerRef customer)...
```

OGFSの対応するSA CLIメソッドは次の構文を取ります。

```
setCustomer self:i=サーバー ID customer:i=カスタマー ID
```

パラメーター名 (`self` と `customer`) がどちらの言語でも同じであることに注意してください(`:i` 記法はフォーマット指定子と呼ばれます。これについてはこの章の後の方で説明します)。この例では、戻り値型はvoidなので、SA CLIメソッドは結果を`stdout`に書き出しません。SA CLIメソッドがオブジェクトを表す文字列を返す方法の詳細については、戻り値 (40ページ) を参照してください。

SA CLIメソッドとUnixコマンドの違い

OGSH ではUnix コマンドと SA CLI メソッドの両方を実行できますが、SA CLI メソッドはいくつかの点で異なっています。

- 多くのUnixコマンドと異なり、SA CLIメソッドは`stdin`からデータを読み取りません。したがって、パイプ (|) で接続されたコマンドのグループ内にSA CLIメソッドを挿入することはできません(ただし、SA CLIメソッドは`stdout`への出力は行います)。
- ほとんどのUnixコマンドはパラメーターをフラグと値で受け取ります (例、`ls -l /usr`)。SA CLIメソッドでは、コマンドラインパラメーターは名前と値のペアを等号で結んだものです。

- Unixコマンドはテキストベースです。これらのコマンドは、データを文字列で受け取って返します。これに対して、SA CLIメソッドは複合オブジェクトを受け取って返すことができます。
- SA CLIメソッドでは、パラメーターと戻り値の形式を指定できます。Unixコマンドにはこれに相当する機能はありません。

SA CLIメソッドのチュートリアル

このチュートリアルでは、実際の環境で実行できる例を使用して、SA CLIメソッドの概要を説明します。このチュートリアルを終了すると、SA CLIメソッドを実行したり、SAオブジェクトの`self`ファイルを調べたり、複数のサーバーに対してSA CLIメソッドを呼び出すスクリプトを作成したりできるようになります。

チュートリアルを開始する前に、次のことが必要です。

- SAクライアントにログオンできること。
- SAユーザーが少なくとも1つの管理対象サーバーで読み取り/書き込みアクセス権を持つこと。アクセス権は通常はセキュリティ管理者によって割り当てられます。詳細については『SA 管理ガイド』を参照してください。
- SAユーザーが同じ管理対象サーバーですべてのOGSHアクセス権を持つこと。これらのアクセス権の詳細については、『SAユーザーガイド: Server Automation』の「aaaユーティリティ」を参照してください。
- OGSHとOGFSについての知識があること。これらの機能を初めて使用する場合は、このチュートリアルを実行する前に、『SAユーザーガイド: Server Automation』の「Global Shell」を参照してください。

このチュートリアルのコマンド例は、`abc.example.com`という名前のWindowsサーバーを対象としています。このサーバーは、「All Windows Servers」という名前のサーバーグループに属しています。これらのコマンドを試すときには、`abc.example.com`を、自分がアクセス権を持つ管理対象サーバーのホスト名に置き換えてください。

1 OGSHセッションを開きます。

Global ShellセッションはSAクライアントから開くことができます。[**アクション**] メニューで、[**Global Shell**] を選択します。OGSHセッションは、デスクトップ上で動作しているターミナルクライアントから開くこともできます。詳細については、『SAユーザーガイド: Server Automation』の「Global Shellセッションを開く」を参照してください。

2 サーバーのSA CLIメソッドのリストを表示します。

サーバーの`method`サブディレクトリに、そのサーバーで実行可能なメソッドに対応する実行可能ファイルが含まれます。次の例は、`abc.example.com`サーバーのSA CLIメソッドのリストを表示します。

```
$ cd /opsw/Server/@/abc.example.com
$ ls -1
addDeviceGroups
attachPolicies
attachVirtualColumn
checkDuplex
clearCustAttrs
...
```

これらのメソッドにはインスタンスコンテキストがあり、特定のサーバーインスタンス (この例では`abc.example.com`) で動作します。サーバーインスタンスは、メソッドのパスから知ることができます。静的コンテキストを持つメソッドについては手順5で説明します。

3 SA CLIメソッドをパラメーターなしで実行します。

abc.example.comが属するパブリックサーバーグループを表示するには、getDeviceGroupsメソッドを呼び出します。

```
$ cd /opsw/Server/@/abc.example.com
$ ./getDeviceGroups
Accounting App
All Windows Servers
Visalia Vendors
```

4 メソッドをパラメーター付きで呼び出します。

メソッドのコマンドラインパラメーターは、スペース文字で区切られた名前と値のペアで示されます。次のsetCustomerの呼び出しでは、パラメーター名はcustomerで、値は20039です。パラメーター名の末尾の:iは、後で説明するID形式指定子です。

次のメソッド呼び出しは、abc.example.comサーバーのカスタマーをOpswareからC39に変更します。カスタマー C39のIDは20039です。

```
$ cd /opsw/Server/@/abc.example.com
$ cat attr/customer ; echo
Opsware
$ method/setCustomer customer:i=20039
$ cat attr/customer ; echo
C39
```

5 管理対象サーバーの静的コンテキストメソッドのリストを表示します。

静的コンテキストメソッドは、/opsw/apiディレクトリにあります。これらのメソッドは、オブジェクトの特定のインスタンスに限定されません。

サーバーの静的メソッドのリストを表示するには、次のコマンドを入力します。

```
$ cd /opsw/api/com/opsware/server/ServerService/method
$ ls
```

リストに表示されるメソッドは、手順2で表示されるものと同じです。

6 メソッドをselfパラメーター付きで呼び出します。

このステップでは、getDeviceGroupsを静的コンテキストメソッドとして呼び出します。手順3に示したインスタンスコンテキストメソッドと異なり、静的コンテキストメソッドにはサーバーインスタンスを識別するためのselfパラメーターが必要です。

たとえば、abc.example.comサーバーのIDが530039だとします。このサーバーのグループのリストを表示するには、次のコマンドを入力します。

```
$ cd /opsw/api/com/opsware/server/ServerService/method
$ ./getDeviceGroups self:i=530039
Accounting App
All Windows Servers
Visalia Vendors
```

このgetDeviceGroupsの呼び出しを、インスタンスコンテキストを示す手順3の呼び出しと比較してみてください。どちらの呼び出しも、APIの対応する同じメソッドを呼び出し、同じ結果を返します。

7 サーバーのselfファイルを調べます。

SAでは、各管理対象サーバーは1つのオブジェクトです。ただし、OGFSはファイルシステムであり、オブジェクトモデルではありません。selfファイルは、SAオブジェクトのさまざまな表現にアクセスするために使用できます。表現には、ID、名前、構造があります。

サーバーのデフォルトの表現は名前です。たとえば、サーバーの名前を表示するには、次のコマンドを入力します。

```
$ cd /opsw/Server/@/abc.example.com
$ cat self ; echo
abc.example.com
```

サーバーのIDがわかっている場合、次の例のように、selfファイルから名前を取得できます。

```
$ cat /opsw/.Server.ID/530039/self ; echo
abc.example.com
```

8 selfファイルのID形式指定子を指定します。

selfファイルの特定の表現を選択するには、ピリオドの後にファイル名、その後に形式指定子を入力します。たとえば、次のcatコマンドには、サーバー IDを表示する形式指定子 (:i) が含まれています。

```
$ cd /opsw/Server/@/abc.example.com
$ cat .self:i ; echo
com.opsware.server.ServerRef:530039
```

この出力は、abc.example.comのIDが530039であることを示します。
com.opsware.server.ServerRefは、SA APIの対応するオブジェクトであるサーバー参照のクラス名です。

先頭のピリオドは、ファイルとメソッドの戻り値に形式指定子を付ける場合には必要ですが、メソッドのパラメーターには使用しません。

9 構造形式指定子を示します。

構造形式指定子 (:s) は、複合オブジェクトの属性を示します。属性は名前と値のペアとして表示され、すべてのペアは中括弧に囲まれます。構造形式は、コマンドラインで複合オブジェクトをメソッドパラメーターに指定する場合に使用されます(メソッド呼び出しの例については、パラメーターとしての複合オブジェクトと配列 (39ページ) を参照してください)。

次の例は、abc.example.comを構造形式で表示します。

```
$ cd /opsw/Server/@/abc.example.com
$ cat .self:s ; echo
{
managementIP="192.168.8.217"
modifiedBy="spujare"
manufacturer="DELL COMPUTER CORPORATION"
use="UNKNOWN"
discoveredDate=1149012848000
origin="ASSIMILATED"
osSPVersion="SP4"
locale="English_United States.1252"
reporting=false
netBIOSName=
previousSWReg=1150673874000
osFlavor="Windows 2000 Advanced Server"
...
```

サーバーの属性も、attrディレクトリのファイルによって次のように表されます。

```
$ pwd
```

```
/opsw/Server/@/abc.example.com
$ cat attr/osFlavor ; echo
Windows 2000 Advanced Server
```

10 SA CLIメソッドを呼び出すスクリプトを作成します。

このステップに示すスクリプトの例は、All Windows Servers という名前のパブリックサーバーグループのすべてのサーバーを反復処理します。各サーバーに対して、スクリプトはgetCommCheckTime SA CLIメソッドを実行します。

初めに、OGFSのホームディレクトリに戻ります。

```
$ cd
$ cd public/bin
```

次に、viエディターを実行します。

```
$ vi
```

viで、次の各行を挿入してbashスクリプトを作成します。

```
#!/bin/bash
# iterate_time.sh

METHOD_DIR="/opsw/api/com/opsware/server/ServerService/method"
GROUP_NAME="All Windows Servers"
cd "/opsw/Group/Public/$GROUP_NAME/@/Server"

for SERVER_NAME in *
do
   SERVER_ID=`cat $SERVER_NAME/.self:i`
   echo $SERVER_NAME
   $METHOD_DIR/getCommCheckTime self:i=$SERVER_ID
   echo
   echo
done
```

viでファイルをiterate_time.shという名前で保存します。viを終了します。

iterate_time.shのアクセス権をchmodで変更し、実行します。

```
$ chmod 755 iterate_time.sh
$ ./iterate_time.sh
abc.example.com
2006/06/20 16:46:56.000
...
```

形式指定子

形式指定子は、SA CLI環境の値の表示や解釈の方法を指定します。形式指定子は、メソッドのパラメーター、メソッドの戻り値型、selfファイル、オブジェクト属性に適用できます。フォーマット指定子を指定するには、コロンと表2に示す文字の1つを追加します。

ファイルまたはメソッドの戻り値にフォーマット指定子を指定する場合、ファイルまたはメソッド名の前にピリオドを付ける必要があります。フォーマット指定子を持つメソッドパラメーターには、先頭のピリオドはありません。

表 2　　形式指定子の一覧

形式指定子	説明	有効なオブジェクトタイプ	メソッドパラメーターとしての使用
`:n`	**名前**: オブジェクトを識別する文字列。一意の名前であることが推奨されますが、必須ではありません。名前がないオブジェクトの場合、この表現はID表現と同じです。	SAオブジェクト	可能。If the name is ambiguous, an error occurs.
`:i`	**ID**: オブジェクトタイプとそのSA IDを一意に識別する形式。オブジェクト参照とも呼ばれます。	SAオブジェクト、日付 (`java.util.Calendar`) オブジェクト	可能。タイプがコンテキストから明らかな場合、タイプは省略できます。
`:s`	**構造**: コマンドラインで複合値を指定するためのコンパクトな表現。属性は中括弧で囲まれます。	任意の複合オブジェクト	可能
`:d`	**ディレクトリ**: 属性をOGFSのディレクトリで表します。	属性である任意の複合オブジェクト。この表現は、メソッドパラメーターまたは戻り値には使用できません。	不可

形式指定子の位置

形式指定子は、対象の項目のすぐ後に指定します。ファイルの場合、形式指定子はファイル名の後に置きます。次の例では、先頭のピリオドに注意してください。

```
cat .self:s
```

メソッドの戻り値型に適用する場合、形式指定子はメソッド名の後に置きます。次の呼び出しは、返されるグループのIDを表示します。

```
./.getDeviceGroups:i
```

メソッドパラメーターの場合、形式指定子は、次の例のように、パラメーター名の後、等号の前に置きます。

```
./setCustomer self:i=9977 customer:i=239
```

形式指定子を持つメソッドパラメーターには先頭のピリオドはありません。

デフォルトの形式指定子

すべての値またはオブジェクトには、デフォルトの形式指定子があります。たとえば、osVersion属性のデフォルトの形式指定子は名前です。次の2つのcatコマンドは、同じ出力を生成します。

```
cd /opsw/Server/@/d04.example.com/attr
cat osVersion
cat .osVersion:n
```

名前形式指定子は、サーバーやカスタマーなど、モデルリポジトリに格納されているSAオブジェクトに対するデフォルトです。その他の複合オブジェクトに対しては、構造形式指定子がデフォルトです。

ID形式指定子の例

次の例は、d04.example.comサーバーが属するファシリティのIDを表示します。

```
cd /opsw/Server/@/d04.example.com/attr
cat .facility:i ; echo
```

(前のechoコマンドはオプションです。これは、出力を読みやすくするための改行文字を生成します。セミコロンは、同じ行に入力されたbashステートメントを区切るためのものです)。

ID形式指定子による値の出力は、前にJavaクラス名が付きます。たとえば、ファシリティ値のIDが39の場合、前のcatコマンドは次の出力を表示します。

```
com.opsware.locality.FacilityRef:39
```

次のgetDeviceGroupsメソッドの呼び出しは、d04.example.comが属するパブリックサーバーグループのIDをリストします。

```
cd /opsw/Server/@/d04.example.com/method
./.getDeviceGroups:i
```

ID形式のその他の例については、selfファイル (35ページ) を参照してください。

構造形式指定子の構文

構造形式は、さまざまな属性を持つ複合オブジェクトを表現します。この形式は、複合オブジェクトをメソッドパラメーターに指定するために使用できます。例については、パラメーターとしての複合オブジェクトと配列 (39ページ) を参照してください。

構造形式は中括弧で囲んだ名前と値のペアの連続です。ペアの間は空白文字で区切られます。名前と値のペアは属性を表します。構造形式の構文は次のとおりです。

```
{ name-1=value-1 name-2=value-2 ...}
```

単純な例を次に示します。

```
{ version=10.1.3 isCurrent=true }
```

区切り文字としては任意の空白文字が使用できます。

```
{
  version=10.1.3
  isCurrent=true
}
```

属性は構造で指定することもできます。これにより、ネストしたオブジェクトを表現できます。次の例では、versionDesc属性が構造で表現されています。

```
{
program=agent
versionDesc={
        version=10.1.3
        isCurrent=true
        comment="Latest version"
        }
}
```

構造内部で配列を指定するには、属性名を繰り返します。次の構造には、steps という名前の配列があります。この配列は3要素で、値は33、14、28です。

```
{ moduleName="Some Initiator" steps=33 steps=14 steps=28 }
```

構造形式指定子の例

次の例は、facility属性の構造形式を指定します。

```
cd /opsw/Server/@/d04.example.com/attr
cat .facility:s
```

この cat コマンドは、次の出力を生成します。customers は配列で、このファシリティに関連付けられているすべてのカスタマーに対応する要素を持ちます。

```
{
modifiedBy="192.168.9.246"
customers="Customer Independent"
customers="Not Assigned"
customers="Opsware Inc."
customers="Acme Inc."
...
ontogeny="PROD"
createdBy=
status="ACTIVE"
createdDt=-1
realms="Transitional"
realms="C39"
realms="C39-agents"
modifiedDt=1146528752000
name="C39"
displayName="C39"
}
```

次のgetDeviceGroupsの呼び出しは、戻り値の構造形式指定子を示します。

```
cd /opsw/Server/@/d04.example.com/method
./.getDeviceGroups:s
```

このgetDeviceGroupsの呼び出しは、次の出力を表示します。d04.example.comは2つのサーバーグループに属するので、出力には2 つの構造が含まれます。各構造の devices 配列には、そのグループに属するサーバーが要素として含まれます。

```
{
dynamic=true
devices="m302-w2k-vm1.dev.example.com"
```

```
devices="d04.example.com"
...
status="ACTIVE"
public=true
fullName="Device Groups Public All Windows Servers"
description="test"
createdDt=-1
modifiedDt=1142019861000
parent="Public"
}

{
dynamic=true
devices="opsware-nibwp.build.example.com"
devices="glengarriff.snv1.dev.example.com"
devices="millstreet"
...
fullName="Device Groups Public z_testsrvgroup"
...
}
```

構造形式指定子は、値オブジェクト (VO) を取得するメソッドに対するデフォルトです。たとえば、次の2つのgetServerVOの呼び出しは等価です。

```
cd /opsw/Server/@/d04.example.com/method
./.getServerVO:s
./getServerVO
```

この例で、getServerVOは次の出力を表示します。

```
{
managementIP="192.168.198.93"
modifiedBy=
manufacturer="DELL COMPUTER CORPORATION"
use="UNKNOWN"
discoveredDate=1145308867000
origin="ASSIMILATED"
osSPVersion="RTM"
locale="English_United States.1252"
reporting=false
netBIOSName=
previousSWReg=1147678609000
osFlavor="Windows Server 2003, Standard Edition"
peerIP="192.168.198.93"
modifiedDt=1145308868000
...
serialNumber="HVKZS51"
}
```

この構造は、SA APIのServerVOクラスを表します。この構造のすべての属性は、それぞれattrディレクトリの1つのファイルに対応します。次の例では、getServerVOコマンドとcatコマンドはどちらもサーバーのserialNumber属性の値を表示します。

```
cd /opsw/Server/@/d04.example.com
./method/getServerVO | grep serialNumber
cat attr/serialNumber ; echo
```

ディレクトリ形式指定子の例

次のコマンドは、現在の作業ディレクトリを、サーバーd04.example.comに関連付けられているカスタマーに変更します。

```
cd /opsw/Server/@/d04.example.com/attr/.customer:d
```

次のコマンドは、このカスタマーの名前をリストします。

```
cat /opsw/Server/@/d04.example.com/attr/\
.customer:d/attr/name
```

ディレクトリ指定子は、ディレクトリ名を必要とするコマンド引数のみに使用できます。次のcatコマンドは、ディレクトリを表示しようとするため失敗します。

```
cat /opsw/Server/@/d04.example.com/attr/.customer:d # WRONG!
```

一方、次のコマンドは有効です。

```
ls /opsw/Server/@/d04.example.com/attr/.customer:d
```

値の表現

SA CLIメソッドは、シェル環境 (OGSH) で実行されるため、データを文字列で受け取って返します。一方、対応するAPIメソッドは、数値、ブール値、オブジェクトなどの他のデータ型を受け取って返します。ここでは、OGFSとSA CLIメソッドが文字列以外のデータ型を表現する方法を説明します。

OGFS内のSAオブジェクト

SAのデータモデルには、サーバー、サーバーグループ、カスタマー、ファシリティなどのオブジェクトが含まれます。OGFSでは、これらのオブジェクトはディレクトリ構造で表現されます。

```
/opsw/Customer
/opsw/Facility
/opsw/Group
/opsw/Library
/opsw/Realm
/opsw/Server
. . .
```

上のリストは一部だけを示しています。完全なリストを見るには、ls /opswと入力してください。

オブジェクト属性:

SAオブジェクトの属性は、attrサブディレクトリのテキストファイルによって表されます。各ファイルの名前は、属性の名前と一致します。ファイルの内容は、属性の値を示します。

たとえば、/opsw/Server/@/buzz.example.com/attrディレクトリには次のファイルが含まれます。

```
agentVersion
codeset
createdBy
createdDt
```

```
customer
defaultGw
description
discoveredDate
facility
hostName
locale
lockInfo
loopbackIP
managementIP
manufacturer
. . .
```

buzz.example.comサーバーの管理IPアドレスを表示するには、次のコマンドを入力します。

```
cd /opsw/Server/@/buzz.example.com/attr
cat managementIP ; echo
```

カスタム属性

カスタム属性は、サーバーなどのSAオブジェクトに割り当てることができる名前と値のペアです。OGFSでは、カスタム属性はCustAttrサブディレクトリのテキストファイルで表されます。カスタム属性を作成するには、OGSHセッションでCustAttrの下に新しいテキストファイルを作成します。次の例は、名前がMyGreetingで値がhello thereのカスタム属性をbuzz.example.comサーバーに作成します。

```
cd /opsw/Server/@/buzz.example.com/CustAttr
echo -n "hello there" > MyGreeting
```

その他の例については、『SAユーザーガイド: Server Automation』の「カスタム属性の管理」を参照してください。

selfファイル

selfファイルは、サーバーやカスタマーなどのSAオブジェクトのディレクトリにあります。このファイルは、形式指定子に応じて、現在のオブジェクトのさまざまな表現へのアクセスを可能にします(詳細については、形式指定子 (29ページ) を参照してください)。

buzz.example.comサーバーのIDをリストするには、次のコマンドを入力します。

```
cd /opsw/Server/@/buzz.example.com
cat .self:i ; echo
```

サーバーの場合、デフォルトの形式指定子は名前です。次のコマンドは同じ出力を表示します。

```
cat self ; echo
cat .self:n ; echo
```

次のコマンドは、サーバーの属性を構造形式でリストします。

```
cat .self:s
```

プリミティブ値

表3は、APIとSA CLIメソッドでのその文字列表現の間でのプリミティブ値の変換方法を示します。日付を除いて、プリミティブ値は形式指定子をサポートしません。日付はID形式指定子をサポートします。

表 3　　プリミティブ型とSA CLIメソッドの間の変換

プリミティブ型	Javaの相当する型	SA CLIメソッドの出力	SA CLIメソッドの入力
String	`java.lang.String`	現在のセッションのエンコードで表現された文字列。	現在のセッションのエンコードからUnicodeに変換された文字列。
数値	`byte`、`short`、`int`、`long`、`float`、`double`、およびそれらに相当するオブジェクト	10進形式。ローカライズなし。非常に大きいか小さい値の場合は科学的記数法。	例 - 10進: `101`、`512.34`、`-104` 16進: `0x1F32, 0x2e40` 8進: `0543` 科学的記数法: `4.3E4`、`6.532e-9`、`1.945e+02`
ブール値	`boolean`、`Boolean`	`true`または`false`	文字列true (大文字と小文字は区別されません) は、`true`と評価されます。それ以外の値は、`false`と評価されます。
バイナリデータ	`byte[]`、`Byte[]`	バイナリ文字列。セッションエンコードからの変換はありません。	バイナリ文字列。セッションエンコードへの変換はありません。
日付	java.util.Calendar	日付値。デフォルトでは次の形式で表現されます。 `YYYY/MM/DD HH:MM:SS.mmm` 時刻はUTCで表現されます。ID形式指定子を指定した場合、値はエポックからのミリ秒数 (UTC) で表されます。	出力と同じ。

配列

配列オブジェクトの表現は、配列が単独で (配列属性ファイルまたはメソッドの戻り値で) 使用されるか、複合オブジェクトの構造内に含まれるかによって異なります。

単独の配列オブジェクトは、元になる型に基づいて、改行文字を区切りとして表現されます。配列要素内の改行文字は\n、バックスラッシュは\\のようにエスケープされます。

配列値は、元になる型でサポートされる任意の表現で出力または入力できます。たとえば、デフォルトでは、getDeviceGroupsメソッドはグループを名前でリストします。

```
All Windows Servers
Servers in Austin
Testing Pool
```

ID形式指定子を指定した場合 (.getDeviceGroups:i)、メソッドはグループのIDを表示します。

```
com.opsware.device.DeviceGroupRef:15960039
com.opsware.device.DeviceGroupRef:10390039
com.opsware.device.DeviceGroupRef:17380039
```

複合オブジェクトの構造内に含まれる配列は、属性を名前に使用して、名前と値のペアの集合で表現されます。属性は複数回、すなわち配列の各要素に対して1回ずつ現れます。属性が現れる順序によって、配列内の要素の順序が決まります。次の例に示す構造には、2つの要素が含まれています。1つはsubjectという文字列、もう1つはranksという3要素の数値配列です。

```
{ subject=my favorites ranks=17 ranks=44 ranks=24 }
```

配列はディレクトリで表現することもできます。配列ディレクトリ内の各配列要素には、対応するファイル (プリミティブ型の場合) またはサブディレクトリ (複合型の場合) があります。各エントリの名前は、配列要素のインデックス番号 (先頭が0) です。

複合オブジェクトの属性である配列を変更するには、その属性ファイルを編集します。この操作を行うと、編集したファイルの内容によって配列全体が置き換えられます。

要素が複合オブジェクトである配列を変更するには、そのディレクトリ表現を変更します。要素の値を変更するには、要素ファイルを編集します。たとえば、5個の文字列要素からなる配列があるとします。lsコマンドを実行すると、次のように要素が表示されます。

```
0 1 2 3 4
```

次のコマンドは、3番目の要素の値を変更します。

```
echo -n "My new value" > 2
```

SA CLIメソッドのパラメーターと戻り値

ここでは、メソッドのコンテキスト (インスタンスまたは静的)、パラメーターの使用法、戻り値、終了ステータスについて説明します。

メソッドのコンテキストとselfパラメーター

OGFSでは、メソッドは複数の場所に存在します。メソッドの場所はそのコンテキスト (インスタンスまたは静的) に関連しています。

インスタンスコンテキストのメソッドは、特定のSAオブジェクトのmethodディレクトリに存在します。メソッドの呼び出しにselfパラメーターは必要ありません。メソッドによって影響されるオブジェクトのインスタンスは、メソッドの場所によって示されます。次の例は、d04.example.comサーバーのカスタマーを変更します。

```
cd /opsw/Server/@/d04.example.com/method
./setCustomer customer:i=9
```

静的コンテキストのメソッドは、/opsw/apiの下の1つの場所にあります。メソッドの呼び出しには、影響されるインスタンスを指定するためのselfパラメーターが必要です。次の静的コンテキストの例で、self:iは管理対象サーバーのIDを指定します。

```
cd /opsw/api/com/opsware/server/ServerService/method
./setCustomer self:i=230054 customer:i=9
```

コマンドラインでの引数渡し

コマンドライン引数は、名前と値のペアを等号 (=) で結んだもので指定されます。名前と値のペアとペアの間は、1つ以上の空白文字 (通常はスペース) で区切ります。コマンドラインに指定する名前は、SA APIの対応するJavaメソッドのパラメーター名と同じです。

たとえば、SA APIのsetCustomFieldメソッドは次のように定義されています。

```
public void setCustomField(CustomFieldReference self,
   java.lang.String fieldName, java.lang.String strValue)...
```

次のSA CLIメソッドの例は、IDが3670039のサーバーのカスタムフィールドに値を割り当てます。

```
cd /opsw/api/com/opsware/server/ServerService/method
./setCustomField self:i=3670039 \
fieldName="Service Agreement" strValue="Gold"
```

前に説明したように、インスタンスコンテキストのメソッドにはself パラメーターは不要です。次のsetCustomFieldの例は前の例と等価です。

```
cd /opsw/.Server.ID/3670039
./setCustomField \
fieldName="Service Agreement" strValue="Gold"
```

コマンドライン引数は任意の順序で指定できます。次の2つのSA CLIメソッド呼び出しは等価です。

```
./setCustomField fieldName="My Stuff" strValue="abc"
./setCustomField strValue="abc" fieldName="My Stuff"
```

パラメーターにNull値を指定するには、パラメーターを省略するか、等号の後ろに空白を入れます。次の例で、myParamの値はNullです。

```
./someMethod myField="more info" myParam= anotherParam=9834
./someMethod myField="more info"          anotherParam=9834
```

パラメーターの型の指定

メソッドのパラメーターに抽象型が指定されている場合、値とともに具象型を指定する必要があります。次の例では、com.opsware.folder.FolderRef型が必要です。

```
cd /opsw/api/com/opsware/folder/FolderService/method
./remove self:i="com.opsware.folder.FolderRef:730555"
```

具象型を指定しない場合、次のエラーメッセージが表示されます。

オブジェクト型型名は抽象型です。具象型を指定してください。

パラメーターとしての複合オブジェクトと配列

複合オブジェクトを引数として渡すには、構造形式指定子の構文 (31ページ) に示すように、オブジェクトの属性を中括弧で囲みます。

次の例は、AllMineというパブリックサーバーグループを作成します。createメソッドにはpatternという1つのパラメーターがあり、属性parentとshortNameが中括弧に囲まれています。この例で、getPublicRootはトップパブリックグループのIDである2340555を返します。

```
cd /opsw/api/com/opsware/device/DeviceGroupService/method
./.getPublicRoot:i ; echo
./create "pattern={ parent:i=2340555 shortName='AllMine' }"
```

配列パラメーターを指定するには、各配列要素に対して1回ずつパラメーター名を繰り返し指定します。たとえば、次のassignメソッドの呼び出しは、policiesという名前の配列パラメーターの最初の2つの要素を指定しています。

```
cd /opsw/api/com/opsware/swmgmt
cd SoftwarePolicyService/method
./attachPolicies self:i=4220039 \
policies:i=4400335 policies:i=4400942
```

オーバーロードされたメソッド

Javaメソッド名がオーバーロードされるのは、同じクラスに同じ名前の複数のメソッドがあり、それらのパラメーターリストが異なっている場合です。SA CLIメソッドがオーバーロードされている場合、コマンドラインの引数名によって、どのメソッドが呼び出されるかが決まります。たとえば、setCustomFieldメソッドは、異なるデータ型の設定をサポートするためにオーバーロードされています。次の2つのコマンドは、それぞれメソッドの異なるバージョンを呼び出します。

```
./setCustomField \
fieldName="Service Agreement" strValue="Gold"
./setCustomField \
fieldName=hmp longValue=2245
```

戻り値

SA CLIメソッドに対応するAPIメソッドが値を返す場合、SA CLIメソッドは値をstdoutに出力します。Unixコマンドの場合と同様、メソッドのstdoutをファイルにリダイレクトしたり、環境変数に代入したりできます。

戻り値の表現を変更するには、メソッド名の前にピリオドを付け、後ろに形式指定子を追加します。次の例は、サーバー参照をデフォルトの名前でなくIDで返します。

```
cd /opsw/api/com/opsware/server/ServerService/method
./.findServerRefs:i
```

メソッドの戻り値型と互換性のない形式指定子を指定した場合、ファイルシステムはエラーを生成します。

終了ステータス

Unix シェルコマンドと同様、SA CLI メソッドは終了ステータス ($?) で呼び出しの結果を示します。終了ステータスが0の場合は成功を示し、0以外の場合はエラーを示します。SA CLIメソッドはエラーメッセージをstderrに出力します。

表 4　　SA CLIメソッドの終了ステータスコード

終了ステータス	カテゴリ	説明
0	成功	メソッドは正常に完了しました。
1	コマンドライン解析エラー	メソッド呼び出しのコマンドラインの形式が正しくなく、オプション (--option[=value]) とパラメーター値 (param=value) のセットに解析できません。
2	パラメーター解析エラー	パラメーター値を API が必要とするオブジェクトタイプに解析できません。
3	API使用法エラー	無効なパラメーター値などの使用法エラーによって呼び出しが失敗しました。
4	アクセスエラー	ユーザーにはこの操作を実行するアクセス権がありません。
5	その他のエラー	終了ステータス1～4で示されるもの以外のエラーが発生しました。

たとえば、次のbashスクリプトは、getDeviceGroupsメソッドの終了ステータスを確認します。

```
#!/bin/bash

cd /opsw/Server/@/toro.snv1.corp.example.com/method
./getDeviceGroups
cmnd_exit_status=$?

if [ $cmnd_exit_status -eq 0 ]
then
  echo "The command was successful."
else
  echo "The command failed."
```

```
  echo "Exit status = "  $cmnd_exit_status
fi
```

各SA CLIメソッドは、対応する1つのAPIメソッドを呼び出します。APIメソッドで例外が発生した場合、SA CLIメソッドは0以外の終了ステータスを返します。メソッド呼び出しをデバッグする場合、発生した例外に関する情報を表示できると役に立つことがあります。OGFSの /sys/last-exceptionファイルには、最近のAPI呼び出しから発生した例外のスタックトレースが記録されています。このファイルを読み取ると、ファイルの内容は破棄されます。

検索フィルターとSA CLIメソッド

SA APIには、オブジェクト参照をパラメーターとして受け取るメソッドが多数あります。検索基準に基づいてオブジェクト参照を取得するには、findServerRefsやfindJobRefsなどのメソッドを呼び出します。たとえば、findServerRefsを呼び出すと、hostname属性にexample.comを持つすべてのサーバーを検索することができます。

検索の構文

findServerRefsなどのメソッドは、次の構文を取ります。

findオブジェクトRefs filter=[オブジェクトタイプ:]式

filterパラメーターには、検索条件を指定する式が含まれます。式は括弧または中括弧で囲みます。単純な式の構文を次に示します。

```
value-object.attribute operator value
```

(この構文は単純化されています。完全な定義については、フィルターの文法 (161 ページ) を参照してください)。

検索の例

SAオブジェクトタイプのほとんどには、対応する検索メソッドがあります。ここでは、そのうちいくつかの使用法を示します。他のSA CLIメソッドでの検索の使用法については、スクリプトの例 (44ページ) を参照してください。

サーバーの検索

ホスト名にexample.comを含むサーバーを検索します。

```
cd /opsw/api/com/opsware/server/ServerService/method
./.findServerRefs:i \
filter='device:{ ServerVO.hostname CONTAINS example.com }'
```

use属性の値がUNKNOWNまたはPRODUCTIONのサーバーを検索します。

```
cd /opsw/api/com/opsware/server/ServerService/method
./.findServerRefs:i \
filter='{ ServerVO.use IN "UNKNOWN" "PRODUCTION" }'
```

次のbashスクリプトは、サーバーを検索し、それらのIDを一時ファイルに保存し、各IDを別のメソッド呼び出しのパラメーターとして指定する方法を示します。このスクリプトは、各Linuxサーバーが属するパブリックグループを表示します。

```
#!/bin/bash
```

```
TMPFILE=/tmp/server-list.txt
rm -f $TMPFILE

cd /opsw/api/com/opsware/server/ServerService/method

./.findServerRefs:i \
filter='{ ServerVO.osVersion CONTAINS Linux }' > $TMPFILE

for ID in `cat "$TMPFILE"`
do
  echo Server ID:$ID
  ./getDeviceGroups self:i=$ID
  echo
done
```

ジョブの検索

ここに示す例は、サーバー監査やポリシー修復などのジョブのIDを返します。

正常に完了したジョブを検索します。

```
cd /opsw/api/com/opsware/job/JobService/method
./.findJobRefs:i filter='job:{ job_status = "SUCCESS" }'
```

(job_statusに使用できる値の一覧については、『SA統合ガイド』の「ジョブ承認の統合」を参照してください)。

正常に完了したか、警告が発生して完了したジョブを検索します。

```
cd /opsw/api/com/opsware/job/JobService/method
./.findJobRefs:i \
filter='job:{ job_status IN "SUCCESS" "WARNING" }'
```

今日開始されたジョブを検索します。

```
cd /opsw/api/com/opsware/job/JobService/method
./.findJobRefs:i \
filter='job:{ JobInfoVO.startDate IS_TODAY "" }'
```

すべてのサーバー監査ジョブを検索します。

```
cd /opsw/api/com/opsware/job/JobService/method
./findJobRefs \
filter='job:{ JobInfoVO.description = "Server Audit" }'
```

IDが280039のサーバー上で実行されたジョブを検索します。

```
cd /opsw/api/com/opsware/job/JobService/method
./.findJobRefs:i filter='job:{ job_device_id = "280039" }'
```

Find today's jobs that have failed:

```
cd /opsw/api/com/opsware/job/JobService/method
./.findJobRefs:i \
filter='job:{ (( JobInfoVO.startDate IS_TODAY "" ) \
& ( job_status = "FAILURE" )) }'
```

その他のオブジェクトの検索

ここでは、ソフトウェアポリシーおよびパッケージを検索する例を示します。

jdoeというSAユーザーによって作成されたソフトウェアポリシーを検索します。

```
cd /opsw/api/com/opsware/swmgmt/SoftwarePolicyService/method
./.findSoftwarePolicyRefs:i \
filter='{ SoftwarePolicyVO.createdBy CONTAINS jdoe }'
```

名前にismtoolが含まれるWindows 2003プラットフォーム用のMSIを検索します。

```
cd /opsw/api/com/opsware/pkg/UnitService/method
./.findUnitRefs:i \
filter='software_unit:{ ((UnitVO.unitType = "MSI") \
& ( UnitVO.name contains "ismtool" ) \
& ( software_platform_name = "Windows 2003" )) }'
```

名前が117170-01のSolarisパッチを検索します。

```
cd /opsw/api/com/opsware/pkg/solaris/SolPatchService/method
./.findSolPatchRefs:i filter='{name = 117170-01}'
```

名前にTestという文字列を含み、親フォルダー名がMy Stuffのフォルダーを検索します。

```
cd /opsw/api/com/opsware/folder/FolderService/method
./.findFolders:s \
filter='( ( FolderVO.name CONTAINS "Test" ) \
& (folder_parent_name = "My Stuff" ) )'
```

検索可能な属性と有効な演算子

値オブジェクトの属性の中には、検索フィルターで使用できないものがあります。たとえば、ServerVO.useで検索することはできますが、ServerVO.OsFlavorで検索することはできません。

オブジェクトタイプのどの属性が検索可能かを知るには、getSearchableAttributesメソッドを呼び出します。次の例は、検索式に指定できるServerVOの属性のリストを表示します。

```
cd /opsw/api/com/opsware/search/SearchService/method
./getSearchableAttributes searchableType=device
```

searchableTypeパラメーターは、オブジェクトタイプを示します。searchableTypeで使用可能な値を知るには、次のコマンドを入力します。

```
cd /opsw/api/com/opsware/search/SearchService/method
./getSearchableTypes
```

属性に対してどの演算子が有効かを知るには、getSearchableAttributeOperatorsメソッドを呼び出します。次の例は、属性ServerVO.hostnameに対して有効な演算子 (CONTAINS、INなど) のリストを表示します。

```
cd /opsw/api/com/opsware/search/SearchService/method
./getSearchableAttributeOperators searchableType=device \
searchableAttribute=ServerVO.hostname
```

スクリプトの例

ここでは、さまざまなSA CLIメソッドを呼び出す単純なbashスクリプトのコードリストを示します。これらのスクリプトは、コマンドラインでメソッドパラメーターを渡す方法を示しており、複合オブジェクトやselfパラメーターも使用されています。これらのスクリプトの例をコピーして再使用する場合は、コード内に書かれているオブジェクト名 (d04.example.com) を適切に変更してください。OGFSでスクリプトを作成して実行する手順のチュートリアルについては、手順10 (29ページ) を参照してください。

スクリプトremediate_policy.sh (46ページ) は、ソフトウェアポリシーを作成し、ポリシーにパッケージを追加し、最後の行で、startFullRemediateNowメソッドを呼び出してパッケージを管理対象サーバーにインストールします。

create_custom_field.sh

このスクリプトは、TestFieldAという名前のカスタムフィールド (仮想列) を作成し、フィールドをすべてのサーバーにアタッチし、1つのサーバーに対してこのフィールドの値を設定します。カスタムフィールドはアタッチされるまでSAクライアントには表示されません。カスタムフィールドは、サーバー、デバイスグループ、ソフトウェアポリシーに対して作成できます。カスタムフィールドを作成するには、SAユーザーが属するユーザーグループに仮想列の管理アクセス権が必要です。

カスタム属性と異なり、カスタムフィールドはそのタイプのすべてのインスタンスに適用されます。OGFSでカスタム属性を作成する例については、『SAユーザーガイド: Server Automation』の「カスタム属性の管理」を参照してください。

create_custom_field.shスクリプトには、次のコードがあります。

```
#!/bin/bash
# create_custom_field.sh

cd /opsw/api/com/opsware/custattr/VirtualColumnService/method

# Create a virtual column.
# Remember the name because you cannot search for the
# displayName.
./create vo='{ name=TestFieldA type=SHORT_STRING \
displayName="Test Field A" }'

column_id=`./.findVirtualColumn:i name=TestFieldA`

echo --- column_id = $column_id

cd /opsw/api/com/opsware/server/ServerService/method

# Attach the column to all servers.
# All servers will have this custom field.
./attachVirtualColumn virtualColumn:i=$column_id

# Get the ID of the server named d04.example.com
devices_id=`./.findServerRefs:i \
filter=\
'device:{ ServerVO.hostname CONTAINS "d04.example.com" }'`

echo --- devices_id = $devices_id
```

```
# Set the value of the custom field (virtual column) for
# a specific server.
./setCustomField self:i=$devices_id fieldName=TestFieldA \
strValue="This is something."
```

create_device_group.sh

このスクリプトは、静的デバイスグループを作成し、そのグループにサーバーを追加します。次に、動的グループを作成し、そのグループにルールを設定し、グループのメンバーを更新します。最後のステートメントは、動的グループに属するデバイスのリストを表示します。

スクリプトのコードを次に示します。

```
#!/bin/bash
# create_device_group.sh

cd /opsw/api/com/opsware/device/DeviceGroupService/method

# Get the ID of the public root group (top of hierarchy).
public_root=`./.getPublicRoot:i`

# Create a public static group.
./create "vo={ parent:i=$public_root shortName='Test Group A' }"

# Get the ID of the group just created.
group_id=`./.findDeviceGroupRefs:i \
filter='{ DeviceGroupVO.shortName = "Test Group A" }' `

echo --- group_id = $group_id

cd /opsw/api/com/opsware/server/ServerService/method

# Get the ID of the server named d04.example.com
devices_id=`./.findServerRefs:i \
filter=\
'device:{ ServerVO.hostname CONTAINS "d04.example.com" }'`

echo --- devices_id = $devices_id

cd /opsw/api/com/opsware/device/DeviceGroupService/method

# Add a server to the device group.
./addDevices \
self:i=$group_id devices:i=$devices_id

# Create a dynamic device group.
./create \
"vo={ parent:i=$public_root \
shortName='Test Dyn B' dynamic=true }"

# Get the ID of the device group.
dynamic_group_id=`./.findDeviceGroupRefs:i \
filter='{ DeviceGroupVO.shortName = "Test Dyn B" }' `

echo --- dynamic_group_id = $dynamic_group_id
```

```
# Set the rule so that this group contains servers with
# hostnames containing the string example.com.
# The rule parameter has the same syntax as the filter
# parameter of the find methods.
./setDynamicRule self:i=$dynamic_group_id \
rule='device:{ ServerVO.hostname CONTAINS example.com }'

# By default, membership in dynamic device groups is refreshed
# once
# an hour, so force the refresh now.
./refreshMembership selves:i=$dynamic_group_id now=true

# Display the names of the devices that belong to the group.
echo --- Devices in group:
./getDevices selves:i=$dynamic_group_id
```

create_folder.sh

このスクリプトは、/Test 1というフォルダーを作成し、ルート (/) フォルダーの下のフォルダーのリストを表示し、/Test 1/Test 2というサブフォルダーを作成します。作成したフォルダーは、SAクライアントのナビゲーションペインの「ライブラリ」の下に表示されます。

このスクリプトのコードを次に示します。

```
#!/bin/bash
# create_folder.sh

cd /opsw/api/com/opsware/folder/FolderService/method

# Get the ID of the root (top) folder.
root_id=`./.getRoot:i`

# Create a new folder under the root folder.
./create vo="{ name='Test 1' folder:i=$root_id }"

# Display the names of the folders under the root folder.
./getChildren self:i=$root_id

# Get the ID of the folder "/Test 1"
folder_id=`./.getFolderRef:i path="Test 1"`

# Create a subfolder.
./create vo="{ name='Test 2' folder:i=$folder_id }"

# Get the ID of the folder "/Test 1/Test 2"
folder_id=`./.getFolderRef:i path="Test 1" path="Test 2"`
echo folder_id = $folder_id
```

remediate_policy.sh

このスクリプトは、Test 2という名前の既存のフォルダーにTestPolicyAという名前のソフトウェアポリシーを作成し、ismtoolを含むパッケージをこのポリシーに追加し、ポリシーを1つのサーバー (グループでなく) にアタッチし、そのサーバーを修復します。修復操作は、パッケージをサーバーにインストールするジョブを起動します。ジョブの進行状況と結果はSAクライアントで確認できます。SA CLIメソッドでジョブを検索する例については、ジョブの検索 (42ページ) を参照してください。

このスクリプトでは、SoftwarePolicyServiceのcreateメソッドで、platformsパラメーターの値がコードに直接書き込まれています。ここにあるスクリプト例では、できる限りオブジェクトを名前で検索し、値を直接書き込まないようにしています。プラットフォームの場合は、name属性での検索が困難です。この属性はSAクライアントに表示されるdisplayName属性(検索不可)とは異なっているからです。プラットフォームIDを知る最も簡単な方法は、twisterでPlatformService.findPlatformRefsメソッドをパラメーターなしで実行することです。

このスクリプトのupdateメソッドには、softwarePolicyItemsのIDが直接書き込まれています。ソフトウェアリポジトリに似た名前のパッケージが多数含まれている場合、このオブジェクトは名前で検索するのが困難だからです。IDを知る1つの方法は、SAクライアントを実行し、ファイル名やオペレーティングシステムなどのフィールドでソフトウェアを検索し、検索で見つかったパッケージを開き、パッケージのプロパティ表示にあるSA IDを記録することです。

次のリストでは、updateメソッドの中に不要な改行が入っています。このコードをコピーする場合、voパラメーターが1行になるようにスクリプトを編集してください。

remediate_policy.shスクリプトのソースコードを次に示します。

```
#!/bin/bash
# remediate_policy.sh

# Get the ID of the folder where the policy will reside.
cd /opsw/api/com/opsware/folder/FolderService/method
folder_id=`./.findFolders:i filter='{ FolderVO.name = "Test 2" }'`

cd /opsw/api/com/opsware/swmgmt/SoftwarePolicyService/method

# Create a software policy named TestPolicyA.
# This policy resides in the folder located in the preceding findFolders
# call.
# The platform for this policy is Windows 2008 (ID 160076)
./create vo="{ platforms:i=160076 name="TestPolicyA" \
folder:i=$folder_id lifecycle=AVAILABLE }"

policy_id=`./.findSoftwarePolicyRefs:i \
filter='{ SoftwarePolicyVO.name = "TestPolicyA" }'`

echo --- policy_id = $policy_id

# Call the update method to add a package to the software policy.
# The package ID for the "ismtool" msi installer is 4010001.
# Note that "force = true" is required.
./update self:i=$policy_id force=true \
vo='{ softwarePolicyItems:i=com.opsware.pkg.windows.MSIRef:4010001 }'

cd /opsw/api/com/opsware/server/ServerService/method

# Get the ID of the server named d04.opsware.com
devices_id=`./.findServerRefs:i \
filter='device:{ ServerVO.hostname CONTAINS "d04.opsware.com" }'`

echo --- devices_id = $devices_id
```

```
# Attach the policy to a single server (not a group).
./attachPolicies self:i=$devices_id \
policies:i=$policy_id

# Remediate the server to install the package in the policy.
job_id=`./.startFullRemediateNow:i self:i=$devices_id`

echo --- job_id = $job_id
```

remove_custom_field.sh

運用環境ではほとんどありませんが、テスト環境ではカスタムフィールドの削除が必要になることがあります。カスタムフィールドを削除するには、先にアタッチを解除しておく必要があります。

remove_custom_field.shのコードを次に示します。

```
#!/bin/bash
# remove_custom_field.sh

if [ !-n "$1" ]
   then
   echo "Usage:‘basename $0‘ <name>"
   echo "Example:‘basename $0‘ hmp"
   exit
fi

cd /opsw/api/com/opsware/custattr/VirtualColumnService/method

column_id=‘./.findVirtualColumn:i name=$1‘

echo --- column_id = $column_id

cd /opsw/api/com/opsware/server/ServerService/method

# Column must be detached before it can be removed.
./detachVirtualColumn virtualColumn:i=$column_id

cd /opsw/api/com/opsware/custattr/VirtualColumnService/method

# Remove the virtual column.
./remove self:i=$column_id
```

schedule_audit_task.sh

このスクリプトは、監査タスクを開始し、未来の日付にスケジュールします。SA CLIメソッドでは、日付パラメーターは次の構文で指定されます。

```
YYYY/MM/DD HH:MM:SS.sss
```

タスクを起動するメソッドstartAuditは、監査を実行するジョブのIDを返します。SA CLIメソッドでジョブを検索する例については、ジョブの検索 (42ページ) を参照してください。

schedule_audit_task.shのコードを次に示します。

```
#!/bin/bash
# schedule_audit_task.sh

cd /opsw/api/com/opsware/compliance/sco/AuditTaskService/method

# Get the ID of the audit task to schedule.
audit_task_id=`./.findAuditTask:i \
filter='audit_task:{ (( AuditTaskVO.name BEGINS_WITH "HW check" ) \
& ( AuditTaskVO.createdBy = "gsmith" )) }'`

echo --- audit_task_id = $audit_task_id

# Schedule the audit task for Oct. 16, 2013.
# In the startDate parameter, note that the last delimiter for the time
# is a period, not a colon.
job_id=`./startAudit self:i=$audit_task_id \
schedule:s='{ startDate="2013/10/16 00:00:00.000" }' \
notification:s='{ onFailureOwner="sjones@opsware.com" \
onFailureRecipients="jdoe@opsware.com" \
onSuccessOwner="sjones@opsware.com" \
onSuccessRecipients="jdoe@opsware.com" }'`

echo --- job_id = $job_id
```

SA CLIメソッドの使用法情報の取得

将来のリリースでは、SA CLIメソッドで使用法情報を表示できるようになる予定です。それまでは、次に示す方法でAPIドキュメントまたはOGFSから必要な情報を得ることができます。

サービスのリストの表示

SA APIメソッドは、サービスによって分類されています。SA CLIメソッドに対して使用可能なサービスを知るには、OGSHセッションに次のコマンドを入力します。

```
cd /opsw/api/com/opsware
find .-name "*Service"
```

APIドキュメントでサービスのリストを表示するには、ブラウザーに次のURLを指定します。

```
https://OCCホスト:1032
```

OCCホストは、コマンドセンターコンポーネントを実行しているコアサーバーのIPアドレスまたはホスト名です。

APIドキュメントでのサービスの検索

OGFS のサービスのパスは、API ドキュメントの Java パッケージ名に対応します。たとえば、OGFS の ServerServiceメソッドは次のディレクトリにあります。

```
/opsw/api/com/opsware/server
```

APIドキュメントでは、これらのメソッドは次のインタフェースで定義されます。

```
com.opsware.server.ServerService
```

サービスのメソッドのリストの表示

OGFSでは、サービスのmethodディレクトリの内容のリストを表示できます。たとえば、ServerServiceのメソッド名を表示するには、次のコマンドを入力します。

```
ls /opsw/api/com/opsware/server/ServerService/method
```

APIドキュメントでは、次の手順でServerServiceのメソッドを表示できます。

1 左上のペインでcom.opsware.serverを選択します。

2 左下のペインでServerServiceを選択します。

3 メインペインで、下にスクロールしてメソッドを表示します。

メソッドのパラメーターのリストの表示

API ドキュメントで、前の項に示した手順を実行します。サービスインタフェースのページの「メソッドの詳細」の項で、パラメーターと戻り値の型を確認します(メソッドのパラメーターの詳細については、コマンドラインでの引数渡し (38ページ) を参照してください)。

値オブジェクトに関する情報の取得

APIドキュメントには、いくつかのサービスメソッドが値オブジェクト (VO) を引数に取るか値として返すことが記述されています。値オブジェクトにはデータメンバー (属性) が含まれます。たとえば、ServerService.getServerVOメソッドはServerVOオブジェクトを返します。ServerVOに含まれる属性を知るには、次の手順を実行します。

1 APIドキュメントで、ServerVOリンクを選択します。このリンクはいくつかの場所にあります。

- getServerVOのメソッドシグネチャ
- com.opsware.serverのクラスのリスト (左下のペイン)
- 索引ページ。索引ページへのリンクは、APIドキュメントのメインペインの上部にあります。

2 ServerVOページで、getterメソッドとsetterメソッドを見つけます。各getter-setterペアは、値オブジェクトの1つの属性に対応します。たとえば、getCustomerとsetCustomerは、ServerVOにcustomerという属性があることを示します。

属性が変更可能かどうかの判定

クライアントアプリケーションから変更できるオブジェクト属性は一部だけです。属性が変更可能かどうかを知るには、次の手順を実行します。

1 前の項に示した手順で、APIドキュメントの値オブジェクトのページを開きます。

2 setter メソッドの「メソッドの詳細」の項で、「フィールドはクライアントから設定可能」と記されているかどうかを確認します。

サーバーやカスタマーなど、OGFSで表現されるSAオブジェクトの場合、attrディレクトリ内のファイルのアクセスタイプを調べることで、どの属性が変更可能かを知ることができます。読み取り / 書き込み (rw) アクセスタイプを持つファイルは、変更可能な属性に対応します。たとえば、サーバーの変更可能な属性のリストを表示するには、次のコマンドを入力します。

```
cd /opsw/Server/@/server-name/attr
ls -l | grep rw
```

属性がフィルタークエリで使用可能かどうかの判定

値オブジェクトの属性がフィルタークエリ (検索) で使用可能かどうかを知るには、次の手順を実行します。

1 APIドキュメントで、値オブジェクトのページを開きます。

2 属性に対応するgetterメソッドの「メソッドの詳細」の項で、「フィールドはフィルタークエリに使用可能」と記されているかどうかを確認します。

属性が検索可能かどうかをOGSHセッション内で知るには、検索可能な属性と有効な演算子 (43ページ) に記されている方法を使用します。

第 3 章　PytwistによるPython APIアクセス

Pytwistの概要

Pytwistは、管理対象サーバーおよびカスタム拡張からSA APIにアクセスするためのPythonライブラリのセットです(twistはWebサービスデータアクセスエンジンの内部名です)。管理対象サーバーに対して、Pytwistを使用してSA APIを呼び出すPythonスクリプトを用意すれば、エンドユーザーはスクリプトをDSEまたはISMコントロールとして呼び出すことができます。カスタム拡張は、HP SA プロフェッショナルサービスによって作成され、コマンドエンジン (way) で動作するPythonスクリプトです。Pytwistを使うと、SAデータモデルに最近追加されたフォルダーやソフトウェアポリシーなどにアクセスできます。これらはコマンドエンジンスクリプトからはアクセスできません。

この章の内容は、SAのデータモデル、カスタム拡張、エージェント、Pythonプログラミング言語に関する知識がある開発者やコンサルタントを対象としています。

Pytwistのセットアップ

この章の例を試す前に、環境が以下に詳述するセットアップ要件を満たすことを確認してください。

Pytwistでサポートされるプラットフォーム

Pytwistは、管理対象サーバーとコアサーバーでサポートされます。これらのサーバーでサポートされるオペレーティングシステムのリストについては、『SA Release Notes』を参照してください。

PytwistはPythonバージョン2.4に依存します。これはSAのエージェントとカスタム拡張が使用するバージョンと同じです。

WebサービスやJava RMIクライアントと異なり、Pytwistクライアントは内部SAライブラリに依存します。クライアントプログラムが管理対象サーバーでもコアサーバーでもないサーバーからSA APIにアクセスする必要がある場合は、PytwistでなくWebサービスまたはJava RMIクライアントを使用してください。

Pytwistのアクセス要件

Pytwistは、Webサービスデータアクセスエンジンを実行しているコアサーバーのポート1032にアクセスする必要があります。エンジンはデフォルトでポート1032をリッスンします。

管理対象サーバーへのPytwistのインストール

SAのインストールまたはアップグレードの際に、Pytwistライブラリはコマンドエンジンコンポーネントとともにコアサーバー上に配置されます。したがって、カスタム拡張で使用するためにPytwistをインストールする必要はありません。

一方、エージェントのインストールにはPytwistは含まれていません。管理対象サーバーにPytwistをインストールするには、PytwistのZIPファイルを含むポリシーを修復します。SAクライアントで、PytwistのZIPファイルは次のフォルダーにあります。

```
/Opsware/Tools/Python Opsware API Access
```

このフォルダーには、各プラットフォームに対するPytwistのZIPファイルを含む作成済みのソフトウェアポリシーもあります。たとえば、Windows Python SA APIアクセスという名前のポリシーには、Windows XP、2000、2003など用のZIPファイルが含まれています。このポリシーを修復すると、プラットフォームバージョンに一致するZIPファイルだけがインストールされます。たとえば、Windows 2003サーバーに対してポリシーを修復した場合、Windows 2003用のZIPファイルだけがインストールされます。

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

1 SAクライアントで、[デバイス] の下の管理対象サーバーを見つけます。

2 内容ペインで、管理対象サーバーを開きます。

3 [管理対象サーバー] ウィンドウで、[**アクション**] メニューから [**ソフトウェアのインストール**] を選択します。

4 [ソフトウェアのインストール] ウィンドウで、ソフトウェアポリシー (Windows Python SA APIアクセスなど) を選択します。

5 [**インストール**] をクリックします。

6 [修復] ウィザードのステップを [サマリーレビュー] ウィンドウが表示されるまで実行します。

7 [**ジョブの開始**] をクリックします。

Pytwistの例

ここに示すPythonコードの例は、管理対象サーバーからの情報の取得、フォルダーの作成、ソフトウェアポリシーの修復の方法を示します。それぞれのPytwistの例は、次の操作を実行します。

1 パッケージのインポート。

SA API名前空間のオブジェクト (`Filter`など) をインポートする場合、パスではJavaパッケージ名の前に`pytwist`が付きます。`get_server_info.py`の例の`import`ステートメントを次に示します。

```
import sys
from pytwist import *
from pytwist.com.opsware.search import Filter
```

2 `TwistServer`オブジェクトを作成します。

```
ts = twistserver.TwistServer()
```

メソッドの引数に関する情報については、TwistServerメソッドの構文 (59ページ) を参照してください。

3 サービスへの参照を取得します。

サービスのPythonパッケージ名はJavaパッケージ名と同じですが、先頭の`opsware.com`がありません。たとえば、Javaの`com.opsware.server.ServerService`パッケージはPytwistの`server.ServerService`に対応します。

```
serverservice = ts.server.ServerService
```

4 サービスのSA APIメソッドを呼び出します。

```
filter = Filter()
...
servers = serverservice.findServerRefs(filter)
...
for server in servers:
  vo = serverservice.getServerVO(server)
...
```

get_server_info.py

このスクリプトは、ホスト名がコマンドライン引数を含むすべての管理対象サーバーを検索します。検索メソッドfindServerRefsは、サーバー永続的オブジェクトへの参照の配列を返します。各参照に対して、getServerVOメソッドは値オブジェクト (VO) を返します。これはサーバーの属性を保持するデータ表現です。get_server_info.pyスクリプトのコードを次に示します。

```
#!/opt/opsware/bin/python
# get_server_info.py

# Search for servers by partial hostname.

import sys
sys.path.append("/opt/opsware/pylibs")
from pytwist import *
from pytwist.com.opsware.search import Filter

# Check for the command-line argument.
if len(sys.argv) < 2:
  print 'You must specify part of the hostname as the search target.'
  print "Example:" + sys.argv[0] + "   " + "opsware.com"
  sys.exit(2)

# Construct a search filter.
filter = Filter()
filter.expression = 'device_hostname *=* "%s"' % (sys.argv[1])

# Create a TwistServer object.
ts = twistserver.TwistServer()

# Get a reference to ServerService.
serverservice = ts.server.ServerService

# Perform the search, returning a tuple of references.
servers = serverservice.findServerRefs(filter)

if len(servers) < 1:
  print "No matching servers found"
  sys.exit(3)

# For each server found, get the server's value object (VO)
# and print some of the VO's attributes.
for server in servers:
   vo = serverservice.getServerVO(server)
   print "Name:" + vo.name
   print "   Management IP:" + vo.managementIP
```

```
   print "   OS Version:" + vo.osVersion
```

create_folder.py

このスクリプトは、createPathメソッドを呼び出して、/TestA/TestBという名前のフォルダーを作成します。createPathのpathパラメーターにはスラッシュは含まれません。pathの各文字列要素がフォルダー内のレベルを示します。次に、スクリプトはルートフォルダー直下のすべてのフォルダーの名前を取得して出力します。create_folder.pyスクリプトのリストを次に示します。

```
#!/opt/opsware/bin/python
# create_folder.py

# Create a folder in SA.

import sys
sys.path.append("/opt/opsware/pylibs")
from pytwist import *

# Create a TwistServer object.
ts = twistserver.TwistServer()

# Get a reference to FolderService.
folderservice = ts.folder.FolderService

# Get a reference to the root folder.
rootfolder = folderservice.getRoot()
# Construct the path of the new folder.
path = 'TestA', 'TestB'

# Create the folder /TestA/TestB relative to the root.
folderservice.createPath(rootfolder, path)

# Get the child folders of the root folder.
rootchildren = folderservice.getChildren(rootfolder,
'com.opsware.folder.FolderRef')

# Print the names of the child folders.
for child in rootchildren:
   vo = folderservice.getFolderVO(child)
   print vo.name
```

remediate_policy.py

このスクリプトは、ソフトウェアポリシーを作成し、サーバーにアタッチし、ポリシーを修復します。いくつかの名前はスクリプトに直接書き込まれています。プラットフォーム、サーバー、親フォルダーがそうです。オプションで、ポリシー名をコマンドラインに指定することもできます。これはスクリプトを何回も実行する場合に便利です。ソフトウェアポリシーのプラットフォームは、ポリシーに含まれるパッケージのOSに一致する必要があります。したがって、スクリプトに直接書き込まれているプラットフォーム名を変更する場合は、unitfilter.expression内の名前も変更する必要があります。

次のリストにはいくつか不要な改行があります。このコードをコピーした場合は、必ず不要な改行を削除してから実行してください。不要な改行がある個所には "NOTE" で始まるコメントが付いています。

```
#!/opt/opsware/bin/python
# remediate_policy.py

# Create, attach, and remediate a software policy.

import sys
sys.path.append("/opt/opsware/pylibs")
from pytwist import *
from pytwist.com.opsware.search import Filter
from pytwist.com.opsware.swmgmt import SoftwarePolicyVO

# Initialize the names used by this script.
foldername = 'TestB'
platformname = 'Windows 2003'
servername = 'd04.example.com'
# If a command-line argument is specified,
# use it as the policy name
if len(sys.argv) == 2:
  policyname = sys.argv[1]
else:
  policyname = 'TestPolicyA'

# Create a TwistServer object.
ts = twistserver.TwistServer()

# Get the references to the services used by this script.
folderservice = ts.folder.FolderService
swpolicyservice = ts.swmgmt.SoftwarePolicyService
serverservice = ts.server.ServerService
unitservice = ts.pkg.UnitService
platformservice = ts.device.PlatformService

# Search for the folder that will contain the policy.
folderfilter = Filter()
folderfilter.expression = 'FolderVO.name = ' + foldername
folderrefs = folderservice.findFolderRefs(folderfilter)

if len(folderrefs) == 1:
  parent = folderrefs[0]
elif len(folderrefs) < 1:
  print "No matching folders found."
  sys.exit(2)
else:
  print "Non-unique folder name:" + foldername
  sys.exit(3)

# Search for the reference to the platform "Windows Server 2003."
platformfilter = Filter()
platformfilter.objectType = 'platform'
doublequote = '\"'
# Because the platform name contains spaces,
```

```
# it's enclosed in double quotes
# NOTE:The following code line has a bad line break.
# The assignment statement should be on a single line.
platformfilter.expression = 'platform_name = ' + doublequote + platformname +
doublequote
platformrefs = platformservice.findPlatformRefs(platformfilter)

if len(platformrefs) == 0:
  print "No matching platforms found."
  sys.exit(4)

# Search for the references to some software packages.
unitfilter = Filter()
unitfilter.objectType = 'software_unit'
# NOTE:The following code line has a bad line break.
# The assignment statement should be on a single line.
unitfilter.expression = '((UnitVO.unitType = "MSI") & ( UnitVO.name contains "ismtool"
) & ( software_platform_name = "Windows 2003" ))'
unitrefs = unitservice.findUnitRefs(unitfilter)

# Create a value object for the new software policy.
vo = SoftwarePolicyVO()
vo.name = policyname
vo.folder = parent
vo.platforms = platformrefs
vo.softwarePolicyItems = unitrefs

# Create the software policy.
swpolicyvo = swpolicyservice.create(vo)

# Search by hostname for the reference to a managed server.
serverfilter = Filter()
serverfilter.objectType = 'server'
# NOTE:The following code line has a bad line break.
# The assignment statement should be on a single line.
serverfilter.expression = 'ServerVO.hostname = ' + servername
serverrefs = serverservice.findServerRefs(serverfilter)

if len(serverrefs) == 0:
  print "No matching servers found."
  sys.exit(5)

# Create an array that has a reference to the
# newly created policy.
swpolicyrefs = [1]
swpolicyrefs[0] = swpolicyvo.ref

# Attach the software policy to the server.
swpolicyservice.attachToPolicies(swpolicyrefs, serverrefs)

# Remediate the policy and the server.
# NOTE:The following code line has a bad line break.
# The assignment statement should be on a single line.
jobref = swpolicyservice.startRemediateNow(swpolicyrefs, serverrefs)
print 'The remediation job ID is %d' % jobref.id
```

Pytwistの詳細

ここでは、Pytwistに固有の動作と構文について説明します。

認証モード

Pytwistクライアントの認証モードは、クライアントからアクセスできるSAの機能とリソースに影響するため重要です。Pytwistクライアントは次のいずれかのモードで動作します。

- **認証済み**: クライアントは`TwistServer`オブジェクトに対して`authenticate(username, password)`メソッドを呼び出しています。`authenticate`メソッドを呼び出した後、クライアントは、SAクライアントにログオンするエンドユーザーと同様に、`username`パラメーターで指定されたSAユーザーとして認証されます。
- **非認証** : クライアントは `TwistServer.authenticate` メソッドを呼び出していません。管理対象サーバー上で、クライアントはエージェント証明書を制御するデバイスと同様に認証されます。カスタム拡張で使用した場合、非認証Pytwistクライアントはコマンドエンジン証明書にアクセスできる必要があります。カスタム拡張と証明書に関する詳細については、テクニカルサポート担当者にお問い合わせください。

TwistServerメソッドの構文

`TwistServer`メソッドは、クライアントからWebサービスデータアクセスエンジンへの接続を構成します(呼び出しの例については、Pytwistの例 (54ページ) を参照してください)。`TwistServer`のすべての引数はオプションです。表5に、引数のデフォルト値を示します。

表 5　　TwistServerメソッドの引数

引数	説明	デフォルト
`host`	接続先のホスト名。	twist
`port`	接続先のポート番号。	1032
`secure`	接続にhttpsを使用するかどうか。 使用可能な値: 1 (true) または0 (false)。	1
`ctx`	接続のSSLコンテキスト。	なし(認証モード (59ページ) も参照)。

`TwistServer`オブジェクトの作成時には、クライアントはサーバーとの接続を確立しません。したがって、接続の問題が存在する場合、クライアントが`authenticate`またはSA APIメソッドを呼び出すまで問題は明らかになりません。

エラー処理

`TwistServer.authenticate`メソッドまたはSA APIメソッドで問題が発生した場合、Python例外が発生します。これらの例外は、次の例に示すように`except`句で捕捉できます。

```
# Create the TwistServerobject.
ts = twistserver.TwistServer(localhost)
```

```
# Authenticate by passing an SA user name and password.
try:
  ts.authenticate('jdoe', 'secretpass')
except:
  print "Authentication failed."
  sys.exit(2)
```

Javaパッケージ名およびデータ型のPytwistへのマッピング

Pytwistインタフェースは Python用ですが、SA APIはJavaで作成されています。2つのプログラミング言語の間の違いのために、Pytwistクライアントはここに記すマッピングルールに従う必要があります。

SA APIドキュメントでは、Javaパッケージ名はcom.opswareで始まります。Pytwistでパッケージ名を指定する場合、次の例のように先頭にpytwistを挿入します。

```
from pytwist.com.opsware.compliance.sco import *
```

SA APIドキュメントでは、メソッドのパラメーターと戻り値がJavaデータ型で指定されています。表6に、PytwistでAPIメソッドを呼び出す場合のJavaデータ型からPythonへのマッピング方法を示します。

表 6　JavaからPythonへのデータ型のマッピング

SA APIでのJavaデータ型	pytwistでのPythonデータ型
Boolean	真の場合は整数1、偽の場合は整数0。
Object[] (オブジェクト配列)	APIメソッド呼び出しの入力パラメーターでは、オブジェクト配列はPythonのタプルまたは配列です。APIメソッド呼び出しからの出力では、オブジェクト配列はPythonタプルで返されます。
マップ	辞書
リスト	配列
Date	エポック (1970年1月1日午前0時) からのミリ秒数を表すlongデータ型。

第 4 章　自動化プラットフォーム拡張 (APX) の作成

この章では、自動化プラットフォーム拡張 (APX) の作成と管理の方法を説明します。これは一般には単に「拡張」と呼ばれます。APXは、シェルスクリプト、Python、Perl、PHPといったスクリプトベースのプログラミングツールを使用して、SAの機能を拡張し、SAと緊密に統合されたアプリケーションを作成するためのフレームワークを提供します。SAには次の2種類のAPXが用意されています。

- **Program APX** (**Script APX**とも呼ばれる) は、Global File System (OGFS) で実行され、すべてのOGFS機能を使用できます。一般的なプログラミング技法を使用してSA APIを使用し、コアの管理対象サーバーにアクセスして新しいカスタム機能を実装できます。たとえば、管理対象サーバーからBIOS情報を収集し、シェルコマンドを使用してカスタムフィールドの値を設定するAPXを作成することができます。詳細についてはProgram APX (63ページ) を参照してください。
- **Web APX**では、Webベースのアプリケーションを作成し、GET URLまたはPOST URLを使用してApache 2.xプロセスまたはCGI/PHPスクリプトを呼び出すことができます。Web APXには、画像などの静的なWebリソースを含めることも、CGIまたはPHPを使用して動的なコンテンツを生成することもできます。詳細についてはWeb APX (63ページ) を参照してください。

APXを使えば、管理対象環境に関するデータにアクセスし、そのデータをWebアプリケーション、スクリプト、プログラム、およびその他のアプリケーションと共有して処理することができます。APXの利点のいくつかを次に示します。

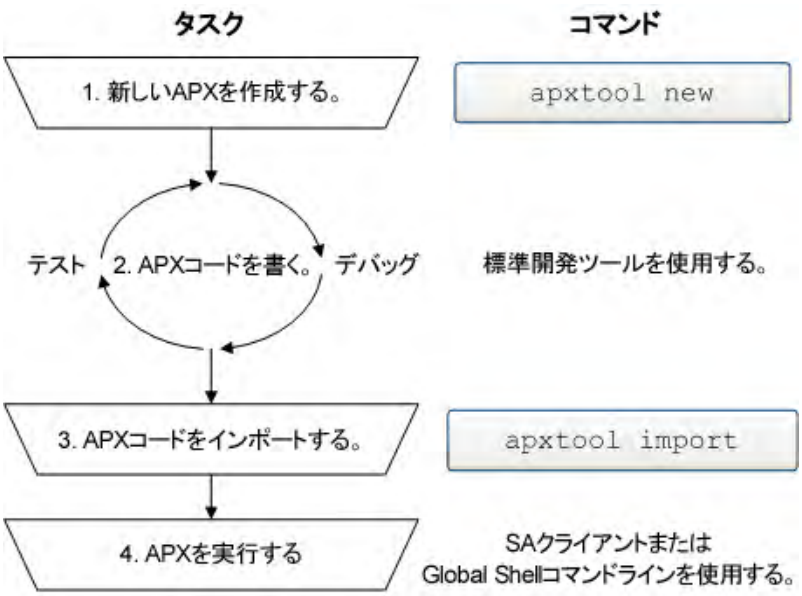
- SAライブラリにリストされ、SAクライアントから使用できます。
- バージョン管理によって一意に識別され、管理されます。
- SAのセキュリティモデルをフルに利用できるため、セキュリティを確保できます。APXでは、必要な場合、APXセッション中にセキュリティを確保しながら一時的にユーザーのアクセス権を通常のデフォルトからエスカレートすることができます。
- SAコア内部およびコア間に拡張可能です。
- サーバーに自動的にプッシュするようにスケジュールできます。
- 監査可能です。
- SAプラットフォームのアップグレードの影響を受けません。アップグレード後もAPXを作成し直す必要はありません。

APX拡張の使用に関する詳細については、『SAユーザーガイド: Server Automation』の「SAの拡張の実行」を参照してください。また、APX拡張はSA Global Shellからも実行できるので、『SAユーザーガイド: Server Automation』の「SA Global Shell」も参照してください。

APXの作成

次の図は、APXの作成の基本的な手順と、そこで使用するコマンドを示します。Web APXの作成方法のチュートリアルについては、チュートリアル: Webアプリケーション APXの作成 (83ページ) を参照してください。Program APXの作成方法のチュートリアルについては、チュートリアル: Program APXの作成 (89ページ) を参照してください。

図 3　　APXの作成



1　新しいAPXを作成するには、`apxtool new`コマンドを使用します。このコマンドはテンプレートファイルのセットを作成します。これらのファイルを編集して、独自のAPXを作成できます。

   `apxtool new`コマンドでは、オプションで新しいAPXを登録することもできます。APXを登録すると、そのAPXの名前がSAで予約されます。このステップでAPXを登録しない場合、この後のステップ3で`apxtool import`コマンドを使用して登録できます。

   詳細についてはapxtoolコマンド (70ページ) を参照してください。

2　APXテンプレートファイルを作成したら、`apxtool new`コマンドで作成されたテンプレートファイルを変更し、場合によっては独自のファイルを追加して、APXコードを開発します。APXコードをテストして、動作を確認することもできます。

3　APXコードをテストしたら、`apxtool import`コマンドを使用してSAにインポートします。

4　APXは、SAクライアントまたはGlobal Shellコマンドラインから実行します。

   — SAクライアントから: [**ライブラリ**] > [**タイプ別**] タブ > [**拡張子**] > [**プログラム**] を選択します。APXを選択します。[**アクション**] > [**実行**] メニューを選択します。

   — Global Shell コマンドラインから : SA クライアントから [**ツール**] > [**Global Shell**] メニューを選択して、Global Shellを開きます。次のコマンドを入力してAPXを実行します。
   </opsw/apx/bin/APX**名**>

   — 詳細については、『SAユーザーガイド: Server Automation』の「SAの拡張の実行」と『SAユーザーガイド: Server Automation』の「SA Global Shell」を参照してください。

VMware ESXiサーバーで動作するAPX拡張を作成する場合、APX拡張はWebサービスインタフェースを通じてリモートでESXiサーバーと通信する必要があります。VMware ESXiサーバーの詳細については、『SAユーザーガイド: Server Automation』の「仮想サーバーの管理」を参照してください。

Program APX

Program APXはScript APXとも呼ばれ、シェルコマンドに似ていて、OGFSサーバースクリプトとして実装されます。OGFS コマンドラインから呼び出し、入力引数はSTDIN またはコマンドライン引数を通じて渡します。出力はSTDOUTおよびSTDERRに行われます。

Program APXはGlobal Shell (OGSH) セッション内部で実行され、APXを呼び出したユーザーがアクセス権を持つすべてのOGSH機能にアクセスできます。これには、`rosh`、CLI、OGFSなどが含まれます。Program APXの作成には、シェルスクリプト、Python、Perlなど、任意のスクリプトベースのツールが使用できます。

Program APXはOGSHコマンドプロンプトから呼び出すことができます。Program APXは通常は同期的に実行されます。すなわち、Program APXが終了するまでシェルプロンプトは戻りません。APXは、twister またはOGFSの定期的ジョブとしてスケジュールすることはできません。

Program APXが存在する場所は、OGFSディレクトリ`/opsw/apx/bin`です。

対話型OGSHセッション中には、`/opsw/apx/bin`に存在するProgram APXのうち、ユーザーが実行アクセス権を持つものだけがユーザーに見えます。ユーザーが実行アクセス権を持たないProgram APXを呼び出そうとすると、シェルから`File Not Found`エラーが返されます。

Program APXは、他のWeb APXまたはProgram APXから呼び出すこともできます。たとえば、Web APXのCGIプログラムまたはPHPスクリプトからProgram APXを呼び出すことができます。

Web APX

Web APXは、CGIプログラムまたはPHPスクリプトで実装されます。これらのCGIプログラムやPHPスクリプトは、ユーザー固有のOGSHセッション内部で実行されます。`rosh`、SA API、CLIなどのSA機能と、その他OGSHセッションから使用可能な任意のコマンドにアクセスできます。Web APXは、PHPモジュールを備えた付属のApache Webサーバーで実行されます。

Web APXにアクセスするには2つの方法があります。1つはInternet ExplorerやFirefoxなどのスタンドアロンのWebブラウザーからアクセスする方法、もう1つはSAクライアントからアクセスする方法です。Microsoft ActiveXはサポートされません。

スタンドアロンのWebブラウザーから初めてWeb APXを呼び出した場合、ログインダイアログが表示され、SAのユーザー資格情報を認証する必要があります。SAクライアントからWeb APXを呼び出した場合は、改めてログインする必要はありません。Web APXを使用すれば、カスタムカスタマーアプリケーションのユーザーインタフェースを構築できます。

Windows Server 2003、2008、2012 でセキュリティ強化の構成がオンになっているときに、Microsoft Internet Explorer バージョン6および7からAPXを呼び出す場合は、Internet Explorerの信頼済みサイトのリストにSA Webクライアント URLをあらかじめ追加しておく必要があります。

APXユーザーの役割

APXユーザーには、表7に示す3つの一般的な役割があります。

表 7　　APXユーザーの役割

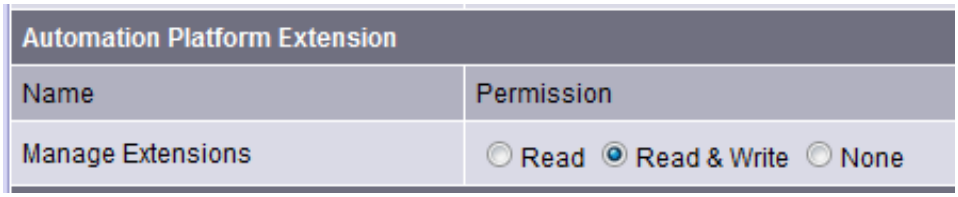
ユーザーの役割	説明
エンドユーザー	APXを実行します。このユーザーは通常、APXを変更したりその内容を見たりするアクセス権はありません。
APX開発者	APXを作成し、公開します。このクラスのユーザーは、APXをインポート/エクスポートしたり、APXの内容を変更したりできます。
APX管理者	ユーザーがどのAPXを実行できるかを決定します。これらのユーザーは、フォルダーのアクセス権を管理することにより、APXの実行アクセス権を割り当てます。APX管理者はAPX自体を変更するアクセス権は持たない場合がありますが、どのAPXを実行可能にするかを判断するためにAPX内容を表示するアクセス権を持つ可能性があります。

APXのアクセス権

APXを使用するには、SAクライアントの機能のアクセス権である**Manage Extensions**が必要です。ユーザーグループには、次のいずれかのアクセス権を与えることができます。

- Manage Extensions: Read
- Manage Extensions: Read & Write
- Manage Extensions: None

図 4　　APX機能のアクセス権



これらの機能のアクセス権は、APX開発者および管理者のみに当てはまります。APXを実行するだけのユーザーには当てはまりません。

- **Read**: このアクセス権は、APXのソースの内容を表示したり、APXソースアーカイブをエクスポート (ダウンロード) したりする権限を付与します。
- **Read & Write**: このアクセス権は、APXの内容の読み取りに加えて変更の権限を付与します。
- **None**: このアクセス権は、APXソースへのすべてのアクセスを禁止します。

SAクライアント機能の**Manage Extensions**アクセス権に加えて、フォルダーのアクセス権 (リスト、読み取り、書き込み、実行) を使用して、ユーザーがどのAPXにアクセスできるかを決定する必要があります。

表 8　APXのアクセス権

アクセス権	説明
List	システムのAPXをリストするアクセス権。
Read	APXの内容を表示するアクセス権。
Write	APXの内容の変更と、APXのインポート/エクスポートのアクセス権。
Execute	APXの実行と、APXのプロパティの表示のアクセス権。

表9のマトリクスは、拡張の管理機能のアクセス権とフォルダーのアクセス権の組み合わせに基づいて、アクセス権がどのように決定されるかを示します。

表 9　APXアクセス権のマトリクス

フォルダーのアクセス権:	Manage Extensions Permission:		
	Read	**Read & Write**	**None**
List	APXのリスト	APXのリスト	APXのリスト
Read	APXのエクスポート	APXのエクスポート	APXのリスト
Write	APXのエクスポート	APXのインポート/エクスポート	APXのリスト
Execute	APXの実行	APXの実行	APXの実行

他のSA機能と同様に、APXへのユーザーのアクセスを許可して、どの管理対象サーバーまたはポリシーにユーザーがAPXを適用できるかを指定できます。

ユーザーが実行アクセス権のないWeb APXにアクセスしようとすると、Webブラウザーは`HTTP 403 Forbidden`リターンコードを受け取ります。

SAのアクセス権の詳細については、『SA 管理ガイド』を参照してください。

アクセス権のエスカレーション

APXを実行する際に、ユーザーはSAで許可されているリソースと操作だけにアクセスする権限があります。しかし、場合によっては、APXの実行中に、SAの権限を超えるエスカレートされたアクセス権をユーザーに一時的に付与することが必要になります。APXの実行中に、デフォルトのSAの権限を超える特定の権限を一時的にユーザーに付与することができます。アクセス権のエスカレーションは、APXを実行するユーザーに対しては透過的です。

たとえば、BIOS情報を収集するアプリケーションを管理対象サーバーに対して実行する場合に、ユーザーがそのためのアクセス権を持っていない可能性があります。BIOS情報収集アプリケーションの実行に必要な権限を持たないユーザーのために、必要な権限をそのユーザーに一時的に付与するAPXを作成することができます。ユーザーの権限は、APXの実行が終了するとデフォルトに戻ります。

権限のエスカレーションは、`apx.perm`ファイルで指定されます。詳細については、APXアクセス権エスカレーション構成ファイル - apx.perm (80ページ) を参照してください。

APXの構造

APXは次の属性を持ちます。

- APXタイプ: Program APX (Script APXとも呼ばれる) またはWeb APX。
- APXの一意の名前: これはAPXの完全な名前で、一意である必要があります。たとえば、com.hp.sa.RestartMyAppのようになります。
- APXの表示名: これは通常、APXの一意の名前よりも短い名前です。たとえば、RestartMyAppのようになります。
- APXバージョン: APXの複数のバージョンを維持するには、バージョン文字列を設定するか、SAに自動的にバージョンを管理させます。APXバージョンは、バージョン1、2、3などの単純な数字でも、任意の英数字文字列でもかまいません。

  詳細については、APXのSAへのインポート - apxtool import (74ページ) とAPXの現在のバージョンの設定 - apxtool setcurrent (77ページ) を参照してください。

ファイル構造

SAから見ると、APXは単にAPXタイプ (Program APXまたはWeb APX) の規約に従うファイルとディレクトリの集合です。これらの規約は、APXランタイムがAPXを正しく実行するために必要です。たとえば、Web APXには`index.html`ファイルまたは`index.php`ファイルが必要です。Program APXには、APXと同じ名前のシェルコマンドが必要です。

APXのファイルに関する詳細については、APXファイル (79ページ) を参照してください。

OGFS統合

APXインフラストラクチャーは、OGFSを利用して、ユーザーセッションを管理し、SAファイルシステム内のAPXのさまざまな部分を公開しています。ここでは、APXがOGFSとその各種アプリケーションにどのように統合されるかを説明します。

APX実行可能ディレクトリ

Program APXは、Global Shell (OGSH) の実行可能プログラムとして扱われます。これらのAPXは、OGSHの実行可能コマンドとして公開されます。これにより、シェルのユーザーはシェルコマンドと同じ方法でAPXを呼び出すことができます。

APXの実行可能ディレクトリは次の形式を取ります。

`/opsw/apx/bin/{APX名}`

ここで、`APX名`はAPXの名前です。`/opsw/apx/bin/{APX名}`で`APX名`を実行すると、`APX名`の現在のバージョンが呼び出されます。

APXランタイムディレクトリ

APXランタイムディレクトリは、APXの実行をサポートするためにAPXランタイムによって使用されます。APXランタイムディレクトリは、APXソースへのアクセス権を持つ必要があります。また、開発者権限を持ち、APXへの読み取りアクセス権を持つユーザーも、APXにアクセスすることができます。APXランタイムディレクトリは、Global ShellのAPX開発者以外からは使用できません。

APXランタイムディレクトリは、APXの現在のバージョンのソースを参照します。これは次の形式を取ります。

/opsw/apx/runtime/{APX**タイプ**}/{APX**名**}

ここで、APX**タイプ**はscriptまたはwebです。

APXインタフェース - APX拡張のカテゴリの定義

APXインタフェースを使用すると、APXの名前付きのカテゴリを作成し、特定のカテゴリに属するすべてのAPXを知ることができます。インタフェースとは、カテゴリの名前です。たとえば、特定の入力パラメーターの組を取り、特定のタイプの出力データを生成するAPXのカテゴリを作成することができます。あるいは、特定の操作の組を実行するAPXのカテゴリを作成することもできます。

さらに、特定のカテゴリのすべてのAPXの名前を取得して、それらを実行するAPXまたは外部アプリケーションを作成することもできます。あるいは、APXまたはアプリケーションは単に特定のカテゴリのAPXのリストを表示して、どれを実行するかをユーザーに選択させることもできます。

APXインタフェースとは、APXの呼び出し元とAPXの間の非公式の規約を定義する名前です。

- **インタフェース名を定義する**APXは、その名前のAPXカテゴリを作成します。
- **インタフェースを実装する**APXは、自分自身をそのカテゴリのAPXと宣言します。

インタフェースの例

SAには、RightClickToRunという名前のインタフェースが用意されています。このインタフェースが定義するカテゴリのAPXは、1つまたは複数のデバイスを入力パラメーターに取り、これらのデバイスに対して実行されます。さらに、SAクライアントはこのインタフェースを実装しているすべてのAPXを [**アクション**] > [**拡張の実行**] メニューに表示します。これにより、ユーザーは1つまたは複数のデバイスを選択して、これらのAPXを選択したデバイスに対して実行することができます。このインタフェースの詳細については、RightClickToRunインタフェース (69ページ) を参照してください。

インタフェースの定義

APXインタフェースは、APXのカテゴリの名前を定義します。インタフェースを実装するすべてのAPXはそのカテゴリに属し、そのインタフェースの規則に従う必要があります。新しいカテゴリを作成するには、APXにインタフェースを「定義」させます。

APXにインタフェースを定義させるには、次の手順を実行します。

1. `apxtool new`コマンドでAPXを作成します。このコマンドの詳細については、新しいAPXの作成 - apxtool new (71ページ) を参照してください。
2. 新しいAPXのファイルを探し、`interfaces`という名前のファイルをテキストエディターで開きます。interfacesファイルは、APXディレクトリ下のAPX-INFディレクトリにあります。
3. `interfaces`ファイルの末尾に、次の3行を追加します。
   - ファイル内のインタフェースセクションの名前。これはインタフェースの一意の名前です。
   - インタフェースの表示名。
   - インタフェースの説明。

   次の例は、com.hp.sa.MyNewInterfaceという名前のインタフェースのインタフェースセクション名、表示名、説明を示します。

```
[com.hp.sa.MyNewInterface]
name=MyNewInterface
description=This is a simple interface for testing purposes.
```

4 変更を保存し、ファイルを閉じます。

5 変更したAPXをapxtool importコマンドでSAにインポートします。このコマンドの詳細については、APXのSAへのインポート - apxtool import (74ページ) を参照してください。

既存のAPXをインタフェースを定義するようにアップグレードする場合は、interfacesファイルを作成し、上記の手順でインタフェースを追加する必要があります。

インタフェースの実装

APXインタフェースは、インタフェースの規則に従うAPXのカテゴリを指定します。APXがカテゴリに属することを指定するには、APXにインタフェースを「実装」させます。APXにインタフェースを実装させるには、次の手順を実行します。

1 apxtool newコマンドでAPXを作成します。このコマンドの詳細については、新しいAPXの作成 - apxtool new (71ページ) を参照してください。

2 新しいAPXのファイルを探し、apx.cfgという名前のファイルをテキストエディターで開きます。

3 apx.cfgファイルでImplementingセクションを探します。このセクションには、APXが実装するインタフェースを指定する方法が簡単に説明されています。

4 apx.cfgファイルで次の行を見つけます。

```
[Implementing]
interfaces=
```

5 interfaces=行を変更して、インタフェースの名前を行の末尾に追加します。たとえば、APXがcom.hp.sa.MyNewInterfaceという名前のインタフェースを実装する場合は、apx.cfgファイルに次の行が含まれるようにします。

```
[Implementing]
interfaces=com.hp.sa.MyNewInterface
```

複数のインタフェースを実装する場合、次のようにコロンで区切ってinterfaces行に追加します。

```
[Implementing]
interfaces=com.hp.sa.MyNewInterface:com.hp.sa.AnotherInterface
```

6 変更を保存し、apx.cfgファイルを閉じます。

7 変更したAPXをapxtool importコマンドでSAにインポートします。このコマンドの詳細については、APXのSAへのインポート - apxtool import (74ページ) を参照してください。

SAクライアントまたはapxtool queryコマンドでAPXを表示する際に、実装されたインタフェースを見るには、APXの現在のバージョンを設定する必要があります。詳細については、APXの現在のバージョンの設定 - apxtool setcurrent (77ページ) を参照してください。

既存のAPXをインタフェースを使用するようにアップグレードする場合は、既存のapx.cfgファイルに上記の手順でインタフェースを追加する必要があります。

RightClickToRunインタフェース

SAには、APXで使用できるcom.hp.client.server.RightClickToRunという名前のインタフェースが用意されています。このインタフェースが使用できるのはProgram APXだけで、Web APXでは使用できません。このインタフェースは、APXに次のことを実行させたい場合に使用します。

- APXへの入力パラメーターとして1つまたは複数のデバイスを取ります。このインタフェースを実装するAPXは、入力引数として-d <**デバイス**ID>を取る必要があります。
- [**アクション**] > [**拡張の実行**] >[**拡張の選択...**] ウィンドウに表示されます。
- SAクライアントの [**アクション**] > [**拡張の実行**] メニューに表示されます。APXがこのメニューに表示されるのは、[**アクション**] > [拡張の実行] > [**拡張の選択...**] メニューを使用して1回実行された後です。

[**アクション**] > [**拡張の実行**] メニューからAPXを実行するには、ユーザーはAPXに対する実行アクセス権を持つ必要があります。ユーザーが実行アクセス権を持たないAPXは、このメニュー項目に表示されません。アクセス権の詳細については、『SA 管理ガイド』を参照してください。

RightClickToRunインタフェースにより、ユーザーはSAクライアントで1つまたは複数のデバイスを選択して、これらのデバイスに対してAPXを実行できます。

[**アクション**] > [**拡張の実行**] メニュー項目を選択すると、<:v "SA Client" 4>はcom.hp.client.server.RightClickToRunを実装しているすべてのProgram APXを表示します。ユーザーがAPXを選択すると、選択したすべてのサーバーに対してAPXが実行されます。APXは選択したそれぞれのサーバーに対して1回ずつ呼び出されます。

APXにこのインタフェースを実装させる方法の詳細については、インタフェースの実装 (68ページ) を参照してください。このインタフェースを実装しているAPXの使用に関する詳細については、『SAユーザーガイド: Server Automation』の「SAの拡張の実行」を参照してください。

インタフェースAPIの使用

SA APIを使用して、自作のアプリケーションをSAおよびAPXと統合できます。特定のインタフェースを実装しているすべてのAPXをアプリケーションから知るには、APIのcom.opsware.apxというパッケージにあるAPXInterfaceServiceというインタフェースを使用します。SA APIの使用法の詳細については、第1章「APIドキュメントとTwister」(21ページ) を参照してください。を参照してください。

apxtoolコマンド

apxtoolコマンドをOGFSセッションで使用すると、APXの作成と管理を行うことができます。apxtoolコマンドは、Global Shellのディレクトリ/opsw/bin/apxtoolで使用できます。

apxtoolを使用してWeb APXを作成する方法のチュートリアルについては、チュートリアル: WebアプリケーションAPXの作成 (83ページ) を参照してください。

apxtoolの構文

APXツールを呼び出すには、OGFSコマンドラインに次のように入力します。

```
apxtool [-h | --help] {機能} arguments
```

APXツールでサポートされるコマンドと引数の一覧を見るには、OGSHコマンドラインからapxtoolを引数なしで実行します。

APXツールがサポートする主な機能を次に示します。

表 10　APXツールの機能

機能	使用法
new	新しいAPXソースディレクトリと、新しいテンプレートファイルのセットをOGFSに作成します。オプションで、APXをSAに登録します。登録すると、APX IDが割り当てられ、SAを使用している (適切なアクセス権を持つ) 他のユーザーからAPXの名前が使用できるようになります。詳細については新しいAPXの作成 - apxtool new (71ページ) を参照してください。
import	APXファイルをSAライブラリにインポートし、APXの新しいバージョンを作成します。オプションで、APXをSAに登録します。登録すると、APX IDが割り当てられ、SAを使用している (適切なアクセス権を持つ) 他のユーザーからAPXの名前が使用できるようになります。詳細についてはAPXのSAへのインポート - apxtool import (74ページ) を参照してください。
setcurrent	SAライブラリ内のAPXの現在のバージョンを設定します。SAではAPXの複数のバージョンを持つことができますが、実行できるのは現在のバージョンだけです。詳細についてはAPXの現在のバージョンの設定 - apxtool setcurrent (77ページ) を参照してください。
query	APXに関する情報を表示します。詳細についてはAPX情報のクエリ - apxtool query (75ページ) を参照してください。
export	APXのすべてのファイルをSAライブラリから別のファイルセットにコピーします。
delete	APXをSAライブラリから削除します。

短いコマンドオプションと長いコマンドオプションの使用

apxtoolコマンドのオプションのほとんどには、短い形式と長い形式があります。

- 短い形式は1個のハイフンと1文字からなります。例としては、-tや-vがあります。
- 長い形式は、2個のハイフンと1語からなります。例としては、"--type" や "--view "があります。

一部のオプションでは、オプションの後に引数が必要です。例としては、"-t webapp" や "-t details" があります。引数の指定には4つの形式があり、どれも効果は同じです。たとえば、次のコマンドはすべて等価であり、同じ結果になります。

```
apxtool query -t webapp
```

```
apxtool query -twebapp
apxtool query -tw
apxtool query --type webapp
apxtool query --type=webapp
```

一部のオプションでは、オプション引数の識別に必要な最小限の文字を入力するだけで済みます。たとえば、query機能の場合、--viewオプションには引数list、details、versionsが必要です。次のコマンドは同じ結果になります。

```
apxtool query --view=details
apxtool query --view=d
apxtool query -vdetails
apxtool query -vd
```

新しいAPXの作成 - apxtool new

APXツールを使用して、新しいAPXを作成し、オプションでAPXの名前をSAに登録することができます。このコマンドはAPXテンプレートファイルのセットを作成します。これらのファイルは変更できます。APXを構成するファイルに関する詳細については、APXファイル (79ページ) を参照してください。

使用法

apxtool new [**オプション**] {**ソースディレクトリ**}

ここで、**ソースディレクトリ**引数は、新しいAPXのテンプレートファイルが作成されるディレクトリを指定します。この引数を省略した場合、テンプレートファイルは現在のディレクトリに作成されます。

表11に、新しいAPXを作成するためのオプションの一覧を示します。

表 11　apxtool newのオプション

オプション	使用法
-h、--help	このヘルプメッセージを表示して終了します。
-t <**タイプ**> --type=<**タイプ**>	(**必須**) APXタイプ。有効な値はscriptまたはwebappです。たとえば、-tsはScript APX、-twはWeb APXを表します(Script APXはProgram APXとも呼ばれます)。
-u <**一意の名前**> --uniquename=<**一意の名前**>	(**必須**) APXの一意の名前。一意の名前は、ファイルシステムの形式に従ったドット区切りの名前です。ドットが少なくとも1つ含まれる必要があります。使用可能な文字は[a-zA-Z0-9_.]です。 **例**: com.hp.sa.security.scan_ports

表 11　apxtool newのオプション (続き)

オプション	使用法
-n <**名前**> --name=<**名前**>	(**オプション**) フォルダー内のAPXの表示名。名前を指定せず、一意の名前を指定した場合は、APXの一意の名前の最後の部分が表示名として使用されます。この名前は指定したフォルダー内で一意である必要があります。 たとえば、一意の名前がcom.hp.sa.MyWebExtの場合、デフォルトの表示名はMyWebExtとなります。
-d <**説明**> --description=<**説明**>	(**必須**) APXの簡単な説明。説明が拡張子.txtのファイル名の場合、そのファイルはテキストファイルと見なされ、その内容がAPXの説明として用いられます。
-r --register	(**オプション**) APXの名前をシステムに登録します。このオプションを指定した場合、-fまたは--folderも指定する必要があります。 apxtool newに-rと-fを指定しなかった場合は、apxtool importで-fを使用する必要があります。
-f <**パス**> --folder=<**パス**>	(**オプション**) APXが登録されるSAフォルダーのパス。これは、特定のフォルダーを一意に識別するものであれば、フルパス、部分パス、絶対パス、相対パスのどれでもかまいません。このオプションが必要なのは、-rまたは--registerを使用した場合だけです。 apxtool newに-rと-fを指定しなかった場合は、apxtool importで-fを使用する必要があります。
-Q、--quiet	(**オプション**) 出力メッセージを抑制します。
-F、--force	(**オプション**) 確認プロンプトを抑制します。

APXの削除 - apxtool delete

APXツールを使用して、既存のAPXをSAライブラリから削除できます。

使用法

apxtool delete [**オプション**]

表12に、APXを削除するためのオプションの一覧を示します。

表 12　apxtool deleteのオプション

オプション	使用法
-h --help	このヘルプメッセージを表示して終了します。
-t <**タイプ**> --type=<**タイプ**>	(**必須**) APXタイプ。有効な値はscriptまたはwebappです。たとえば、-tsはスクリプトを表します。
--id=<APX ID>	(**オプション**) 目的のAPXのオブジェクトID。
-u <**一意の名前**> --uniquename=<**一意の名前**>	(**オプション**) APXの一意の名前。一意の名前は、ファイルシステムの形式に従ったドット区切りの名前です。ドットが少なくとも1つ含まれる必要があります。使用可能な文字は[a-zA-Z0-9_.] です。 **例**: com.hp.sa.security.scan_ports
-n <**名前**>、--name=<**名前**>	(**オプション**) フォルダー内のAPXの表示名。
-f <**パス**>、--folder=<**パス**>	(**オプション**) SAフォルダーのパス。これは、特定のフォルダーを一意に識別するものであれば、フルパス、部分パス、絶対パス、相対パスのどれでもかまいません。
-Q、--quiet	(**オプション**) 出力メッセージを抑制します。
-F、--force	(**オプション**) 確認プロンプトを抑制します。

SAからのAPXのエクスポート - apxtool export

APXツールを使用して、APXをエクスポートできます。エクスポートとは、APXソースアーカイブファイルの特定のバージョンをダウンロードして、ファイルをディレクトリまたは.zipアーカイブファイルに格納することです。

使用法

apxtool export [**オプション**] {**ターゲットディレクトリ**}

ここで、引数**ターゲットディレクトリ**はディレクトリであり、--archiveオプションが指定されているかどうかに応じて、APXソースアーカイブファイルがコピーされるか、APXソースアーカイブの内容が展開されるディレクトリを表します。省略した場合、現在のディレクトリが使用されます。

表13に、APXをエクスポートするためのオプションの一覧を示します。

表 13　apxtool exportのオプション

オプション	使用法
-h、--help	このヘルプメッセージを表示して終了します。
-t <**タイプ**>、--type=<**タイプ**>	(**必須**) APXタイプ。有効な値はscriptまたはwebappです。たとえば、-tsはスクリプトを表します。
--id=<APX ID>	(**オプション**) 目的のAPXのオブジェクトID。
-u <**一意の名前**> --uniquename=<**一意の名前**>	(**オプション**) APXの一意の名前。一意の名前は、ファイルシステムの形式に従ったドット区切りの名前です。ドットが少なくとも1つ含まれる必要があります。使用可能な文字は[a-zA-Z0-9_.] です。 **例**: com.hp.sa.security.scan_ports
-n <**名前**>、--name=<**名前**>	(**オプション**) フォルダー内のAPXの表示名。
-f <**パス**>、--folder=<**パス**>	(**オプション**) SA フォルダーのパス。これは、特定のフォルダーを一意に識別するものであれば、フルパス、部分パス、絶対パス、相対パスのどれでもかまいません。
-v <**バージョン文字列**>、--version=<**バージョン文字列**>	(**オプション**) このオプションは、ダウンロードするAPXバージョンを指定します。省略した場合、現在のバージョンがダウンロードされます。
-a、--archive	指定した場合、元のソースアーカイブのAPXソースをZIPまたはJARファイルでエクスポートします。
-Q、--quiet	(**オプション**) 出力メッセージを抑制します。
-F、--force	(**オプション**) 確認プロンプトを抑制します。

APXのSAへのインポート - apxtool import

APX ツールを使用して、APX をインポートできます。インポートとは、APX の新しいバージョンを公開し、オプションでそのバージョンを現在のバージョンに設定することです。APXがまだ登録されていない場合は、このコマンドはAPXの登録も行います。

実行できるのは現在のバージョンのAPXだけです。現在のバージョンを選択していない場合、APXは実行できません。現在のバージョンを設定するには、apxtool importまたはapxtool setcurrentを使用します。詳細についてはAPXの現在のバージョンの設定 - apxtool setcurrent (77ページ) を参照してください。

使用法

apxtool import [**オプション**] {APX**ソース**}

ここで、APX**ソース**は拡張子.zipまたは.jarのAPXソースアーカイブか、公開するAPXファイルを含むディレクトリです。APXソースは相対パスでも絶対パスでもかまいません。省略した場合、現在のディレクトリが使用されます。指定したディレクトリまたはアーカイブファイルには、ディレクトリAPX-INFが含まれる必要があります。

表14に、APXをインポートする際に使用できるオプションの一覧を示します。

表 14　apxtool importのオプション

オプション	使用法
-h、--help	このヘルプメッセージを表示して終了します。
-c、--setcurrent	指定した場合、新しく公開したバージョンをAPXの現在のバージョンに設定します。
--version=<**バージョン文字列**>	このAPXの新しいバージョン。**バージョン文字列**がapx.cfgにすでに指定されている場合は、このオプションは使用できません。バージョンを指定しない場合、自動的にバージョンが割り当てられます。
-f <**パス**>、--folder=<**パス**>	(**オプション**) SAフォルダーのパス。これは、特定のフォルダーを一意に識別するものであれば、フルパス、部分パス、絶対パス、相対パスのどれでもかまいません。 apxtool newに-rと-fを指定しなかった場合は、apxtool importで-rを使用する必要があります。
-Q、--quiet	(**オプション**) 出力メッセージを抑制します。
-F、--force	(**オプション**) 確認プロンプトを抑制します。

APX情報のクエリ - apxtool query

APXツールを使用して、APX情報を取得して表示できます。追加のオプションを指定することで、結果のAPXを制限できます。同じオプションを複数回指定した場合、論理OR式で結合されます。一致する結果が見つからない場合、このコマンドは終了コード100を返します。

使用法

```
apxtool query [オプション]
```

表15に、APX情報のクエリの際に使用できるオプションの一覧を示します。

表 15　apxtool queryのオプション

オプション	使用法
-h、--help	このヘルプメッセージを表示して終了します。
-v <**表示**>, --view=<**表示**>	(**オプション**) クエリ結果の事前定義された表示の1つを選択します。選択肢としては、list (デフォルト)、details、versionsがあります。 -v listは、APXの基本情報を1行で表したものを表形式で表示します。 -v detailsは、APXの基本情報を複数行で表したものです。 -v versionsは、APXのすべてのバージョンをリストします。表示タイプの指定には、十分な数の文字だけを指定すれば済みます。たとえば、-vdは-v detailsと同じです。レイアウトとしてversionsを選択した場合、クエリの結果は1個のAPXオブジェクトでなければなりません。
-t <**タイプ**>、--type=<**タイプ**>	(**オプション**) 表示するAPXのタイプを指定します。有効な値はscriptまたはwebappまたはinterfaceです。デフォルトでは、すべてのタイプが表示されます。 -t scriptはすべてのScript APXを表示します。 -t webappはすべてのWeb APXを表示します。 -t interfaceは1つまたは複数のインタフェースを定義するすべてのAPXを表示します。 たとえば、apxtool query -tsはすべてのScript APXを表示します。
--id=<APX ID>	(**オプション**) 目的のAPXのオブジェクトID。
-u <**一意の名前**> --uniquename=<**一意の名前**>	(**オプション**) APXの一意の名前。一意の名前は、ファイルシステムの形式に従ったドット区切りの名前です。ドットが少なくとも1つ含まれる必要があります。使用可能な文字は[a-zA-Z0-9_.] です。 **例**: com.hp.sa.security.scan_ports
-n <**名前**>、--name=<**名前**>	(**オプション**) フォルダー内のAPXの表示名。
-f <**パス**>、--folder=<**パス**>	(**オプション**) SAフォルダーのパス。これは、特定のフォルダーを一意に識別するものであれば、フルパス、部分パス、絶対パス、相対パスのどれでもかまいません。
--current	(**オプション**) 指定した場合、現在のバージョンが設定されているAPXオブジェクトだけをクエリの対象とします。

表 15　apxtool queryのオプション (続き)

オプション	使用法
`--format=<`**フォーマット文字列**`>`	(**オプション**) これは高度なオプションであり、APXリストのカスタム表示形式を指定します。 **フォーマット文字列**は文字列で、中に埋め込まれたタグ名が表示時に値に置き換えられます。タグ名の形式は`%(`**タグ名**`)`です。 サポートされるタグ名の一覧を表示するには、フォーマット文字列として`_ "_show_tags__"` を使用します。
`--csv`	(**オプション**) 出力をカンマ区切り値形式で表示します。`--format`オプションを指定した場合は無視されます。
`-Q、--quiet`	(**オプション**) 重要でない出力メッセージを抑制します。

APXの現在のバージョンの設定 - apxtool setcurrent

APXツールを使用して、APXバージョンを現在のバージョンとして設定できます。

実行できるのは現在のバージョンのAPXだけです。現在のバージョンを選択していない場合、APXは実行できません。現在のバージョンを設定するには、`apxtool import`または`apxtool setcurrent`を使用します。詳細についてはAPXのSAへのインポート - apxtool import (74ページ) を参照してください。

使用法

`apxtool setcurrent [`**オプション**`] {`**バージョン文字列**`}`

ここで、引数**バージョン文字列**は、APXの既存のバージョンを一意に識別するために必要です。

表16に、APXバージョンを設定する際に使用できるオプションの一覧を示します。

表 16　apxtool setcurrentのオプション

オプション	使用法
`-h、--help`	このヘルプメッセージを表示して終了します。
`-t <`**タイプ**`>、--type=<`**タイプ**`>`	(**必須**) APXタイプ。有効な値は`script`、`webapp`です。たとえば、`-ts`はスクリプトを表します。
`--id=<APX ID>`	(**オプション**) 目的のAPXのオブジェクトID。

表 16　apxtool setcurrentのオプション (続き)

オプション	使用法
`-u <`**一意の名前**`>` `--uniquename=<`**一意の名前**`>`	(**オプション**) APXの一意の名前。一意の名前は、ファイルシステムの形式に従ったドット区切りの名前です。ドットが少なくとも1つ含まれる必要があります。有効な文字は `[a-zA-Z0-9_.]` です。 **例**: `com.hp.sa.security.scan_ports`
`-n <`**名前**`>`、`--name=<`**名前**`>`	(**オプション**) フォルダー内のAPXの表示名。
`-f <`**パス**`>`、`--folder=<`**パス**`>`	(**オプション**) SA フォルダーのパス。これは、特定のフォルダーを一意に識別するものであれば、フルパス、部分パス、絶対パス、相対パスのどれでもかまいません。
`-Q`、`--quiet`	(**オプション**) 出力メッセージを抑制します。
`-F`、`--force`	(**オプション**) 確認プロンプトを抑制します。

エラー処理

APXツールコマンドは標準のPOSIX規則に準拠しており、成功の場合は0、他のエラーの場合は0以外の値を返します。APXツールは、通常の出力をSTDOUTに、エラーと警告をSTDERRに送信します。エラーが発生した場合、APXツールは通常STDERRに説明のメッセージを出力します。

エラー条件は通常、表17に示すように分類されます。

表 17　APXツールのエラー条件

リターンコード	説明
0	成功
1	構文または使用法のエラー
2	アクセス権関連のエラー
3	ユーザーが操作をキャンセル
4	実行時エラー

文書化されていない他の終了コードが存在する可能性もあります。保証されるのは、終了コードが0の場合、コマンドは操作を正常に完了したということだけです。

APXファイル

ここでは、apxtool newコマンドを実行したときに作成されるテンプレートファイルについて説明します。次の表にファイルの一覧を示します。この後で、いくつかのファイルについて詳細に説明します。

表 18　APXファイル

ファイル名	説明
apx.cfg	APX構成ファイル。APXを完全に記述するメタデータを含みます。詳細についてはAPX構成ファイル - apx.cfg (79ページ) を参照してください。
apx.perm	APXアクセス権ファイル。アクセス権のエスカレーションルールを指定します。詳細についてはAPXアクセス権エスカレーション構成ファイル - apx.perm (80ページ) を参照してください。
description.txt	APXの説明テキスト。apxtool new -dオプションで指定されます。詳細については新しいAPXの作成 - apxtool new (71ページ) を参照してください。
interfaces	APXインタフェース定義ファイル。APXが定義または実装するインタフェースを指定します。詳細についてはAPXインタフェース - APX拡張のカテゴリの定義 (67ページ) を参照してください。
usage.txt	APXの使用法に関する説明テキスト。
run.sh	Program APXの場合のみ、このファイルにはAPXの実行可能コードが含まれます。このファイルには、Program APXの機能が記述されています。例については、チュートリアル: Program APXの作成 (89ページ) を参照してください。
index.php	Web APXの場合のみ、このファイルにはWeb APXのPHPソースコードが含まれます。このファイルには、Web APXの機能が記述されています。例については、チュートリアル: WebアプリケーションAPXの作成 (83ページ) を参照してください。

APX構成ファイル - apx.cfg

構成ファイルapx.cfgは、タイプに関わらずすべてのAPXに存在します。apxtool newコマンドがこのファイルのテンプレートを作成し、ユーザーはそれを変更することができます。このファイルには、APXを完全に記述するメタデータが含まれます。“apx.cfg” は**キー=値**形式を使用してAPXのプロパティを定義します。複数の行は、行継続文字 "\" によって結合されます。

表19 APX構成ファイルの属性に、すべてのAPXに共通の属性が記されています。APXタイプに固有の属性については、対応するAPXタイプの機能仕様に記述されています。いくつかの属性は、apx.cfg構成ファイルから抽出して、SAで管理することができます。説明などの変更可能な属性については、apx.cfgファイルを更新すると、SAで管理されているデータもそれに基づいて更新されます。

apx.cfgファイルの例を見るには、apxtool newコマンドを実行して、作成されたファイルを開いてください。

表 19　APX構成ファイルの属性

属性	変更	説明
タイプ	不可	APX のタイプ。webapp または script のどちらか(Script APX は Program APX とも呼ばれます)。作成後に APX タイプを変更することはできません。
name	可能	これはAPXの表示名であり、マルチバイト文字が含まれることがあります。この名前はいつでも変更できます。この名前はSAクライアントのAPXフォルダーにリストされます。
unique_name	不可	APXの一意の名前。この名前は、OGFSでAPXのファイル名として使用されます。この名前とタイプの組み合わせは、APXを一意に識別するキーの役割を果たします。作成後に名前を変更することはできません。この名前はファイルシステムで用いられるので、ファイルシステムの名前付け仕様に適合する必要があります。一般的には名前にはASCII文字だけを使用します。
version	可能	APX の現在のバージョンを表すバージョン文字列。値が文字列 "auto": で始まる場合、SA がバージョンを自動的に管理し、新しいバージョンごとに1ずつ増加する整数を使用します。
description	可能	APXの機能の説明テキスト。この属性の代わりに description.txtファイルを使用することもできます。
usage	可能	APXの使用法に関する説明テキスト。この属性の代わりに usage.txtファイルを使用することもできます。
interfaces	可能	APX が実装する 1 つまたは複数のインタフェース。複数のインタフェースはコロン (:) 文字で区切ります。
command	可能	APXを呼び出したときに実行される実行可能ファイル。

APXアクセス権エスカレーション構成ファイル - apx.perm

apx.permファイルは、アクセス権のエスカレーションルールを指定するために使用します。このファイルが存在しないか、エスカレーションアクセス権が記述されていない場合、APXはユーザーのデフォルトのアクセス権で動作します。

APX Toolの**New**コマンドで新しいAPXを作成すると、デフォルトのファイル群が生成され、その中にデフォルトのapx.permファイルがありますが、これにはエスカレーションアクセス権は定義されていません。デフォルトのファイルにはいくつかの例がコメントアウトされた形で記述されているので、APX開発者はそれをテンプレートとして利用できます。

エスカレーションを指定する3つの方法を次に示します。

- エスカレーションなし (81ページ)
- すべてのアクセス権 (81ページ)
- エスカレーションあり (81ページ)

エスカレーションなし

エスカレーション属性は指定されません。APXランタイムは、現在のユーザー権限を使用してAPXを実行します。ユーザーが持たない権限を必要とする操作をAPXが呼び出した場合、APXの実行はエラーによって終了します。

すべてのアクセス権

これは特殊な権限で、一時的にすべての操作へのアクセス権をユーザーに付与します。これは、開発またはデモ用にのみ用意されています。アクセス権を細かく調整する手間をかけずに簡単に概念実証やデモを行うために便利です。ただし、セキュリティが無効になるため、本番環境には適しません。

すべての権限を付与するには、`apx.perm`ファイルを編集して、すべての機能に一致するワイルドカード文字を含むマクロを記述します。以下に例を挙げます。

```
use_feature(name="*")
```

エスカレーションあり

事前定義された共通操作のリストを`apx.perm`ファイルに指定します。APXを実行する際に、APXランタイムは一時的にこれらのアクセス権をAPXに付与します。SAには、機能とリソースへのアクセス権の包括的なリストが用意されています。関連する機能のエスカレーションを容易にするため、ワイルドカード文字を使用して関連機能のグループに一致させることができます。以下に例を挙げます。

```
@use_feature(name="Application.*")
```

APXの進行状況の表示

`apxprogress`コマンドをProgram APXで使用すると、APXの進行状況に関する情報を提供することができます。これは、実行に時間がかかるProgram APXで、APXの進行状況をユーザーに通知したい場合に使用できます。

Web APXをProgram APXのフロントエンドとして使用して、Web APXの進行状況を表示することもできます。

apxprogressコマンド

`apxprogress`コマンドを使用すると、Program APXの実行ステップ数を定義し、各ステップの完了を記録することができます。これにより、APXのユーザーは、APXの進行状況と、残りがどれくらいかを知ることができます。

apxprogressの構文

`apxprogress {`**オプション**`}...`

表 20　apxprogressコマンドのオプション

オプション	説明
-i **総ステップ数**	APXの実行にかかる総ステップ数を指定します。このオプションは、APXの最初で1回使用して、APXの実行にかかる総ステップ数を指定します。 このオプションをAPX内で複数回使用すると、ステップ数を増やすことができます。使用するたびに、総ステップ数が指定した値だけ増加します。
-c <**現在のステップ**>	現在のステップ番号を指定します。APXコードの各ステップの終了時に、このオプションを指定してapxprogressを呼び出します。
-m <**メッセージ**>	APXのステータスを示すテキストメッセージを指定します。
-a <**データ**>	APXが自分自身に関して提供する追加情報を指定します。
-d	デバッグモードを示します。コマンドの出力をデバッグ用にstdoutに表示します。
-h	apxprogressコマンドに関するヘルプ情報を表示します。

apxprogressを使用するサンプルシェルスクリプト

次のシェルスクリプトは、apxprogressコマンドを使用するProgram APXの一部です。このAPXは、総ステップ数として100を定義し、現在の進行状況を100回報告します。また、報告のたびにステップ番号を示すメッセージも出力します。

```
#!/bin/sh
#########################################################################
# A simple shell script for a program APX that displays progress
# about itself.
# Author:<名前>
#########################################################################
echo "This is a simple APX that uses apxprogress."

totalsteps=100
apxprogress -i $totalsteps -c 1

for i in `seq $totalsteps`; do
        apxprogress -c $i -m "APX is running, working on step $i"  -d
        sleep 10
done
```

APXの進行状況の表示

SAのAPIメソッドJobService.getProgress()を使用して、apxprogressコマンドを呼び出す実行中のAPXの進行状況の情報にアクセスできます。このメソッドの使用例については、7. TwisterインタフェースへのAPXの進行状況の表示 (93ページ) の中のチュートリアル: Program APXの作成 (89ページ) を参照してください。

チュートリアル: WebアプリケーションAPXの作成

このチュートリアルは、mywebappという名前の単純なWebアプリケーションAPXを作成し、公開し、実行する方法を示します。

このチュートリアルで作成するAPXのデフォルトバージョンを実行すると、PHPコマンドphpinfoの出力が表示されます。チュートリアルの後の方で、PHPコードを変更して、管理対象サーバーのリストを表示する方法を示します。チュートリアルにはソースコードが付属しているので、PHPに関する事前の知識は必要ありません。

次の作業を順番に実行してください。

1. アクセス権の設定とチュートリアルフォルダーの作成 (84ページ)
2. 新規Webアプリケーションの作成 (84ページ)
3. 新規WebアプリケーションのSAへのインポート (86ページ)
4. 新規Webアプリケーションの実行 (86ページ)
5. Webアプリケーションの変更 (87ページ)
6. 変更したWebアプリケーションの実行 (88ページ)

チュートリアルの前提条件

このチュートリアルを実行するには、次の機能と環境が必要です。

- adminまたは**スーパー管理者**グループの他のメンバーとしてSAにログオンできること。adminでログオンすることで、アクセス権を設定できるようになります。
- **上級ユーザー**グループに属するユーザーとしてSAにログオンできること。

  上級ユーザーには、Webアプリケーションを作成して実行するアクセス権があります。このチュートリアルに示すコマンドの例では、このユーザーの名前はjdoeになっています。
- SA Webクライアントでクライアント機能のアクセス権を設定する方法に関する理解。

  アクセス権の詳細については、『SA 管理ガイド』の「ユーザーとグループの設定」の章を参照してください。
- SAクライアントでフォルダーを作成する方法に関する理解

  フォルダーの詳細については、『SAユーザーガイド: Server Automation』を参照してください。
- Global Shellセッションを開く方法に関する理解。

  詳細については、『SAユーザーガイド: Server Automation』の「Global Shell」の章を参照してください。
- lsやcdなどの基本的なUnixコマンドに関する理解。
- HTTPサーバーで動作するWebアプリケーションの開発経験。

1. アクセス権の設定とチュートリアルフォルダーの作成

1 SA Webクライアントにadminでログオンし、**上級ユーザー**グループに次のアクセス権があることを確認します。

- Manage Extensions:Read & Write

このアクセス権は、SA Webクライアントの [クライアント機能] タブにあります。

2 SA クライアントに**上級ユーザー**グループのメンバーでログオンし、SA ライブラリに次のフォルダーを作成します。

```
/Dev/MyApp
```

チュートリアルの後の方で、MyAppフォルダーにWebアプリケーションをアップロードします。チュートリアル以外の環境では、このフォルダーの名前は任意です。Web アプリケーションを置くフォルダーは任意に作成または選択してかまいません。

3 SAクライアントを終了します。

4 SAクライアントにadminでログオンし、MyAppフォルダーの [**フォルダーのプロパティ**] を開きます。

5 [**フォルダーのプロパティ**] の [**アクセス権**] タブで、**上級ユーザー**グループに次のアクセス権があることを確認します。

- フォルダーの内容のリスト表示
- フォルダー内のオブジェクトの読み取り
- フォルダー内のオブジェクトの書き込み
- フォルダー内のオブジェクトの実行

6 SAクライアントを終了します。

2. 新規Webアプリケーションの作成

1 **上級ユーザー**グループに属するSAユーザーとしてGlobal Shellセッションを開きます。

2 コアのOGFSホームディレクトリにmywebappという名前のディレクトリを作成し、そのディレクトリに移動します。

```
$ mkdir mywebapp
$ cd mywebapp
```

Webアプリケーションのファイルは、mywebappディレクトリに格納されます。

3 次に示すように、apxtool newコマンドを使用して、Webアプリケーションのディレクトリ構造とデフォルトのファイルを作成します。

```
$ pwd
/home/jdoe/mywebapp
$ ls
$
$ apxtool new -tw  -d "This is my first app."\
-u com.hp.sa.jdoe.mywebapp
Create source directory /home/jdoe/mywebapp/com.hp.sa.jdoe.mywebapp?Y/N y
Info:Successfully created APX 'mywebapp' source directory:/home/jdoe/
mywebapp.
```

-twオプションはAPXタイプがWebアプリケーションであることを示し、-dは説明を指定し、-uはアプリケーションの一意の名前を指定します。

apxtool newコマンドのオプションの詳細については、次のコマンドでオンラインヘルプを参照してください。

```
$ apxtool new -h
```

4 apxtool newコマンドで作成した新しいディレクトリに移動し、そこにあるファイルのリストを表示します。

```
$ pwd
/home/jdoe/mywebapp
$ cd com.hp.sa.jdoe.mywebapp
$ ls
APX-INF  cgi-bin  css  images  index.php
$ ls -R
.:
APX-INF  cgi-bin  css  images  index.php

./APX-INF:
apx.cfg  apx.perm  description.txt  interfaces  usage.txt

./cgi-bin:

./css:
hp_sa.css

./images:
```

5 デフォルトのindex.phpファイルの内容を表示します。

```
$ cat index.php
<?php

// Show information about PHP
phpinfo();

 ?>
```

他のWebアプリケーションと同様、index.phpファイルはindex.htmlファイルに置き換えることができます。ただし、このチュートリアルではindex.phpファイルを使用し、後でこのファイルを変更します。

6 APX-INFディレクトリにあるファイルをいくつか調べてみます。詳細については、APXファイル (79ページ) を参照してください。

APX-INFディレクトリには、APX Webアプリケーションに固有の情報が含まれています。次のcatコマンドで表示されるように、description.txtファイルにはapxtool newの-dオプションで指定したテキストが記録されています。

```
$ ls APX-INF/
description.txt  apx.cfg  apx.perm  usage.txt
$ cat APX-INF/description.txt
This is my first app $
```

次のgrepコマンドは、APX構成ファイルapx.cfgにあるプロパティの一部を表示します。typeとuniquenameの値は、apxtool newコマンドの-tオプションと-uオプションで決まります。APX構成ファイルの詳細については、APX構成ファイル - apx.cfg (79ページ)を参照してください。

```
$ grep "=" APX-INF/apx.cfg
type=webapp
name=mywebapp
unique_name=com.hp.sa.jdoe.mywebapp
```

3. 新規Webアプリケーションの SAへのインポート

Webアプリケーションをインポートすると、次の処理が実行されます。

- WebアプリケーションがSA内部のHTTPサーバーにインストールされます。
- WebアプリケーションがSAライブラリとGlobal Shellに表示されるフォルダーにコピーされます。
- Webアプリケーションにバージョン番号が割り当てられます。

次に示すように、apxtool importコマンドを入力し、プロンプトにyと答えます。-fオプションは、Webアプリケーションが格納されるSAライブラリ内のフォルダーを指定します。-cオプションは、Webアプリケーションの現在のバージョンを設定します。

```
$ pwd
/home/jdoe/mywebapp/com.hp.sa.jdoe.mywebapp
$
$ apxtool import -f "/Dev/MyApp" -c
APX source is not specified.
Do you want to publish current directory:/home/jdoe/mywebapp/
com.hp.sa.jdoe.mywebapp?Y/N y
APX with unique name 'com.hp.sa.jdoe.mywebapp' does not exist.
Register it into the system?Y/N y
Info:Successfully registered APX 'mywebapp' (310001) in folder '/Dev/MyApp'.
Info:Successfully published a new version '1' for APX 'mywebapp'.
Info:Successfully set APX 'mywebapp'(310001) current version as '1'.
```

4. 新規Webアプリケーションの実行

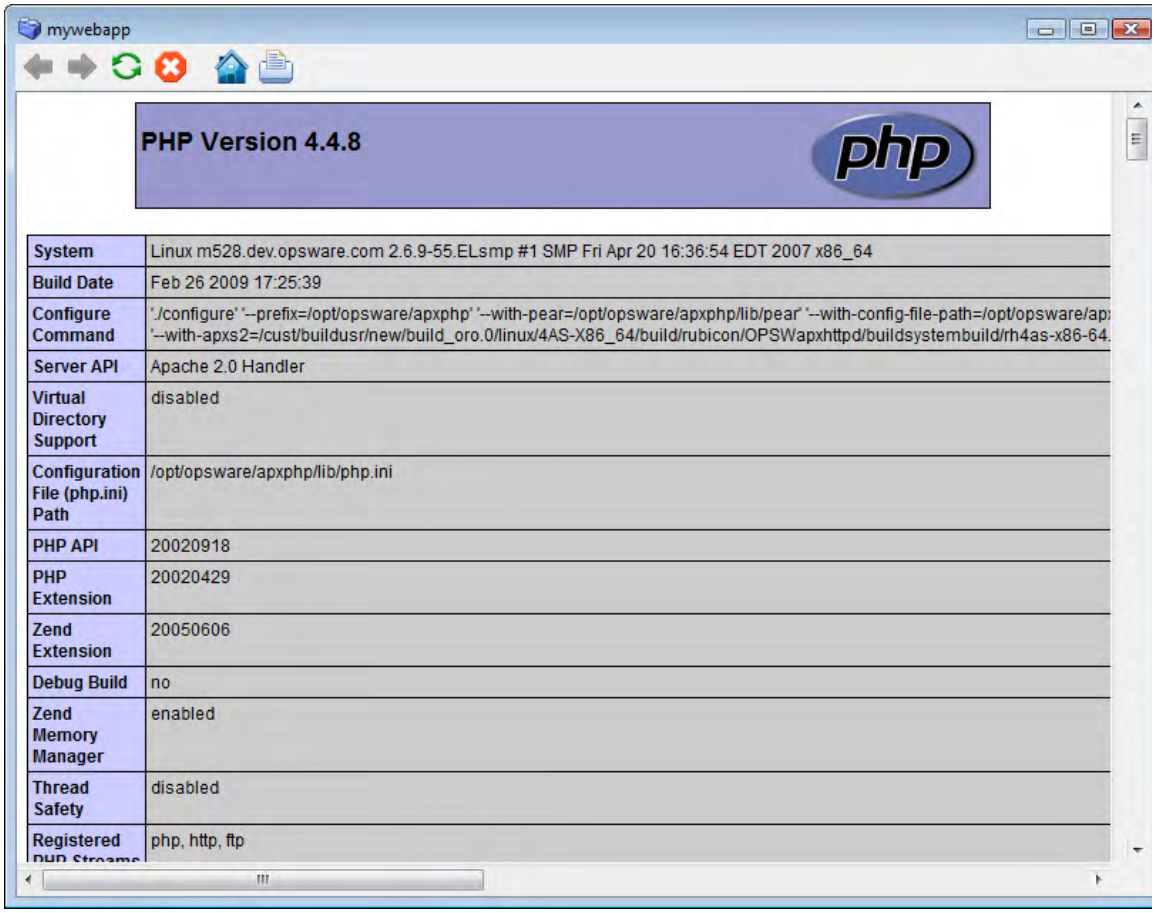
これでWebアプリケーションが公開されたので、エンドユーザーが行うのと同様に、SAクライアントからWebアプリケーションを実行できます。

1. **上級ユーザー**グループに属するユーザーとしてSAクライアントにログオンします。
2. [ライブラリ] タブと [タイプ別] タブを選択します。
3. [**拡張**] > [**Web**] ノードに移動すると、mywebapp拡張が表示されます。

   mywebappが表示されない場合は、1. アクセス権の設定とチュートリアルフォルダーの作成 (84ページ)に示す必要なアクセス権があることを確認してください。
4. Webアプリケーションを実行するには、mywebappを選択し、[**アクション**] > [**実行**] メニューを選択します。

   図5に示すウィンドウが表示されます。Webアプリケーションには、index.phpファイルのphpinfoステートメントによって生成された情報が表示されます。

図 5　　Webアプリケーションバージョン1



5. Webアプリケーションの変更

デフォルトのindex.phpファイルを実行するのは開発環境を確認するにはよい方法ですが、SAの機能は利用していません。ここでは、index.phpファイルを変更して、SAによって管理されるサーバーの名前のリストを表示します。

1　Global Shellセッションで、Webアプリケーションのindex.phpファイルを見つけます。

```
$ cd /home/jdoe/mywebapp/com.hp.sa.jdoe.mywebapp
$ ls
APX-INF  cgi-bin  css  images  index.php
```

2　index.phpファイルをviなどのテキストエディターで開きます。

3　index.phpの内容を次の各行に置き換えます。

```
<html>
<head>
<title>Servers</title>
</head>
<body>
<p>List of servers:</p>
```

```
<?php
passthru("ls /opsw/Server/@");
?>

</body>
</html>
```

上記のpassthruステートメントは、lsコマンドを実行し、stdoutを (再インフレートなしで) Webページに返します。lsコマンドは、OGFSにある管理対象サーバーの名前をリストします。

4 index.phpファイルを保存し、テキストエディターを終了します。

5 変更したWebアプリケーションを公開します。

次のapxtool importコマンドは、現在のバージョンを2に設定します。-Fオプションは確認プロンプトを抑制します。

```
$ apxtool import -f "/home/jdoe/mywebapp/com.hp.sa.jdoe.mywebapp" \
-c --version=2 -F
Info:Successfully published a new version '2' for APX 'mywebapp'
Info:Successfully set APX 'mywebapp'(310001) current version as '2'.
```

6. 変更したWebアプリケーションの実行

1 SAクライアントで、[**表示**] > [**更新**] メニューを使用して、Webアプリケーションの表示を更新します。mywebappのバージョン2が表示されるはずです。

2 mywebappを選択し、[**アクション**] > [**実行**] メニューを選択します。出力は図5のようになります。実際には、PHPのpassthruステートメントとOGSHのlsステートメントの出力が表示され、管理対象サーバーがリストされます。passthruステートメントは、lsコマンドによって返されるサーバー名の区切りの改行を削除します。

チュートリアル: Program APXの作成

このチュートリアルは、単純なシェルスクリプトを実行するmyshellappという名前の単純なProgram APXを作成し、公開し、実行する方法を示します。チュートリアルの後の方で、シェルスクリプトを変更して、apxprogressコマンドを実行し、進行状況の情報を提供する方法を示します。チュートリアルにはソースコードが付属しているので、シェルプログラミングに関する事前の知識は必要ありません。

次の作業を順番に実行してください。

1. アクセス権の設定とチュートリアルフォルダーの作成 (89ページ)

2. 新規Program APXの作成 (90ページ)

3. 新規APXのSAへのインポート (92ページ)

4. 新規APXの実行 (92ページ)

5. APXの変更 (92ページ)

6. 変更したAPXの実行 (93ページ)

7. TwisterインタフェースへのAPXの進行状況の表示 (93ページ)

チュートリアルの前提条件

このチュートリアルを実行するには、次の機能と環境が必要です。

- adminまたは**スーパー管理者**グループの他のメンバーとしてSAにログオンできること。adminでログオンすることで、アクセス権を設定できるようになります。
- **上級ユーザー**グループに属するユーザーとしてSAにログオンできること。

  上級ユーザーには、Webアプリケーションを作成して実行するアクセス権があります。このチュートリアルに示すコマンドの例では、このユーザーの名前はjdoeになっています。
- SA Webクライアントでクライアント機能のアクセス権を設定する方法に関する理解。

  アクセス権の詳細については、『SA 管理ガイド』の「ユーザーとグループの設定」の章を参照してください。
- SAクライアントでフォルダーを作成する方法に関する理解

  フォルダーの詳細については、『SAユーザーガイド: Server Automation』を参照してください。
- Global Shell (OGSH) セッションを開いてGlobal Shellを使用する方法に関する理解。

  詳細については、『SAユーザーガイド: Server Automation』の「Global Shell」の章を参照してください。
- lsやcdなどの基本的なUnixコマンドに関する理解。

1. アクセス権の設定とチュートリアルフォルダーの作成

1 SA Webクライアントにadminでログオンし、**上級ユーザー**グループに次のアクセス権があることを確認します。

  – Manage Extensions:Read & Write

  このアクセス権は、SA Webクライアントの [クライアント機能] タブにあります。

2 SA クライアントに**上級ユーザー**グループのメンバーでログオンし、SA ライブラリに次のフォルダーを作成します。

  /Dev/MyApp

チュートリアルの後の方で、MyAppフォルダーにProgram APXをアップロードします。チュートリアル以外の環境では、このフォルダーの名前は任意です。APXを置くフォルダーは任意に作成または選択してかまいません。

3 SAクライアントを終了します。

4 SAクライアントにadminでログオンし、MyAppフォルダーの [**フォルダーのプロパティ**] を開きます。

5 [**フォルダーのプロパティ**] の [**アクセス権**] タブで、**上級ユーザー**グループに次のアクセス権があることを確認します。

- フォルダーの内容のリスト表示
- フォルダー内のオブジェクトの読み取り
- フォルダー内のオブジェクトの書き込み
- フォルダー内のオブジェクトの実行

6 SAクライアントを終了します。

2. 新規Program APXの作成

1 **上級ユーザー**グループに属するSAユーザーとしてGlobal Shellセッションを開きます。

2 コアのOGFSホームディレクトリにmyshellappという名前のディレクトリを作成し、そのディレクトリに移動します。

```
$ mkdir myshellapp
$ cd myshellapp
```

Program APXのファイルは、myshellappディレクトリに格納されます。

3 次に示すように、apxtool newコマンドを使用して、Program APXのディレクトリ構造とデフォルトのファイルを作成します。

```
$ pwd
/home/jdoe/myshellapp
$ ls
$
$ apxtool new -ts -d "This is my first program APX."\
-u com.hp.sa.jdoe.myshellapp

Create source directory under '/home/jdoe/myshellapp/
com.hp.sa.jdoe.myshellapp' for APX 'myshellapp'?Y/N y
Info:Successfully created source directory '/home/jdoe/myshellapp/
com.hp.sa.jdoe.myshellapp for APX 'myshellapp'.
```

-tsオプションはAPXタイプがProgram APX (Script APXとも呼ばれる) であることを示し、-dは説明を指定し、-uはアプリケーションの一意の名前を指定します。

---

apxtool newコマンドのオプションの詳細については、次のコマンドでオンラインヘルプを参照してください。

```
$ apxtool new -h
```

---

4 apxtool newコマンドで作成されたファイルのリストを表示します。

```
$ pwd
/home/jdoe/mywebapp
```

```
$ ls
com.hp.sa.jdoe.myshellapp
$ cd com.hp.sa.jdoe.myshellapp
$ pwd
/home/jdoe/myshellapp/com.hp.sa.jdoe.myshellapp
$ ls -R
.:
APX-INF  run.sh

./APX-INF:
apx.cfg  apx.perm  description.txt  interfaces  usage.txt
```

5 デフォルトのrun.shファイルの内容を表示します。

```
$ cat run.sh
#!/bin/sh

#########################################################################
# APX myshellapp
#
# Created by: jdoe
#
#########################################################################
echo "This is APX myshellapp"
```

6 APX-INF ディレクトリにあるファイルをいくつか調べてみます。これらのファイルの詳細については、APXファイル (79ページ) を参照してください。

APX-INFディレクトリには、APXに固有の情報が含まれています。次のcatコマンドで表示されるように、description.txtファイルにはapxtool newの-dオプションで指定したテキストが記録されています。

```
$ ls APX-INF/
apx.cfg  apx.perm  description.txt  interfaces  usage.txt
$ cat APX-INF/description.txt
This is my first program APX.$
```

次のgrepコマンドは、APX構成ファイルapx.cfgにあるプロパティの一部を表示します。typeとuniquenameの値は、apxtool newコマンドの-tオプションと-uオプションで決まります。APX構成ファイルの詳細については、APX構成ファイル - apx.cfg (79ページ) を参照してください。

```
$ grep "=" APX-INF/apx.cfg
type=script
name=myshellapp
unique_name=com.hp.sa.jdoe.myshellapp
command=run.sh
```

3. 新規APXのSAへのインポート

APXをインポートすると、次の処理が実行されます。

- SAライブラリに表示されるフォルダーにAPXがコピーされます。
- APXにバージョン番号が割り当てられます。

次に示すように、apxtool importコマンドを入力し、プロンプトにyと答えます。-fオプションは、Webアプリケーションが格納されるSAライブラリ内のフォルダーを指定します。-cオプションは、Webアプリケーションの現在のバージョンを設定します。

```
$ pwd
/home/jdoe/myshellapp/com.hp.sa.jdoe.myshellapp
$
$ apxtool import -f "/Dev/MyApp" -c
APX source is not specified.
Do you want to publish current directory:/home/jdoe/myshellapp/
com.hp.sa.jdoe.myshellapp?Y/N y
APX with unique name 'com.hp.sa.jdoe.myshellapp' does not exist.
Register it into the system?Y/N y
Info:Successfully registered APX 'myshellapp' (20001).
Info:Successfully published a new version '1' for APX 'myshellapp'
Info:Successfully set APX 'myshellapp'(20001) current version as '1'.
```

これでAPXが公開されたので、他のSAユーザーが行うのと同様に、SAクライアントからAPXを実行できます。

4. 新規APXの実行

これでAPXが公開されたので、SAクライアントからAPXを実行できます。

1. **上級ユーザー**グループに属するユーザーとしてSAクライアントにログオンします。
2. [ナビゲーション] ペインで、[ライブラリ] タブを選択し [タイプ別] タブを選択します。
3. [拡張] ノードを開き、[プログラム] ノードを選択します。SAライブラリ内のすべてのProgram APXが表示されます。作成したAPXがここに表示されているはずです。myshellappが表示されない場合は、1. アクセス権の設定とチュートリアルフォルダーの作成 (89 ページ) に示す必要なアクセス権があることを確認してください。
4. APXを選択します。
5. [**アクション**] > [**実行**] メニュー項目を選択します。[プログラム拡張の実行] ウィザードが表示されます。
6. [次へ] ボタンを選択します。
7. [ジョブの開始] ボタンを選択します。
8. APXが終了したら、ステータスインジケーターを選択して詳細を表示します。
9. [閉じる] ボタンをクリックします。

5. APXの変更

このセクションでは、run.shファイルを変更して、進行状況の情報を提供するためのapxprogressコマンドの呼び出しを追加します。

1. Global Shellセッションで、APXのrun.shファイルを見つけます。

```
$ cd /home/jdoe/myshellapp/com.hp.sa.jdoe.myshellapp
$ ls
APX-INF run.sh
```

2 run.shファイルをviなどのテキストエディターで開きます。

3 run.shの内容を次の各行に置き換えます。

```
echo "This is a simple APX that uses apxprogress."

totalsteps=100
apxprogress -i $totalsteps -c 1

for i in `seq $totalsteps`; do
   apxprogress -c $i -m "myshellapx is running, working on step $i" #-d
   sleep 10
done
```

これらのapxprogressコマンドでは、APXに100のステップがあることを指定し、apxprogressを各ステップに1回、合計100回呼び出して、呼び出しの間に10秒間ずつ待ちます。詳細については、APXの進行状況の表示 (81ページ) を参照してください。

デバッグ用には、“#-d” “d” 変更し、シェルスクリプトを手動で実行することで、apxprogressコマンドの出力をstdoutに表示できます。

4 run.shファイルを保存し、テキストエディターを終了します。

5 変更したAPXを公開します。

次のapxtool importコマンドは、APXの新しいバージョンをロードし、現在のバージョンを2に設定します。-Fオプションは確認プロンプトを抑制します。

```
$ apxtool import -f "/home/jdoe/myshellapp" \
-c --version=2 -F
Info:Successfully published a new version '2' for APX 'myshellapp'
Info:Successfully set APX 'myshellapp'(20001) current version as '2'.
```

6. 変更したAPXの実行

これでAPXが変更され、再公開されたので、前のようにSAクライアントからAPXを実行できます。

1 SAクライアントで、[**表示**] > [**更新**] メニューを使用して、プログラム拡張の表示を更新します。myshellappのバージョン2が表示されるはずです。

2 APXを選択します。

3 [**アクション**] > [**実行**] メニュー項目を選択します。[プログラム拡張の実行] ウィザードが表示されます。

4 [次へ] ボタンを選択します。

5 [ジョブの開始] ボタンを選択します。

7. TwisterインタフェースへのAPXの進行状況の表示

apxprogress コマンドは、実行中の APX の進行状況を報告します。この進行状況情報は、API メソッド JobService.getProgress() を呼び出すことによって得られます。ここでは、このメソッドをTwisterインタフェースから実行する方法を示します。SA APIに対するTwisterインタフェースの詳細については、APIドキュメントとTwister (21ページ) を参照してください。

1 SAクライアントで、[ジョブとセッション] タブを選択します。

2 ジョブのリストで、作成したAPXを見つけます。

3 APXジョブのジョブID番号を記録します。これは後のステップで使用します。

4 Webブラウザーに次のURLを入力して、SA Twistインタフェースを実行します。

`https://<`**コアホスト**`>1032`

ここで、<**コアホスト**>はSAコアサーバーのIPアドレスまたはホスト名です。これは、SA APIに対するTwistインタフェースをWebブラウザーに表示します。

5 “Twister” リンクを選択します。これにより、SA APIに対するTwisterインタフェースが表示されます。ここでは、APIインタフェース、パッケージ、メソッドに関する詳細な情報を取得したり、メソッドを実行したりすることができます。

6 `JobService`インタフェースを見つけて選択します。これは`com.opsware.job`パッケージにあります。

7 下の方へスクロールして、`getProgress()`メソッドを見つけます。

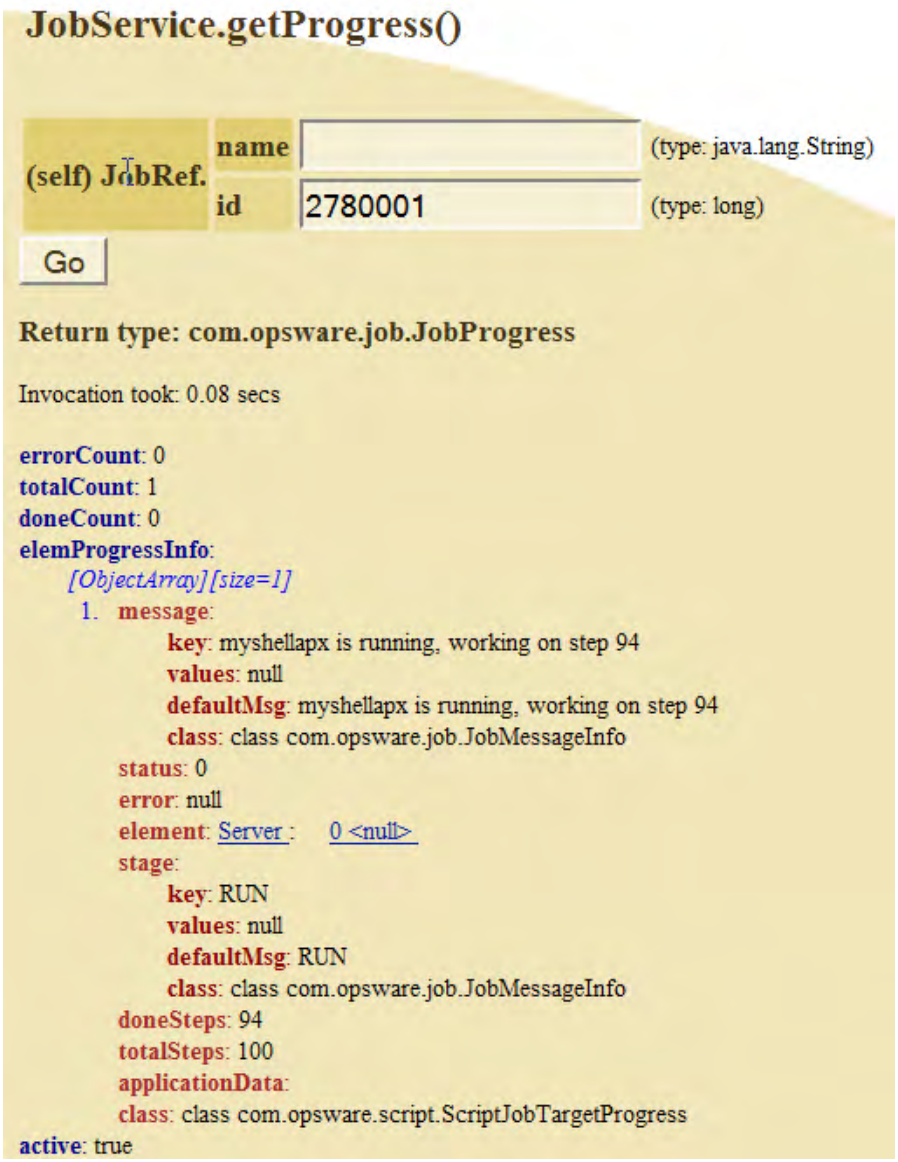
8 `getProgress()`メソッドのすぐ上の [試行] ボタンを選択します。

9 SAの資格情報を入力します。

10 [ログイン] ボタンを選択します。

11 [ID] フィールドに、上の手順3で記録した実行中のAPXのジョブ番号を入力します。

12 [実行] ボタンを選択します。`getProgress()`メソッドが呼び出され、`apxprogress`コマンドによるAPXの現在の進行状況情報が次のように表示されます。このスナップショットでは、総ステップ数が100で、完了したステップ数は94です。`getProgress()`メソッドの出力の詳細については、Twister Webブラウザーの [ナビゲーション] ペインで`getProgress()`メソッドを選択して、Javadocsドキュメントを参照してください。



第 5 章　エージェントツール

エージェントツールの概要

エージェントツールは、管理対象サーバーに関する情報の取得と変更のためのシェルスクリプト、バッチファイル、Pythonスクリプトの集合です。情報の取得と変更は、SAデータベースから行われます。

これらのスクリプトを使用すると、カスタムフィールド、カスタマー割り当て、カスタム属性などを取得し、変更することができます。この機能により、従来サーバー単位で実行する必要があったさまざまな手順を自動化できます。

さらに、スクリプトで取得した情報を、独自設計のカスタムスクリプトに組み込むこともできます。カスタマー割り当てやカスタム属性といった情報は管理対象サーバーごとに異なるため、カスタムスクリプトでこのような情報を「実行時」に取得して使用できれば非常に便利です。

以下に例を挙げます。

- アプリケーションのインストール後の構成作業を行うスクリプトで、サーバーが登録されているファシリティの名前を知る必要があるとします。エージェントツールに含まれるスクリプトを使用すれば、ファシリティ名を取得して、インストール後スクリプトに自動的に挿入することができます。
- 監視エージェントをインストールする際に、インストール後スクリプトは、構成ファイルを変更して、当該ファシリティでの監視サーバーのIPアドレスを書き込む必要があります。エージェントツールに含まれるスクリプトを使用すれば、コアのカスタム属性を読み取って監視サーバーのIPアドレスを取得し、構成ファイルに挿入することができます。
- 多数のサーバーのEEPROMバージョンを取得して、その情報をカスタム属性またはカスタムフィールドに格納するDSEを作成できます。

エージェントツールのスクリプトには、他にも次のような使用法があります。

- ソフトウェアのインストール時に構成に使用する情報をSAコアから取得。
- DSE、Global Shellスクリプト、ソフトウェアのインストールの実行時に、管理対象サーバーからのメタデータをSAデータベースに格納。
- 管理対象サーバーに関するカスタム属性情報を取得。

インストール要件

エージェントツールの要件を次に示します。

オペレーティングシステムのサポート

エージェントツールは、SA管理対象サーバーがサポートするオペレーティングシステムをサポートします。サポートされるオペレーティングシステムのリストについては、『SA Standard/Advanced Installation Guide』を参照してください。

セキュリティ、アクセス制御、認証

エージェントツールは、Unix/Linuxシステムではrootユーザーで、Windowsシステムでは管理者で実行する必要があります。エージェントツールは、サーバーエージェントの証明書を使用して、Webサービスデータアクセスエンジン (twist) に接続し (これはpyTwistのデフォルトの動作)、Webサービスデータアクセスエンジンがエージェントに与える権限を付与されます。これは通常、エージェントツールが実行されるサーバー上の読み取り/書き込み権限に当てはまるので、ユーザー認証は不要です。

例外として、set_customerスクリプトがあります。サーバーをカスタマーに関連付けるには、そのカスタマーに対する読み取りアクセス権が必要です。エージェント証明書は他のカスタマーに対する読み取りアクセス権を持たないため、このスクリプトを実行する際にユーザーは認証を行う必要があります。

その他の要件

- pyTwistへのアクセス権限
- SA APIへのアクセス権限
- インストール済みのPython 2.4 (サーバーエージェントに付属)

インストール:

エージェントツールは、HP SAインストーラーの通常のコアインストールプロセス中にコアにインストールされます。ただし、エージェントツールを管理対象サーバー上で使用できるようにするには、これらのサーバーにもエージェントツールをインストールする必要があります。ここではそのプロセスについて説明します。

エージェントツールは、実行可能スクリプトの集合として管理対象サーバーにインストールされます。これには、シェルまたはバッチスクリプト (オペレーティングシステムに依存) と、これらのスクリプトから呼び出されるPythonスクリプトがあります。これらのスクリプトを管理対象サーバーから実行することにより、SAコアの情報の取得と変更が可能です。これらのスクリプトは、手動で実行することも、パッケージインストールスクリプト、DSE、Global Shellシェルスクリプトなどから呼び出すこともできます。

エージェントツールは、Python SA APIアクセス (pyTwist) ソフトウェアポリシーの一部として提供されています。このポリシーは次のディレクトリにあります。

```
/Opsware/Tools/Python Opsware API Access
```

エージェントツールの手動インストール

管理対象サーバーにエージェントツールをインストールするには、次の手順を実行します。

1. SAクライアントを起動します。
2. [**管理対象サーバー**] リストを表示し、エージェントツールをインストールする管理対象サーバーを選択します。
3. 右クリックして [**ソフトウェアのインストール**] を選択します。
4. [**Python Opsware APIアクセス**] ソフトウェアポリシーを選択します。
5. ソフトウェアポリシーのインストールウィザードが、プロセスの残りの手順をガイドします。

エージェントのインストール時のエージェントツールのインストール

別の方法として、エージェントのインストール時にPython SA APIアクセスソフトウェアポリシーのIDを指定して、これを修復するように指定することもできます。エージェントのインストールの詳細については、『SA管理ガイド』を参照してください。

エージェントツールのアップグレード

エージェントツールはソフトウェアポリシー (pyTwist ソフトウェアポリシーの一部) として提供されているため、コアのアップグレード後に修復を実行することで、エージェントツールの新しいバージョンにアップグレードすることができます。

SAコアをアップグレードすると、Python SA APIアクセスソフトウェアポリシーも更新されます。エージェントツールの古いバージョンは削除され、新しいバージョンがポリシーにアタッチされます。SAコアのアップグレード (この際にエージェントツールもコアのアップグレードの一部として自動的にアップグレードされます) の後で、管理対象サーバー上のエージェントツールをアップグレードするには、次の作業を実行します。

1. エージェントツールがインストールされている管理対象サーバーを選択します。Python SA APIアクセスソフトウェアポリシーにアタッチされているサーバーとグループのリストを表示するには、ポリシー自体を開きます。
2. 選択したサーバーを右クリックして [**修復**] を選択します。
3. [**Python Opsware APIアクセス**] ソフトウェアポリシーを選択します。
4. pyTwistとエージェントツールパッケージの古いバージョンが削除され、新しいバージョンがインストールされます。

データ移行

エージェントツールは管理対象サーバー上に永続的なデータを保持しないため、データの移行や保存の必要はありません。

エージェントツールのスクリプト

使用法

<スクリプト名>.py|bat|sh --引数

表 21　エージェントツールのスクリプト

スクリプト	機能
get_all_cust_attr	サーバーレコードのすべてのカスタム属性を取得します。 使用法: get_all_cust_attr.py [--localonly] [--mode=python|shell|pretty] modeは、出力の形式 (Python辞書、シェルステートメントなど) を指定します。デフォルトはprettyです。 **注**: 複数行のカスタム属性がある場合、modeにshellは使用できません。
get_cust_attr	1つのカスタム属性の値を取得します。 **使用法**: get_cust_attr.py [--localonly] <**カスタム属性名**>
set_cust_attr	サーバー上の1つのカスタム属性の値を設定します。 使用法: set_cust_attr.py <**カスタム属性名**> <**カスタム属性値**>|--valuefile <**値を記録したファイルのパス**>
del_cust_attr	データベース内のサーバーのレコードからカスタム属性を削除します。 **使用法:** del_cust_attr.py <**カスタム属性名**>
get_cust_field	1つのカスタムフィールドの値を取得します。 **使用法:** get_cust_field.py <**カスタムフィールド名**>
set_cust_field	サーバー上の1つのカスタムフィールドの値を設定します。 **使用法:** set_cust_field.py <**カスタムフィールド名**> <**カスタムフィールド値**>|--valuefile <**値を記録したファイルのパス**>
get_customer	サーバーが関連付けられているカスタマー名を取得します。 **使用法**: ./get_customer.py
set_customer	サーバーが関連付けられているカスタマー名を設定します。 **使用法:** set_customer.py<**カスタマー名**>
get_facility	サーバーが関連付けられているファシリティの名前を取得します。 **使用法**: ./get_facility.py

表 21　エージェントツールのスクリプト (続き)

スクリプト	機能
get_info	サーバーのすべてのフィールドを (OGSHのサーバーの情報ファイルと似た形式で) 出力します。 **使用法**: `get_info.py`

get_history	サーバー固有のイベントを印刷します。
	使用方法:
	`get_history.py --startdate <`**エポックを基準とした開始日 (秒単位)**`>`
	`[--enddate <`**エポックを基準とした終了日 (秒単位)**`>]`
	`[--username <SAS`**ユーザー名**`>] [--password <SAS`**パスワード**`>]`

sub_text_file	テキストファイルを読み込み、ファイルからトークン/パラメーターを探して、カスタム属性の値に置き換え、変更したファイルをstdoutに出力します。使用できるファイル形式については下を参照してください。 **使用法**: `sub_text_file.py [--localonly]` **<トークンを記録したファイルのパス>**

sub_text_fileスクリプトに使用できる形式

`sub_text_file`スクリプトに渡すテキストファイルは任意の内容を持つことができますが、スクリプトは@文字が2個ある行を探し、@文字のペアとその間の文字列をトークンとして扱います。行に@文字が1個あってもその行は無視されますが、同じ行にもう1個@文字があると、2個の@文字の間のすべての文字列がトークンと見なされます。

トークンは、@文字の間に指定されたカスタム属性の値に置き換えられます。たとえば、`@dns_server@`という文字列は、カスタム属性`dns_server`の値に置き換えられます。このカスタム属性が存在しないか値が空の場合は、トークンは空文字列に置き換えられます。

次のエントリを含むテキストファイルがあるとします。

```
IP:@monitoring_server_ip@
```

スクリプトの出力は次のようになります。

```
IP:82.159.202.117
```

ここで、`IP`は`monitoring_server_ip`によって取得される値です。

出力

`sub_text_file`スクリプトは`stdout`に出力します。必要な場合、出力をファイルにリダイレクトできます。また、zipファイルに格納されている`.template`ファイルを使用して、出力をフォーマットできます。以下に例を挙げます。

```
$AGENTTOOLSPATH/sub_text_file.sh petstore_config.template >
petstore_config.cfg
```

エージェントツールのスクリプトの例

エージェントツールのスクリプトの簡単な使用例を次に示します。

Unix/Linux

この例は、ファシリティの名前を含むメッセージを、ユーザーがUnixサーバーにログインしたときに表示されるMOTD (Message of the Day) に挿入します。

```
./etc/opt/opsware/pytwist/pytwist.conf
facility_name=`$AGENTTOOLSPATH/get_facility.sh`
echo "You have connected to a server in the $facility_name facility.For hardware
information on this server as stored in Opsware, run $AGENTTOOLSPATH/
get_info.sh."> /etc/motd
```

Windows

このWindowsの例は、サーバーに関する情報を記録したテキストファイルを、すべてのユーザーのデスクトップに置きます。

```
call "C:\Program Files\Common Files\Opsware\etc\pytwist\
pytwist_conf.bat"

call"%AGENTTOOLSPATH%\get_info.bat" > "%SYSTEMDRIVE%\Documents and
Settings\All Users\Desktop\server_info_from_Opsware.txt"
```

エージェントツールへのパスはコードに直接書き込まず、次の方法を使用してください。

1.PyTwist構成ファイルを実行します。
Unix:
`./etc/opt/opsware/pytwist/pytwist.conf`
Windows:
次のファイルを呼び出します。
`C:\Program Files\Common Files\Opsware\etc\pytwist\pytwist_conf.bat`

2.環境変数を使用します。
Unix:
`$AGENTTOOLSPATH`
Windows:
`%AGENTTOOLSPATH%`

この方法を使用すれば、エージェントツールへのパスが将来変更されても、スクリプトのエラーを防ぐことができます。

第 6 章　Microsoft Windows PowerShell/SA統合

Microsoft Windows PowerShellの概要

Windows PowerShellは、Microsoftの.Net 2.0 Frameworkクラスライブラリに統合された、システム管理者やプログラマーのための拡張可能なコマンドシェルです。.NET の共通言語ランタイムと .NET Framework を使用し、.NETオブジェクトを受け取って返します。Windowsの管理と構成のためのツールと方法を強化する役割を果たします。

Windows PowerShell には多数の「コマンドレット」が装備されており、さまざまな機能を提供します。コマンドレットは個別に使用することも、組み合わせてもっと複雑な作業のために使用することもできます。

Windows PowerShellは、コンピューターのファイルシステムへのアクセスを提供するだけでなく、PowerShellプロバイダーによりレジストリやデジタル署名証明書ストアなどのデータストアへのアクセスを可能にします。プロバイダーとは、サービスとデータソースの間の一様なインタフェースを提供するソフトウェアモジュールです。

SAでWindows PowerShellを使用するには、Microsoft Windows PowerShellの知識と使用経験が前提となります。PowerShellに関する情報や手順については、http://www.microsoft.com/ja-jpを参照してください。

付属のコマンドレットは管理対象サーバー上のデータを変更できるので、Windows PowerShell とその使用法について十分に理解しておくことが重要です。

Windows PowerShellとSAとの統合

SAでは、Windowsが動作する管理対象サーバーでのMicrosoft Windows PowerShellとの初期統合が提供されています。PowerShellはSAユーザーインタフェースから使用でき、SAデータは標準PowerShell環境または任意のPowerShell実行空間から利用できます。PowerShell実行空間とは、PowerShellランタイムシステムのホスティング環境です。

SAでは次のPowerShellコマンドレットが使用可能です。

- Get-SASServer
- Set-SASServer
- Get-SASJob

また、SAにはPowerShell SASプロバイダー (PowerShell環境でSAコア内のオブジェクトにアクセスするためのコンポーネント) も付属しています。

統合PowerShell/SAコマンドレット

表22に、SA付属の統合PowerShell/SAコマンドレットの一覧と説明を示します。

表 22　PowerShellコマンドレット

コマンドレット	説明	引数

`Get-SASServer`	Retrieves server data from specified server(s)	`-Credential <PSCredential>`
		`-Core <Hostname|IPAddress>`
		`-Name < ListOfHostnameFragments> |`
		`-Id <ListOfServerIDs>`

`Get-SASJob`	Retrieves data for specified jobs	`-Credential <PSCredential>`
		`-Core <Hostname|IPAddress>`
		`-JobFilter <ListOfJobIDs>`

`Set-SASServer`	Retrieves a list of managed servers	`-Credential <PSCredential>`
		`-Core <Hostname|IPAddress>`
		`-Server <ServerVO>`

インストール要件

システム管理者のWindowsデスクトップにインストールするコマンドレットとPowerShell SAプロバイダーのアセンブリ、構成、セットアップファイルを含むMSIインストーラーパッケージ。

オペレーティングシステムのサポート

- Windows Server 2003
- Windows Server 2008
- Windows Server 2008 R2 x64
- Windows Server 2012

インストール:

Microsoft Windows PowerShell/SA統合を実装するには、次の作業を実行します。

1. OCCの [**ライブラリ**] > [**ソフトウェアポリシー**] で、Microsoft Windows PowerShell/SAコネクター MSIを見つけます。
2. MSIを実行して、SA固有のコマンドレットとSAプロバイダーを定義するアセンブリをインストールします。ファイル`readme.rtf`に最新情報が記載されています。Microsoft Windows PowerShell初期化スクリプトprofile.ps1 (.bashrcと同様のもの) と、SA環境でのPowerShellの使用方法を示すサンプルPowerShellスクリプトのセットもインストールされます。

デフォルトでは、MSIはコネクターをC:\Program Files\Opsware\PsSasにインストールします。

ファイルSAS-WSAPI.ps1は、コマンドレットを使用せずにPowerShellから直接WS-APIにアクセスする方法を示します。

Microsoft PowerShell/SA統合の機能

Microsoft PowerShellは次の分野でオプションとして使用できます。

- 管理対象サーバーへのリモートアクセス
- 監査とスナップショットのルール
- DSEスクリプト統合

管理対象サーバーへのリモートアクセス

SAクライアントから、リモートターミナルを開くのと同じ方法で、任意の管理対象サーバー (サーバーのグループは不可) に対するリモートPowerShellセッションを開くことができます。

1 SAクライアントを起動します。

2 ナビゲーションペインで、[**デバイス**] > [**すべての管理対象サーバー**] を選択します。

3 管理対象サーバーを選択して開きます。

デバイスエクスプローラーウィンドウで、[**アクション**] メニューから [**リモートPowerShellの起動**] を選択します。

リモートPowerShellセッションにログインしている間は、「WMI呼び出し」を含むスクリプトは実行できません。WMI呼び出しを含むスクリプトを実行しようとすると、そのスクリプトの実行に必要なアクセス権を持つグループのメンバーであっても、**アクセスが拒否されました**エラーが発生します。

監査とスナップショットのルール

Microsoft PowerShellは、SA監査と統合されています。カスタムスクリプトルールを構成する際のオプションとして、バッチ、Python 2、Visual Basicに加えてMicrosoft PowerShellスクリプトが使用できるようになりました。監査の詳細については、『SAユーザーガイド: 監査とコンプライアンス』を参照してください。

DSEスクリプト統合

管理対象サーバーに対して、Pytwistを使用してSA APIを呼び出すPowerShellスクリプトを用意すれば、エンドユーザーはスクリプトをDSEまたはISMコントロールとして呼び出すことができます。Pytwist APIを呼び出すスクリプトの作成方法については、PytwistによるPython APIアクセス (53ページ) を参照してください。

サンプルセッション

ここでは、Windows PowerShell/SA統合の使用方法を示す4つのシナリオを紹介します。

- シナリオ1は、SAコアから管理対象サーバーデータを抽出して変更し、コアに書き戻す方法を示します。
- シナリオ2は、SA管理対象サーバーデータをWindows PowerShell/SA統合を使用してExcelスプレッドシートにエクスポートする方法を示します。
- シナリオ3は、SAコアをWindows PowerShell PSdriveとしてマウントし、仮想ファイルシステム内を調べる方法を示します。
- シナリオ4は、Windows PowerShell環境で使用可能なすべてのタイプのSAオブジェクトをリストする方法を示します。

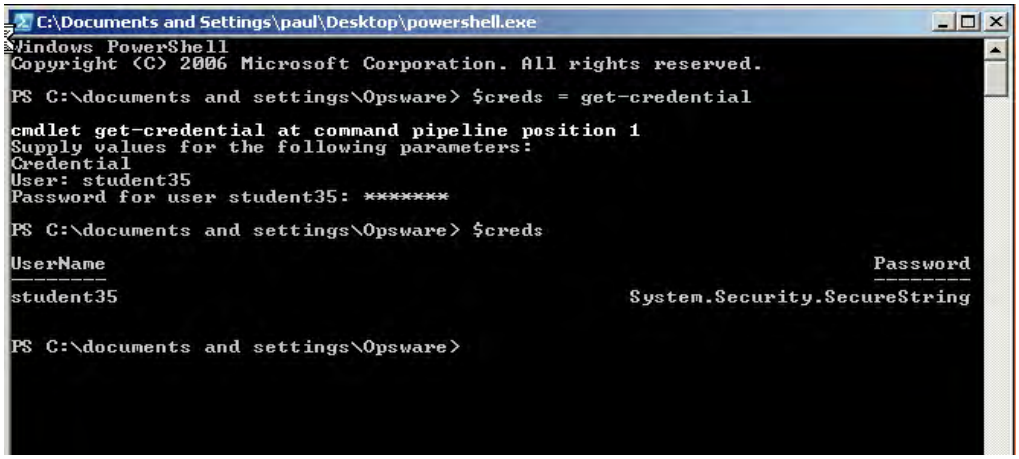
シナリオ1

SA コアに対する認証を行い、管理対象サーバーに関するデータを取得し、データを変更し、SA コアに書き戻します。

1 デスクトップアイコンからPowerShellプロンプトを開きます。

2 SAコア資格情報をPowerShellシェル変数にセキュアに格納します。詳細については図6を参照してください。

図 6　SA資格情報のPowerShell変数への格納



3 Get-SasServerコマンドレットを使用すると、図7に示すようにサーバーを表すSAレコードを取得できます。

図 7　Get-SasServerコマンドレットの使用

返されたオブジェクトはシェル変数に格納されます。

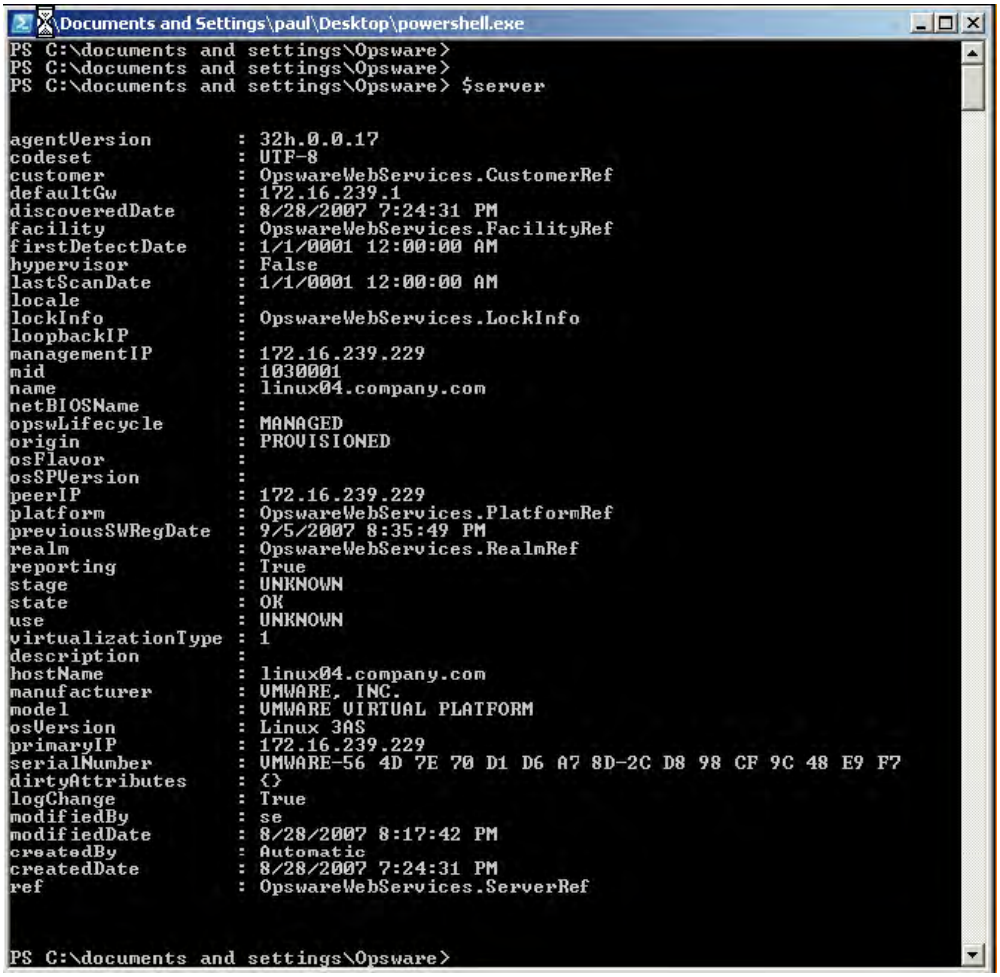
Get-SasServer コマンドレットは、サーバーデータを取得するSAコアを識別するパラメーター、SAコアに操作のための資格情報 (操作を実行するSAユーザーアカウントの識別と認証に使用) を提供するパラメーター、要求対象のサーバーを識別するパラメーターを取ります。

Get-SasServerを初め、すべてのコマンドレットの引数に関する詳細を知るには、PowerShellのGet-Helpベースコマンドレットを次のように使用します。

```
Get-Help Get-SasServer -detailed
```

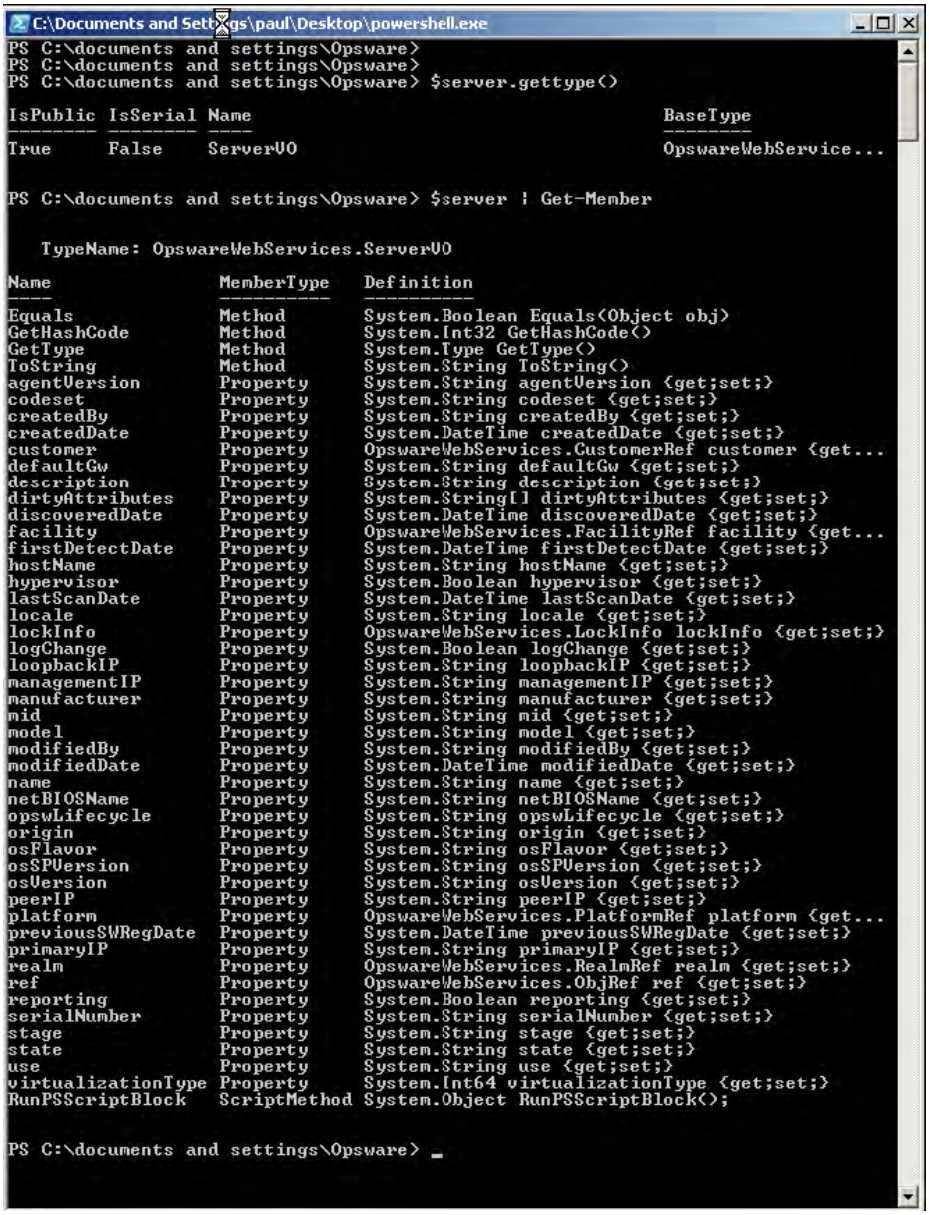
4 返されたオブジェクトのプロパティを見るには、シェル変数の名前を入力します。詳細については図8を参照してください。

図 8 SAサーバーのプロパティの表示



5 オブジェクトのプロパティ、プロパティのタイプ、PowerShell スクリプトからオブジェクトに対して呼び出せるメソッドのリストを表示する方法については、図9を参照してください。

図 9　　オブジェクトのプロパティのリストの表示



6 オブジェクトの**説明**属性をWindows PowerShellで変更した後、`Set-SasServer`コマンドレットを呼び出して、変更した`ServerVO`オブジェクトをコマンドレットに渡すことができます。このコマンドレットは、`ServerVO`オブジェクトを受け取って、SAコアの管理対象サーバーレコードを更新します。`Set-SasServer`コマンドレットは、更新したデータを書き込むSAコアを識別するパラメーターと、操作を実行するSAユーザーアカウントを識別する資格情報のパラメーターを受け取ります。

更新操作が完了すると、更新された`ServerVO`がWindows PowerShellに返され、図10に示すようにプロパティがプロンプトに表示されます。

図 10　オブジェクトの説明の変更

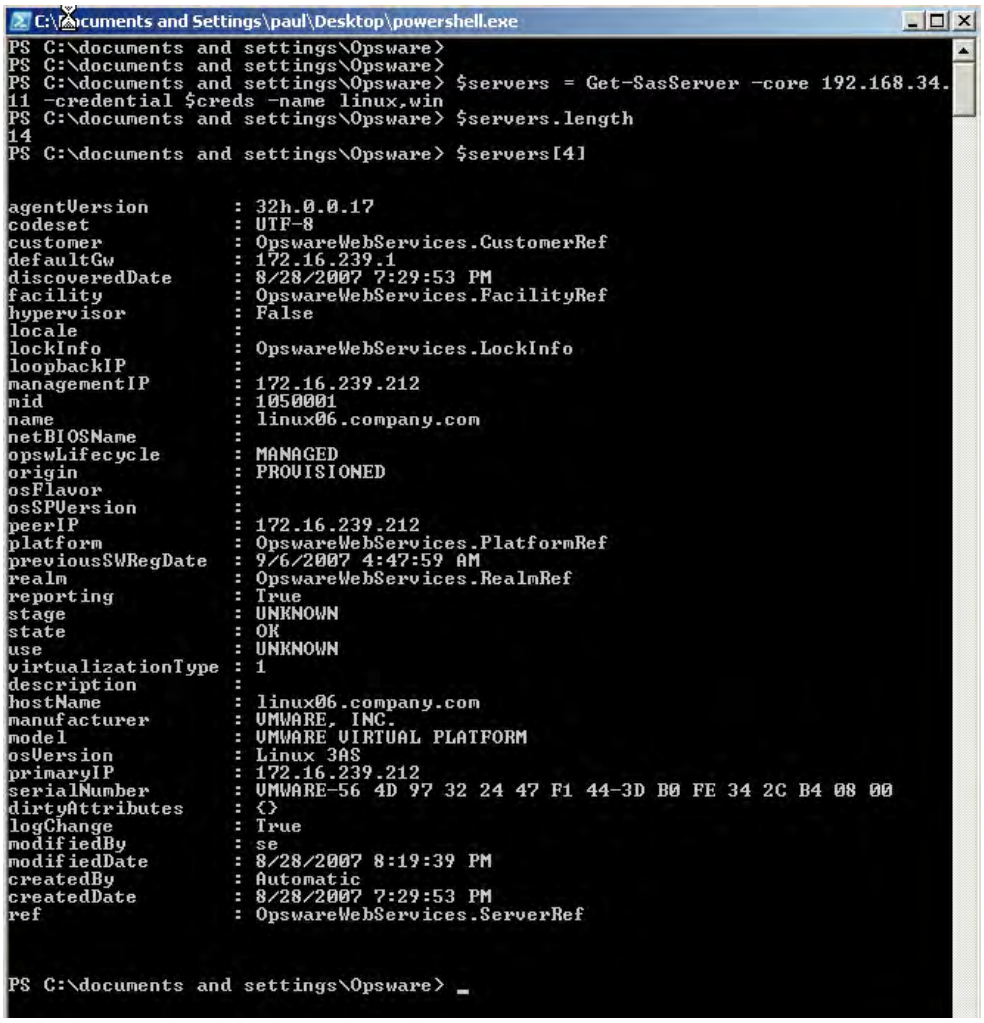
シナリオ2

このシナリオは、SAコアからすべての管理対象サーバーデータを取得し、Microsoft Excelに表示する方法を示します。

1 `Get-SasServer` コマンドレットを使用して、各 Linux および Windows 管理対象サーバーに対応する`ServerVO`をSAコアから取得します。次に示すセッションで、-nameパラメーターは名前に一致するフィルターのリスト (例、`-name linux,win`) をSAコアに提供するために使用されています。

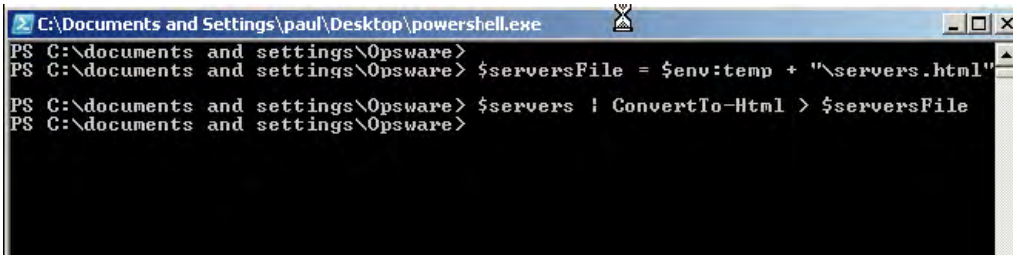
Get-SasServerコマンドレットはServerVOの配列を返します。この例では配列の長さは14です。この配列にインデックスを指定することで、任意のServerVOオブジェクトを得ることができます。詳細については図11を参照してください。

図 11　Get-SasServerコマンドレットでの名前フィルターの使用



2　その後、ServerVOデータをHTMLにフォーマットして、一時ファイルに保存できます。一時ファイルはTEMPディレクトリに作成されます。PowerShellセッションで、%TEMP%環境変数の値を取得するには、$env:tempと入力します。詳細については図12を参照してください。

図 12　ServerVOデータをHTMLに変換して一時ファイルに保存



3　ベースWindows PowerShellコマンドレットのNew-Objectを使用して、Microsoft Excelを起動し、そのExcelインスタンス内部に新規ワークブックを作成し、一時ファイルの内容をワークブックに入力することができます。最後に、実行中のExcelインスタンスを表示します。これにより、Excelがフォアグラウンドに移動します。その後、データを日付や列の値などで並べ替えて、たとえば、各サーバーがSAコアの管理対象になった日付を知ることができます。詳細については図13を参照してください。

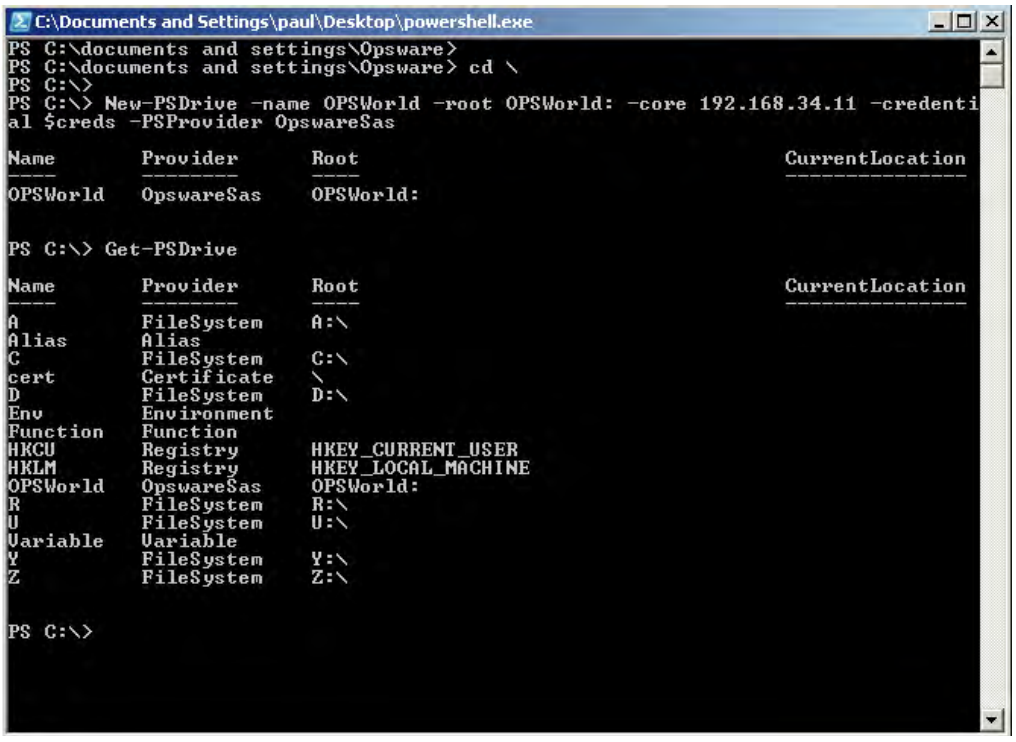
図 13　New-Objectコマンドレットによる Microsoft Excelの起動

シナリオ3

このシナリオでは、SAコアをWindows PowerShell PSDriveとしてマウントし、SAのSA**Jobs**フォルダーを表示して、その内容を読み取る方法を示します。

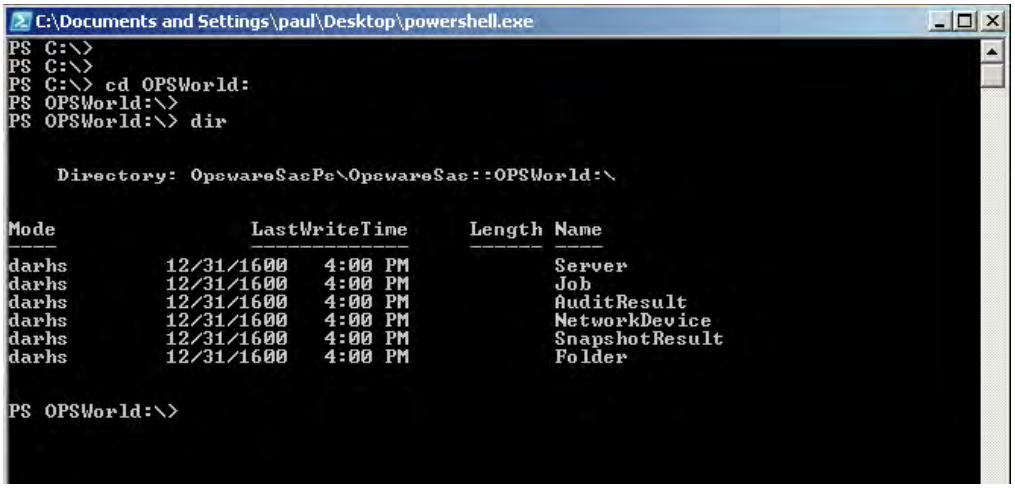
1　SAコアをWindows PowerShell PSDriveとしてマウントします。PowerShellでは、さまざまなデータストアやリポジトリを、ファイルシステムと同じ方法で操作できます。このシナリオでは、SAコア (具体的には管理対象環境データストア) を「マウント」して、`OPSWorld`という名前のドライブとして操作できるようにします。この仮想ファイルシステムに対するデータの読み書きが行われるか、クライアントからファイルシステムのナビゲーションが実行されるたびに、Windows PowerShell ベースシステムは、PowerShell SASプロバイダー (`-PSProvider OpswareSas`) を呼び出します。詳細については図14を参照してください。

図 14　SAコアをWindows PowerShell PSDriveとしてマウント



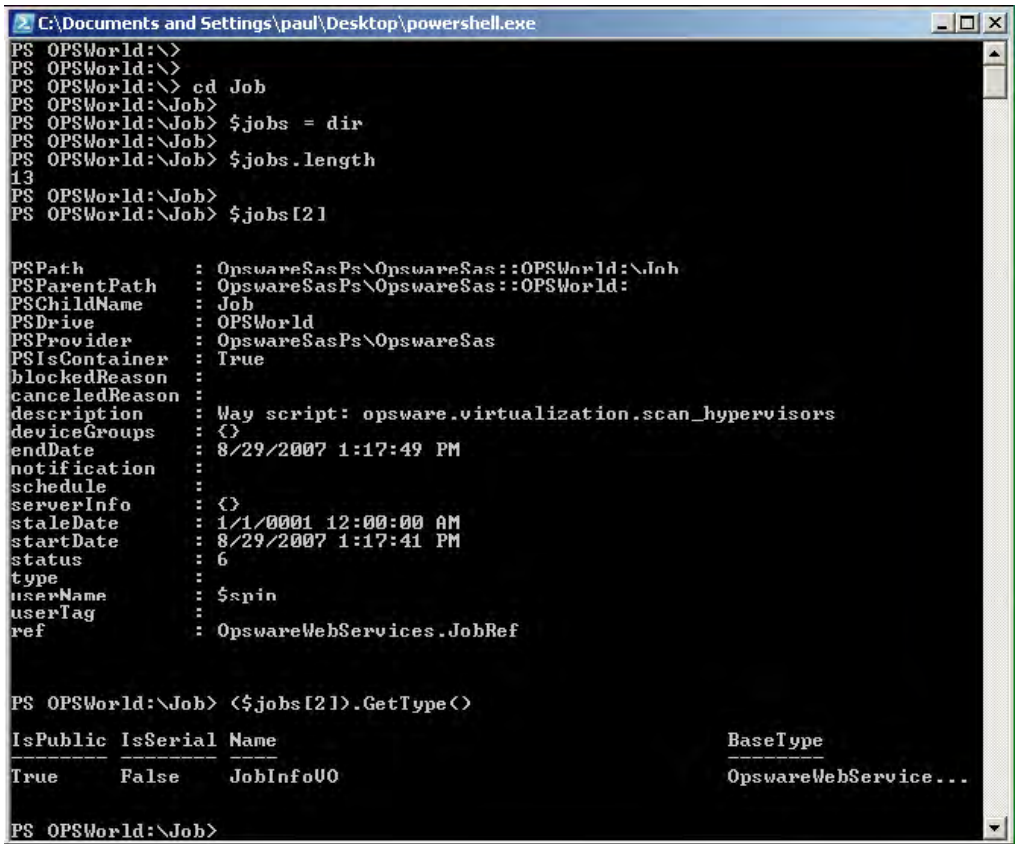
2　新しくマウントしたドライブにディレクトリを変更し、ディレクトリリストを取得します。`dir`は`Get-ChildItem`コマンドレットに対するPowerShellエイリアスです。詳細については図15を参照してください。

図 15　Get-Childコマンドレットの別名であるDIR



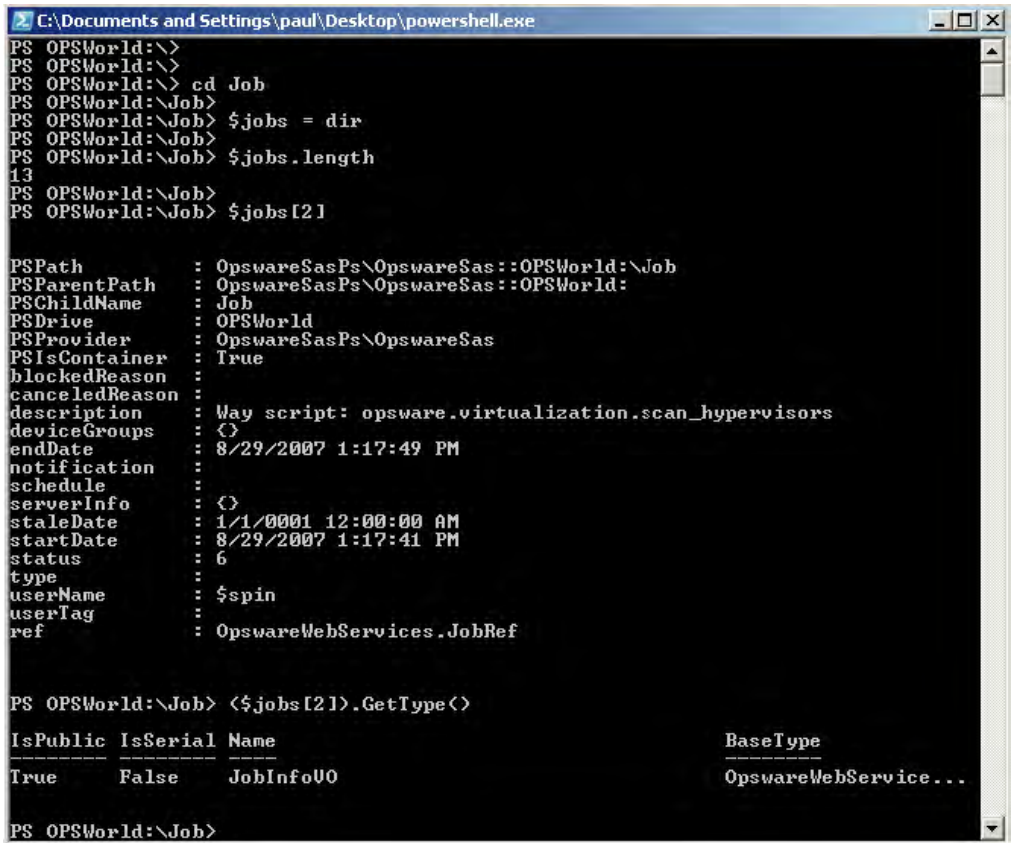
3　**Jobs**フォルダーに移動し、ディレクトリリストを取得し、ディレクトリリストをシェル変数に保存します。SAコアから取得されたJobInfoVOオブジェクトの配列がシェル変数に格納され、インデックスを使用してアクセスできます。詳細については図16を参照してください。

図 16　ディレクトリリストのPowerShell変数への保存



4 C:ドライブに移動し、OPSWorld PSDrivexを削除します。詳細については次を参照してください。図17

図 17 OPSWorld PSDriveの削除

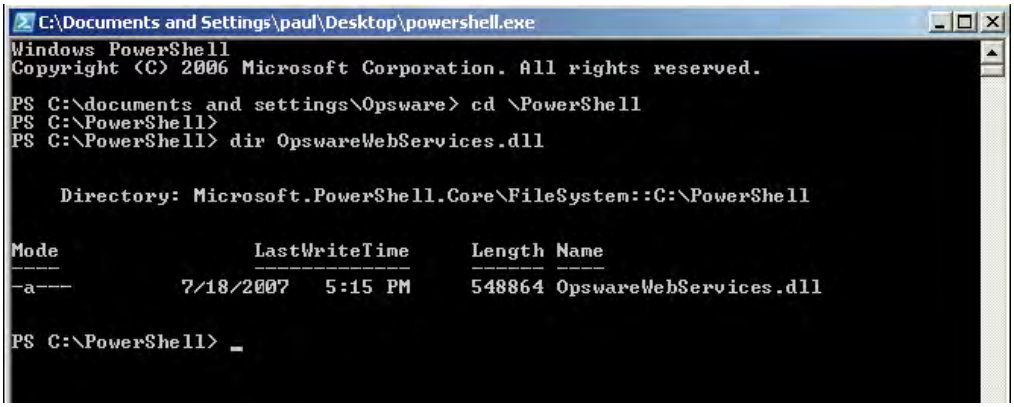


シナリオ4

このシナリオは、Windows PowerShell環境で使用可能なすべてのタイプのSAオブジェクトを調べる方法を示します。

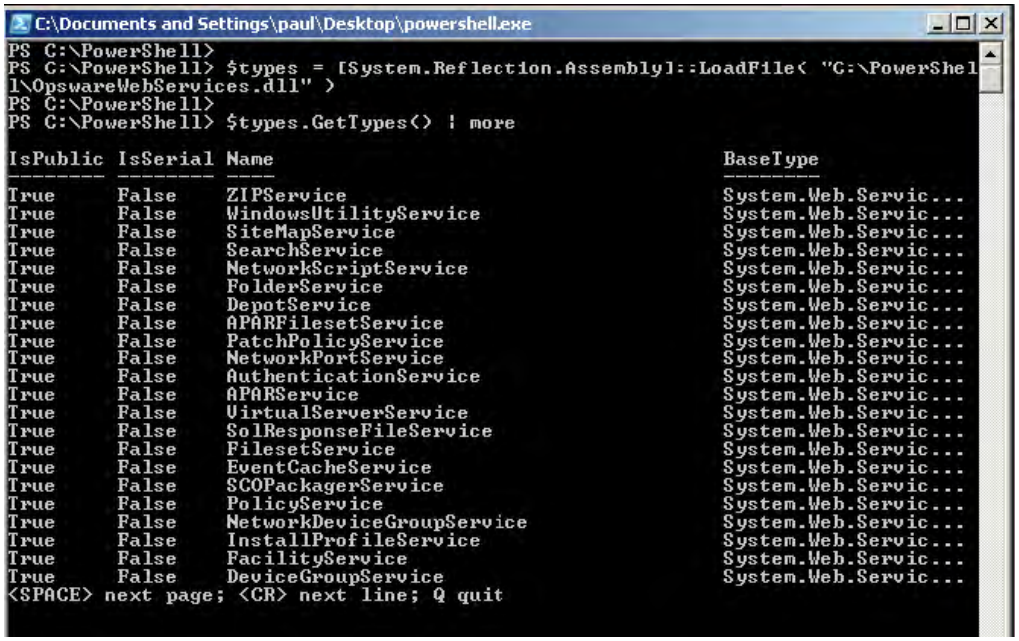
1 PowerShell SASプロバイダーとコマンドレットがある.NETアセンブリを見つけます。詳細については図18を参照してください。

図 18 PowerShell SASプロバイダーとコマンドレットがある.NETアセンブリを見つける



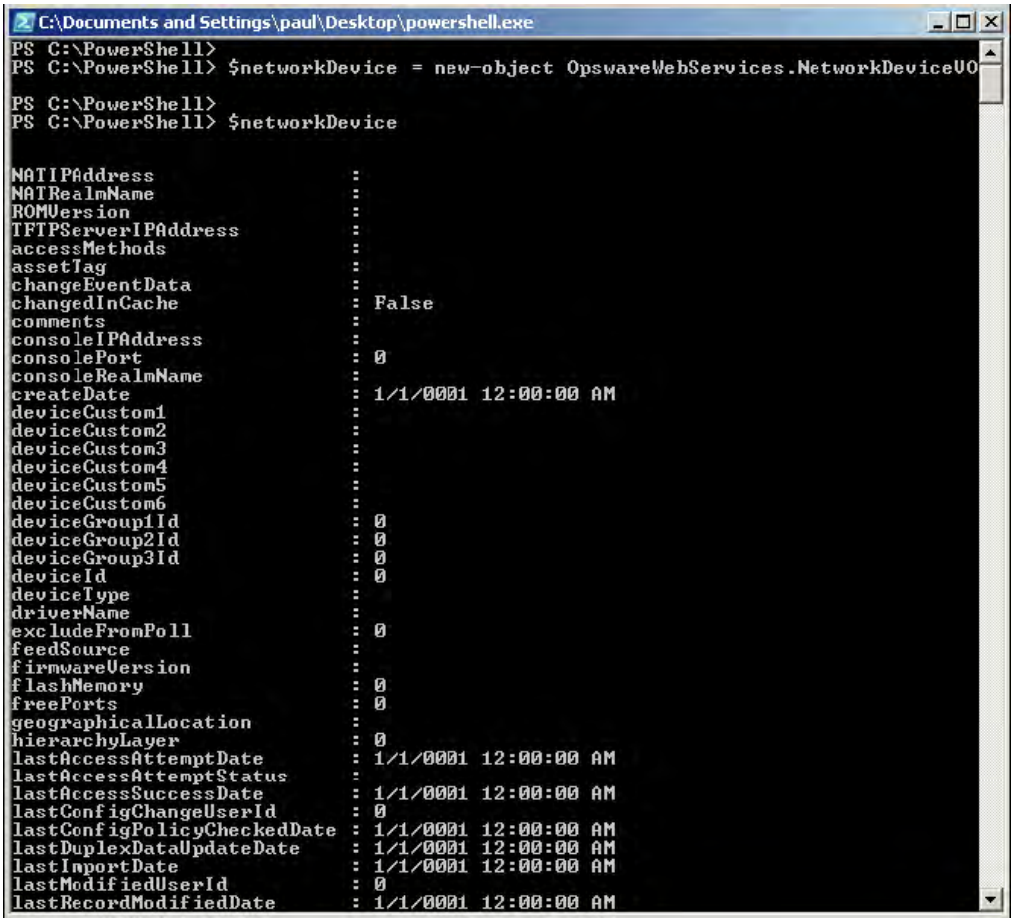
2 .NET Reflectionを使用して.NETアセンブリをロードし、ロードされたタイプを調べます。Windows PowerShell環境で使用可能なすべてのSAタイプが表示されます。詳細については図19を参照してください。

図 19 .NETアセンブリをロードしてタイプを調べる



3 NetworkDeviceVOのインスタンスを作成します。これは作成直後のNetworkDeviceVOで、PowerShell環境でスクリプトやレポートの作成に使用できるネットワークデバイスのすべての属性を示します。詳細については図20を参照してください。

図 20　NetworkDeviceVOのインスタンスの作成



第 7 章　Java RMIクライアント

Java RMIクライアントの概要

Java Remote Invocation (RMI) クライアントを使えば、ネットワークを通じてSA コアにアクセスできるサーバーから、SA APIのメソッドを呼び出すことができます。クライアントを実行しているサーバーは、SAコアや管理対象サーバーでなくてもかまいません。クライアントは、コアに接続する際に、エンドユーザーがSAクライアントにログオンする場合と同様に、SAのユーザー名とパスワードを指定します。ユーザーが属するグループによって、クライアントから使用できるSAリソースとタスクが決まります。

この章の内容は、SAの基礎とJavaプログラミング言語に関する知識があるソフトウェア開発者を対象としています。

Java RMIクライアントの設定

SA API用のJava RMIクライアントを開発する前に、次の手順を実行します。

1. SAコアを開発環境にインストールします。プロダクションコアは使用しないでください。
2. Java RMIクライアントを構築して実行する開発サーバーを用意します。
3. 開発サーバーにJava SE 6 SDKをインストールします。
4. 開発サーバーから、OCCコンポーネントを実行しているSAコアサーバーへのネットワーク接続が可能であることを確認します。
5. SAコアサーバーから開発サーバーに`opswclient.jar`ファイルをダウンロードします。
   `opswclient.jar`ファイルには、SA API用のJava RMIスタブが含まれています。Java RMIクライアントをコンパイルして実行する際に、`opswclient.jar`を`classpath`オプションに指定します。

   `opswclient.jar`をダウンロードするには、次のURLを指定します。ここで、OCCホストはOCCコンポーネントを実行しているコアサーバーです。

   `https://OCC`**ホスト**`:/twister/opswclient.jar`

また、`spinclient-latest.jar`ファイルと`opsware_common-latest.jar`ファイルも必要です。このファイルは、次のディレクトリで稼働しているSAコアから取得できます。

`/opt/opsware/twist/lib/`

さらに、このサンプルをコンパイルして実行するには、`.jar`ファイルを`classpath`パラメーターに追加する必要があります。

Java RMIの例

ここでは、GetServerInfoという名前の単純なJava RMIクライアントを示します。

GetServerInfoクライアントは、コマンドライン引数に指定したホスト名の全部または一部から、管理対象サーバーを検索します。見つかった管理対象サーバーのそれぞれに対して、クライアントはサーバー名、管理IPアドレス、OSバージョンを出力します。

GetServerInfoクライアントは、次のステップを実行します。

1 SAに接続します。

```
OpswareClient.connect("https", host, (short)port,
userPasswd[0], userPasswd[1], true);
```

2 ServerServiceインタフェースへの参照を取得します。

```
serverSvc = (ServerService)OpswareClient.getService
(ServerService.class);
```

3 ServerServiceのメソッドを呼び出します。

```
ServerRef[] serverRefs = serverSvc.findServerRefs(filter);
. . .
ServerVO[] serverVOs = serverSvc.getServerVOs(serverRefs);
. . .
System.out.println(serverVOs[i].getName());
```

GetServerInfoの例のコンパイルと実行

例をコンパイルして実行する前に、次の作業を実行します。

1 opsware_common-latest.jar、spinclient-latest.jar、opswclient.jar の各ファイルを取得します (Java RMIクライアントの設定 (117ページ) を参照してください)。

2 SA_Platform_Developer_Guide_examples.zipファイルをHPセルフソルブ (http://support.openview.hp.com/selfsolve/document/KM00417670/binary/SA_10_Platform_Developer_Guide_examples.html) からダウンロードし、GetServerInfo.javaファイルを解凍します。

3 クライアントをコンパイルするには、opsware_common-latest.jar、spinclient-latest.jar、opswclient.jarの各ファイルをclasspathパラメーターで指定します。

```
javac -classpath :path/opswclient.jar:path/opsware_common-latest.jar:path/
spinclient-latest.jar GetServerInfo.java
```

4 クライアントを実行するには、次のコマンドを入力します。ターゲットには、SAで管理するサーバーの名前の全部または一部を指定します (WindowsでのJava classpathの区切り文字は";"です)。

```
java -classpath  .:path/opswclient.jar:path/opsware_common-latest.jar:path/
spinclient-latest.jar \
GetServerInfo [オプション]ターゲット
```

次の例で、GetServerInfoはホストc44 (OCCコアコンポーネントが実行されているホスト) のポート443でSAに接続します。プログラムは、ホスト名に文字列opswを含む管理対象サーバーの情報を表示します。

```
java -classpath .:/home/jdoe/opswclient.jar:/home/jdoe/opsware_common-
latest.jar:/home/jdoe/spinclient-latest.jar \
GetServerInfo --host c44.dev.example.com --port 443 opsw
```

5 プロンプトに応じてSAのユーザー名とパスワードを入力します。SAユーザーは、コマンドラインに指定されたターゲットに一致するサーバーに対する読み取りアクセス権を持つ必要があります。

第 8 章　Webサービスクライアント

Webサービスクライアントの概要

SA APIはWebサービスをサポートします。Webサービスとは、SOAP (Simple Object Access Protocol) やWSDL (Web Services Definition Language) などのオープンな業界標準に基づくプログラミング環境です。Web サービスクライアントは、PerlやC#などのさまざまなプログラミング言語を使用して (これについてはこの章の後の方で説明します)、またはMicrosoft Visual Studio .NETやBEA WebLogic WorkshopなどのWebサービス対応の開発環境を使用して作成できます。

この章の内容は、SAの基礎とWebサービス開発に関する知識があるソフトウェア開発者を対象としています。

このリリースで提供されているプログラム言語のバインド

SAのこのリリースには、C#用のWebサービスクライアントスタブが付属しています。Perlで作成したWebサービスクライアントにはクライアントスタブは不要です。

このリリースには、JavaおよびPython用のWebサービスクライアントスタブは付属していません。ただし、前の各章で説明したように、JavaクライアントはRMIを通じて、PythonクライアントはPytwistを通じてSA APIにアクセスできます。

サービスの場所とWSDLのURL

クライアントは次の構文のURLでWebサービスにアクセスします。ここで、ホストはOCCコアコンポーネントを実行しているサーバーであり、ポートはHTTPSプロキシのポートです (デフォルトのプロキシポートは443です)。パッケージ名は、サービスが属するJavaライブラリに対応します。

```
https://ホスト:ポート/osapi/パッケージ名/Webサービス名
```

WSDLファイルは、次の構文のURLにあります。

```
https://ホスト:ポート/osapi/パッケージ名/Webサービス名?WSDL
```

たとえば、次のURLはFolderServiceの場所とWSDLを表します。

```
https://occ.c38.example.com:443/osapi/com/opsware/folder/FolderService

https://occ.c39.example.com:443/osapi/com/opsware/folder/FolderService?wsdl
```

SOAP バインドスタイルはRPC (リモートプロシージャーコール) であり、トランスポートプロトコルはHTTPSです。

Webサービスクライアントのセキュリティ

SA APIの他のクライアントと同様、WebサービスクライアントはSAの操作を実行するために認証され、承認される必要があります。クライアントとSAコア内のWebサービスコンポーネントとの間の通信は暗号化されます。アクセスは、OCCコアコンポーネントのHTTPSプロキシポートを通じて接続するHTTPSクライアントに限定されています(デフォルトのポートは443です)。

オーバーロードされた操作

SA APIにはオーバーロードされた操作がありますが、WSDL 2.0仕様はオーバーロードをサポートしません。SA APIのオーバーロードされた操作は、Webサービスからは単一の操作として公開されます。

Javaインタフェースのサポート

SA APIはJavaインタフェースを使用しますが、Webサービスはインタフェースをサポートしません。回避策として、WSDLファイルはインタフェースを`xsd:anyType`にマッピングします。C#などのオブジェクト指向プログラミング言語でクライアントをコーディングする場合、インタフェースを返すAPIメソッドの戻り値型は、具象クラスにキャストする必要があります。インタフェースの配列は`Object[]`に変換されます。配列メンバーの固有の型は、シリアル化/逆シリアル化の際に保持されます。C#コードの例については、インタフェース戻り値型の処理 (133ページ) を参照してください。

サポートされないデータ型

SA APIで使用されている次のデータ型は、SOAPではサポートされません。

```
java.util.Properties
com.opsware.common.ModifiableMap
com.opsware.acm.ValueSet
com.opsware.swmgmt.PolicyOverrideFilter
```

Webサービスから除外されたメソッド

次のSA APIメソッドは、サポートされないデータ型をパラメーターまたは戻り値型として使用します。このため、これらはWebサービスの操作としては公開されていません。

```
com.opsware.custattr.CustomAttribute.getCustAttrs
com.opsware.custattr.CustomAttribute.setCustAttrs
com.opsware.custattr.CustomField.getCustomFields
com.opsware.custattr.CustomField.setCustomFields
com.opsware.pkg.Patch.getPolicyOverrideRefs
```

java.util.Mapの部分的サポート

Axisは`java.util.Map`を、キーと値のペアの集合である`apachesoap:Map`に変換します。.NETでは、この変換は動作しません。たとえば、C#クライアントは、キーと値のペアの空の配列を受け取ります。一方、PerlのSoap::Liteではこの変換が動作します。したがって、`java.util.Map`を使用するSA APIは、Webサービスの操作として使用できます。

次のメソッドは、`java.util.Map`をパラメーターまたは戻り値型として使用します。

```
com.opsware.acm.GroupConfigurable.getApplicationInstances
com.opsware.acm.ServerConfigurable.getCustAttrsWithRC
```

```
com.opsware.compliance.sco.CMLSnapshot.getValueSet
com.opsware.compliance.sco.CMLSnapshot.setValueSet
com.opsware.compliance.sco.SnapshotResultService.remediateCMLSnapshot
com.opsware.custattr.VirtualColumnVO.getConfigInfo
com.opsware.custattr.VirtualColumnVO.setConfigInfo
```

サポートされないデータ型を使用するVOのメソッド

次のVOのメソッドは、サポートされないデータ型をパラメーターまたは戻り値型として使用します。

```
com.opsware.acm.ApplicationInstanceVO.getValueset
com.opsware.acm.ApplicationInstanceVO.setValueset
com.opsware.acm.ConfigurableVO.getValueset
com.opsware.acm.ConfigurableVO.setValueset
com.opsware.virtualization.VirtualConfigNode.getProperties
com.opsware.virtualization.VirtualConfigNode.setProperties
com.opsware.virtualization.VirtualServerConfig.getProperties
com.opsware.virtualization.VirtualServerConfig.setProperties
```

VOの作成または更新の際のsetDirtyAtrributesの呼び出し

Web サービスクライアントは、サービスに対して create または update メソッドを呼び出す前に、setDirtyAttributesを呼び出す必要があります。setDirtyAttributesメソッドは、createまたはupdateの呼び出しによって設定する必要があるVOの属性 (フィールド) を明示的にマークします。setDirtyAttributesで指定される属性名は、大文字と小文字が区別されます。

たとえば、FolderVOオブジェクトのdescription属性を変更する場合、次のコードのようにupdateを呼び出す前にsetDirtyAttributesを呼び出します。

```
// fs is FolderService
FolderVO folderVO = fs.getFolderVO(folderRef);
folderVO.setDescription("credit card processing");
folderVO.setDirtyAttributes(new String[]{"description"});
fs.update(folderRef, folderVO, true, true);
```

Webサービスクライアントがsetdirtyattributesを呼び出す必要があるのは、AxisがXMLからのXMLオブジェクトを逆シリアル化する方法に原因があります。setDirtyAttributesを呼び出さない場合、Axisは読み取り専用の属性を含めてVOのすべての属性に対してsetterを呼び出すため、ReadOnlyExceptionが発生します。

SA WebサービスAPI 2.2との互換性

SA Web サービス API 2.2 は、本書で記述している SA API とは互換性がありません。メソッドのシグネチャ、サービス、WSDL、ポートバインドが異なります。新しくWebサービスクライアントを作成する場合は、SA WebサービスAPI 2.2ではなくSA APIを使用してください。

Perl Webサービスクライアント

ここでは、SA APIにアクセスするPerl Webサービスクライアントを作成するための詳細な手順とサンプルコードを示します。

Perlクライアントに必要なソフトウェア

開発環境には次のPerlモジュールが必要です。

- Crypt-SSLeay-0.51
- IO-Socket-SSL-0.95
- Net_SSLeay.pm-1.25
- HTML-Parser-3.35
- MIME-Base64-3.01
- URI-1.30
- libwww-perl-5.76
- SOAP-Lite-0.65_6

Perlのバージョンによっては、新しいバージョンのモジュールが必要になる場合があります。

Perlデモプログラムの実行

デモプログラムを実行するには、次の手順を実行します。

1. SA_Platform_Developer_Guide_examples.zipファイルをHPセルフソルブ(http://support.openview.hp.com/selfsolve/document/KM00417670/binary/SA_10_Platform_Developer_Guide_examples.html) からダウンロードし、`uapisample.pl`ファイルを解凍します。
2. `uapisample.pl`ファイルを編集し、コードに直接書き込まれている`host`、`username`、`password`、および`serverID`などのオブジェクトIDの値を変更します。
3. `uapisample.pl`を実行します。

「`Certificate Verify Failed`」というエラーが発生する場合は、サンプルファイルの次の行をコメント解除し、証明書ファイルの有効なパスを指定します。

```
#$ENV{HTTPS_CA_FILE} = "path_to/opsware-ca.crt";
```

証明書ファイルは、SAコアの次の場所にあります。

```
/var/opt/opsware/crypto/twist/opsware-ca.crt
```

Perlサンプルコード

次のコードは、先ほどダウンロードしたZIPファイルに含まれるPerlプログラムuapisample.plの一部です。

サービスURIの設定

```
# Construct the URI for the service.
#
my $username = "integration";
my $password = "integration";
my $protocol = "https";
my $host = "occ.c38.dev.example.com";
my $port = "443";
my $contextUri = "osapi/com/opsware/";
my $folderServiceName = "folder/FolderService";
my $folderUri   = "http://www.example.com/" .$contextUri .
$folderServiceName;

# Create a proxy to the FolderService.
#
my $folderProxy = $protocol ."://".$username ."::".$password ."@" .$host ."::".
$port ."/" .$contextUri .$folderServiceName;
```

新しいサービスの開始

```
my $folderPort = SOAP::Lite
    -> uri($folderUri)
    -> proxy($folderProxy);
```

サービスメソッドの呼び出し

```
my $root = $folderPort->getRoot()->result();
print 'Got root folder:' .$root->{'name'} ."\n";

# Alternative:
my $root = $folderPort->SOAP::getRoot();
print 'Got root folder:' .$root->{'name'} ."\n";
```

VOの取得

```
$rootVO = $folderPort->getFolderVO(SOAP::Data->name('self')
->value(\SOAP::Data->name('id')->type('long')->value(0)))
->result();

# The preceding call to getFolderVO does not pass a FolderRef
# parameter.If a method such as FolderService.remove accepts a
# FolderRef parameter, use the following code:
#
```

```
my $folderToBeRemoved = SOAP::Data-name('self')
->attr({ 'xmlns:ns_fs' => 'http://folder.example.com/FolderService'}) -
>type('ns_fs:FolderRef')->value(\SOAP::Data->name('id')->type('long') -
>value(123456));
$folderPort-remove($folderToBeRemoved);

# To see the Perl representation of the returned VO, you can use
# the Dumper method.This will help you understand how to
# construct the dirty attributes of a VO for a create or update
# method.
#
use Data::Dumper;
print Dumper($folderVO);
```

配列の取得

```
# Construct $folder, the FolderRef before getting the array.
#
my $folder = SOAP::Data-name('self') -attr({ 'xmlns:ns_fs' = 'http://
folder.example.com'}) -type('ns_fs:FolderRef')-value(\SOAP::Data-
name('id')-type('long') -value($root-{'id'}));

# The getChildren method returns an array of FNodeReference
# objects.
#
my $children = $folderPort-getChildren($folder, SOAP::Data-name('type')-
type('string')-value(''))-result();

foreach $child (@{$children}){
   print 'Get child:' .$child-{'name'} ."\n";
}
```

オブジェクト配列の構築

```
# For a function that takes an object array as a parameter,
# such the getVOs method, take the following approach:
# First, construct the Array object elements individually
# and put them in an array.
#
my @refs = [];
foreach my $ref (@{$myRefs}){
     # Assume myRefs was returned from a previous
     # Web Services call.
     my $object = SOAP::Data->name('FacilityRef')
                      ->value(\SOAP::Data->name('id')
                              ->type('long')
                              ->value($ref->{'id'}
                       )
                  )->attr({ 'xmlns:facility' => 'http://locality.example.com'})
                   ->type('facility:FacilityRef');
     push @refs, $object;
}
```

```
# Second, construct an Array Object and put the array in it.
#
my $selves = SOAP::Data->name("selves" =>
                \SOAP::Data->name("element" => @refs)-
>type("facility:FacilityRef"))
             ->attr({ 'xmlns:facility' => 'http://locality.example.com'})
             ->type("facility:ArrayOfFacilityRef");
```

VOの更新または作成

```
# This example updates the description attribute of a ServerVO.
#
my $serverID = 40038;
my $server = SOAP::Data->name('self')->value(\SOAP::Data->name('id')-
>type('long')->value($serverID));

# Don't forget to set dirtyAttributes for the attributes
# you want to update.You also need dirtyAttributes for
# create methods that pass a VO.
#
my @dirtyAttrs = ('description');
my $serverVO = SOAP::Data->name('vo') ->attr({ 'xmlns:ns_ss' => 'http://
server.example.com'}) ->value(\SOAP::Data->value( SOAP::Data-
>name('description')->value('PERL_UPDATE_DESC')->type('string'), SOAP::Data-
>name('logChange')->value('false')->type('boolean'), SOAP::Data-
>name('dirtyAttributes' => \SOAP::Data->name("element" => @dirtyAttrs)-
>type("string")) ->type("ns_ss:ArrayOf_soapenc_string"), ));

my $force = SOAP::Data->name('force')->value('true')->type('boolean');
my $refetch = SOAP::Data-name('refetch')-value('true')-type('boolean');

# Call the update method.
#
print 'Invoking method serverWSPort.update...', "\n";
my $updatedServerVO = $serverWSPort-update(
                        $server,
                        $serverVO,
                        $force,
                        $refetch)->result();
print "New description:", $updatedServerVO->{'description'}, "\n";
```

SOAPフォールトの処理

```
# Make sure that you turn off on_fault subroutine in the
# "use SOAP::Lite ..." statement.
#
# The fault member of a SOAP return will be set if the Web
# Service call throws an exception.
# The following code tries to get a folder that does not exist:
#
my $testVO = $folderPort->getFolderVO(SOAP::Data->name('self') -
>value(\SOAP::Data->name('id')->type('long')->value(123456)));
```

```
if($testVO-fault){
    print $testVO->faultstring ."\n";
    # This will print the error msg.
    print "ExceptionName:" . getExceptionName($testVO) ."\n"; # A NotFoundException
should be displayed here
    # The code that deals with the error goes here....
}
...
# The following subroutine extracts the exception name from the
# returned faultdetail.
#
sub getExceptionName {
    my $fault = shift; #get the fault object
    if($fault->faultdetail->{'fault'}){
        return ref($fault->faultdetail->{'fault'});
    }
}
...
# As shown in the preceding code, it's easier to handle SOAP
# faults if you execute functions like this:
#
#     my $data = $port->function(...);
# Not like this:
#     $port->SOAP::function(...);
#     $port->function(...)->result;
```

Webサービス用のPerlオブジェクトの構築

Webサービス操作を呼び出す前に、Perlクライアントは入力パラメーターに必要なデータ構造を作成する必要があります。データ構造の作成に必要な情報は、APIドキュメント (javadocs) とサービスのWSDLファイルに記述されています。ここに示すPerlコードの例は、getServerVO操作に対する入力パラメーターの構築方法を示します。コードの後の手順は、APIドキュメントとWSDLファイルから入力パラメーターに関する情報を入手する方法を示します。

getServerVOを呼び出すためのソースコード

次のPerlコードは、入力パラメーター selfを設定してから、getServerVO操作を呼び出します。この呼び出しは、IDが12345の管理対象サーバーのVO (値オブジェクト) を取得します。

```
# Create a top-level SOAP::Data object that represents the
# with the name self.
#
$self = SOAP::Data->name('self')

# The namespace corresponds to the schema of the data type
# of the SOAP:Data object.The name chosen (ns_ss) is
# arbitrary.
#
$self->attr({'xmlns:ns_ss' =>
'http://server.example.com/ServerService'});

# Specify the type (ServerRef) for the parameter self, using the
# name of the namespace from the preceding statement.
```

```
#
$self->type('ns_ss:ServerRef');

# Create the value for the parameter.The value is a pointer
# to a SOAP::Data object.The number 12345 is the SA ID of # a managed server.
#
my $id = SOAP::Data-name(id)-type(long)-value(12345);

# From the self object, point to the value.
#
$self-value(\$id);

# Finally, call getServerVO:
#
my $data = $serverPort-getServerVO($self);
if($data-fault){
   # Handle exceptions here ...
}
else{
   my $serverVO = $data-result;
}
...
```

getServerVOの設定のための情報の入手方法

getServerVOの呼び出しのコードを作成するために必要な情報を入手するには、次の手順を実行します。

1 ブラウザーで、次のURLのAPIドキュメント (javadocs) にアクセスします。

```
https://OCCホスト:1032/twister/docs/index.html
```

OCCホストは、コマンドセンターコンポーネントを実行しているコアサーバーのIPアドレスまたはホスト名です。(Twisterでメソッドを呼び出す手順については、APIドキュメントとTwister (21ページ) を参照してください)。

2 APIドキュメントを調べて、メソッドの入力パラメーターと戻り値を確認します。

getServerVOメソッドは、インタフェースcom.opsware.server.ServerServiceで定義されています。次のメソッドシグネチャから、getServerVOがServerRefをパラメーターとして受け取り、ServerVOを返すことがわかります。

```
public ServerVO getServerVO(ServerRef self)
              throws java.rmi.RemoteException,
                     NotFoundException,
                     AuthorizationException
```

3 ブラウザーで次のURLを指定して、ServerServiceのWSDLファイルを開きます。

```
https://occ_host/osapi/com/opsware/server/ServerService?wsdl
```

4 WSDLファイルで、ServerServiceの名前空間を見つけます。

```
<schema targetNamespace="http://server.example.com" xmlns="http://
www.w3.org/2001/XM
LSchema">
```

次のPerlステートメント (前記のコードリスト内に存在) が名前空間を指定しています。

```
$self->attr({'xmlns:ns_ss =>
'http://server.example.com/ServerService'});
```

5 WSDLファイルで、getServerVO操作を見つけ、入力メッセージ名getServerVORequestを確認します。

```
<wsdl:operation name="getServerVO" parameterOrder="self">
  <wsdl:input message="impl:getServerVORequest" name="getServerVORequest"/>
  <wsdl:output message="impl:getServerVOResponse" name="getServerVOResponse"/>
  <wsdl:fault message="impl:NotFoundException" name="NotFoundException"/>
  <wsdl:fault message="impl:AuthorizationException" name="AuthorizationException"/>
</wsdl:operation>
```

6 WSDLファイルで、getServerVORequestメッセージを見つけます。

```
<wsdl:message name="getServerVORequest">
  <wsdl:part name="self" type="impl:ServerRef"/>
</wsdl:message>
```

getServerVORequestメッセージ要素は、getServerVOの入力パラメーターの名前 (self) と型 (ServerRef) を定義します。次のPerlステートメントは、ServerRefを指定します。

```
$self->type('ns_ss:ServerRef');
```

7 WSDLファイルで、ServerRefのcomplexTypeを見つけます。

```
<complexType name="ServerRef">
 <complexContent>
  <extension base="tns1:ObjRef">
   <sequence>
    <element name="secureResourceTypeName" nillable="true"
type="soapenc:string"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

ServerRefはObjRefを拡張しています。

8 WSDLファイルで、ObjRefのcomplexTypeを見つけます。

```
<complexType abstract="true" name="ObjRef">
 <sequence>
  <element name="id" type="xsd:long"/>
  <element name="idAsLong" nillable="true" type="soapenc:long"/>
  <element name="name" nillable="true" type="soapenc:string"/>
 </sequence>
</complexType>
```

ObjRefで、名前 (id) と型 (long) を確認します。これらのデータ型は、次のPerlステートメントで指定されています。

```
my $id = SOAP::Data-name(id)-type(long)-value(12345);
```

C# Webサービスクライアント

ここでは、SA APIにアクセスするC# Webサービスクライアントを作成するための詳細な手順とサンプルコードを示します。

C#クライアントに必要なソフトウェア

C# Webサービスクライアントを開発するには、開発環境に次のソフトウェアが必要です。

- Microsoft .NET Framework SDKバージョン1.1
- SA API用C#クライアントスタブ

C#クライアントスタブの入手方法

SAには、各サービス用のスタブファイル (例、`FolderService.cs` が用意されます。すべてのスタブは、同じ名前空間`OpswareWebServices`を持ちます。スタブの他に、SAには`ServerRef`などの共有クラスを含む`shared.cs`ファイルもあります。

C#スタブを含むZIPファイルを入手するには、次のURLを指定します。ここで、OCCホストはOCCコンポーネントを実行しているコアサーバーです。

`https://OCC`**ホスト**`:1032/twister/opswcsharpclient.zip`

サービスとオブジェクトで定義されている定数は、C#スタブでは定義されていません。定数に関する情報を得るには、APIドキュメント (javadocs) を使用します。詳細については定数のフィールド値 (22ページ) を参照してください。

C#スタブドキュメントへのアクセス

Ndocで生成されたリファレンスドキュメントが、コンパイル済みのWindowsヘルプファイルとして利用できます。これはC#スタブと同じZIPファイルに含まれています(NDocは、.NETアセンブリと、C#コンパイラーが出力したXML ドキュメントファイルから、コードドキュメントを生成します)。このリファレンスドキュメントには、クラス階層とメンバーメソッドのシグネチャに関する構文情報 (説明はなし) が記述されています。説明については、APIドキュメントとTwister (21ページ) に示す方法で対応するjavadocsを参照してください。

C#デモプログラムのビルド

デモプログラムをビルドするには、次の手順を実行します。

1. SA_Platform_Developer_Guide_examples.zipファイルをHPセルフソルブ (http://support.openview.hp.com/selfsolve/document/KM00417670/binary/SA_10_Platform_Developer_Guide_examples.html) からダウンロードし、次のファイルを解凍します。
   - `App.config` - アプリケーション設定
   - `WebServicesDemo.cs` - サービスメソッドを呼び出すクライアントコード
   - `MyCertificateValidation.cs` - 証明書検証クラス
2. 次のディレクトリを作成します。

   `C:\wsapi`

3 Visual Studio.2008 スタートページで、[新規プロジェクト] を選択し、次の値を持つプロジェクトを作成します。

- プロジェクトタイプ: Visual C#プロジェクト
- テンプレート: コンソールアプリケーション
- 名前: WSAPIDemo
- 場所: C:\wsapi

この操作により、新しいディレクトリC:\wsapi\WSAPIDemoが作成され、その下にいくつかのファイルが置かれます。

4 新しいプロジェクトで、オブジェクトリストからデフォルトの Program ファイルと AssemblyInfo.cs ファイルを削除します。

5 手順1で入手したファイルをC:\wsapi\WSAPIDemoディレクトリにコピーします。

6 C#クライアントスタブの入手方法 (131ページ) で指定したURLからクライアントスタブをダウンロードします。

7 C#クライアントスタブをC:\wsapi\WSAPIDemoディレクトリにコピーします。

8 前の2つのステップでコピーしたファイルをWSAPIDemoプロジェクトに追加します。

- Visual Studioで、[プロジェクト] メニューから [既存項目の追加] を選択します。
- C:\wsapi\WSAPIDemoディレクトリに移動し、デモファイルをすべて選択します (.csと .config)。

9 System.Web.Services.dllへの参照を追加します。

- Visual Studioで、[プロジェクト] メニューから [参照の追加] を選択します。
- .NETタグの下で、System.Web.Services.dllという名前のコンポーネントを参照します。
- System.Web.Services.dllをクリックし、[選択]、[OK] をクリックします。

10 プロジェクトの作成時に別のテンプレートを使用した場合、System、System.XML、System.Dataへの参照を追加することが必要な場合があります。[プロジェクトの参照] をチェックして、これらの参照を追加する必要があるかどうかを判断します。

11 App.configファイルで、username、password、hostの値と、serverIDなどの直接書き込まれているオブジェクトIDを変更します。

12 Visual Studioで、[ビルド] メニューから [WSAPIDemoのビルド] を選択します。

C#デモプログラムの実行

デモプログラムを実行するには、次の手順を実行します。

1 Visual Studio 2008のコマンドプロンプトを開きます。

[スタート] > [すべてのプログラム] > [Microsoft Visual Studio 2008] > [Visual Studio ツール] > [Visual Studio 2008コマンドプロンプト]

2 次のディレクトリに移動します。

```
C:\wsapi\WSAPIDemo\bin\Debug
```

3 次のコマンドを入力します。

```
WSAPIDemo.exe
```

C#サンプルコード

次のコードは、先ほどダウンロードしたZIPファイルに含まれるC#プログラムWebServicesDemo.csの一部です。

証明書処理の設定

```
# This setup is required just once for the client.
#
ServicePointManager.CertificatePolicy = new MyCertificateValidation();
```

Assign the URL Prefix

```
# This is the URL prefix for all services.
#
wsdlUrlPrefix = protocol + "://" + host + ":"+ port + "/" + contextUri + "/";
```

サービスの開始

```
FolderService fs = new FolderService();
fs.Url = wsdlUrlPrefix + "com.opsware.folder/FolderService";
```

サービスメソッドの呼び出し

```
FolderRef root = fs.getRoot();
FolderVO vo = fs.getFolderVO(root);
```

インタフェース戻り値型の処理

```
# In the API, FolderVO.getMembers returns an array of
# FNodeReference interfaces, but Web Services does not support
# interfaces.In the C# stub, the return type of
# FolderVO.members is Object[].If a returned Object type will
# be used as a parameter that must be a specific type, then you
# must cast it to that type.For example, the following code
# casts elements of the returned array to FolderRef as
# appropriate.
#
Object[] members = vo.members;
for(int i=0;i<members.Length;i++)
{
Console.WriteLine("Got object:" + members[i].GetType().FullName + " --> " +
((ObjRef)members[i]).name);
   if(members[i] is FolderRef) {
   Console.WriteLine("I am a FolderRef:" +
   ((FolderRef)members[i]).name);
   }
}
```

VOの更新または作成

```
# When updating a VO, the changed attributes must be set in
# dirtyAttributes.(The VO passed to a create method has
# the same requirement.)
#
# Note:If you update a VO that was returned from a service
# method invocation, such as getFolderVO, then you must
# set the logChange attribute of the VO to false:
#    vo.logChange = false;
#
# The following code changes the name of a folder.
#
Console.WriteLine("Changing name from " + vo.name +
" to yo_csharp.");
vo.name = "yo_csharp";
vo.dirtyAttributes = new String[]{"name"};
# Manually set dirty fields being changed.
#
vo = fs.update(folder, vo, true, true);
Console.WriteLine("Folder name changed to:" + vo.name);
```

例外の処理

```
# .NET converts Web Services faults into SoapExceptions
# without trying to deserialize them into application
# exceptions first.As a result, your code cannot catch
# application exceptions.As a workaround, the C# stubs
# provided by SA include SOAPExceptionParser,
# a class that enables you to get information from
# SOAPExceptions.The following code shows how to get the
# exception name and error message by calling the getDetail
# method of SOAPExceptionParser.
#
try{
// Try to get a non-existent folder here.
} catch(SoapException e){
    SoapExceptionDetail detail =
    SoapExceptionParser.getDetail(e);
    Console.WriteLine("SoapExceptionDetail.name:" +
    detail.exceptionName);
    Console.WriteLine("SoapExceptionDetail.msg:" +
    detail.message);
...
}
```

C#でのパスワードセキュリティ

FolderService メソッドは、ユーザー名とパスワードのペアを App.config ファイルから読み取ります。このメソッドの例を次に示します。

```
User user = new User();
user.username = "user";
user.password = "password";
FolderService fs = new FolderService();
fs.Url = wsdlUrlPrefix + "com.opsware.folder/FolderService";
fs.user = user;
```

パスワードを平文でApp.configファイルに記録したくない場合は、SecureUserクラスを使用してパスワードを暗号化できます。SecureUserクラスは、.NET 2.0のC# SecureStringを使用します。パスワードは暗号化されてSecureStringに記憶されます。また、getPassword()メソッドは内部からしか見えません。SecureUserは静的クラスなので、ユーザー名とパスワードの設定は1回だけ、あるいはユーザーを切り替えたときだけ行えばすみます。各サービスは、ユーザー名とパスワードをまずSecureUserから取得し、その後に過去との互換性のためにuserメンバー変数、その後にApp.configから取得します。SecureUserは、パスワードをStringまたはSecureStringで受け取ります。どちらの場合も、クライアントはSecureUser.setUser()メソッドに渡されたパスワード変数をクリーンアップする責任があります。

パスワードはいずれかの時点でメモリ上の通常のC#文字列に変換する必要があります。これは次のガベージコレクションが発生するまで解放されません。SecureUser で保証されるのは、内部的なパスワードの記憶がセキュアであることだけです。

次の例は、ユーザー名とパスワードをセキュアに設定する方法を示します。

```
SecureString passwd = new SecureString();
passwd.AppendChar('p');
passwd.AppendChar('a');
passwd.AppendChar('s');
passwd.AppendChar('s');
passwd.AppendChar('w');
passwd.AppendChar('d');
SecureUser.setUser("username", passwd); // that's it, no need to set up user for each
service.
passwd.Dispose(); // resets passwd and frees up memory so no copy remains from caller.
```

第 9 章　プラグ可能チェック

プラグ可能チェックの概要

SAの監査と修復機能では、SA管理対象サーバーに対するコンプライアンス情報の定義と監視が可能です。コンプライアンス標準は常に進化し続けるため、SAでは、特殊化されたカスタムチェックやポリシーを作成したり、SAに付属するチェックやポリシーを拡張したりできます。プラグ可能チェックとは、1つまたは複数の監査ポリシーに属する監査ルールです。コマンドライン環境でプラグ可能チェックを作成し、チェックをアップロードしてから、SAクライアントで監査ポリシーに追加します。

この章の内容は、XMLと、SAの監査と修復機能に関する知識があるソフトウェア開発者を対象としています。

プラグ可能チェックの設定

プラグ可能チェックを開発する前に、次の手順を実行します。

1. SAコアを開発環境にインストールします。プロダクションコアは使用しないでください。
2. エージェントがインストールされているサーバーに、OCLI 1.0 をインストールします。OCLI 1.0 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。

プラグ可能チェックのチュートリアル

このチュートリアルでは、HelloWorld Checkという名前のプラグ可能チェックを作成する方法を示します。この単純なチェックは、`/var/tmp/helloworld`ファイルがUnix管理対象サーバーに存在することを確認します。このファイルが存在しない場合、プラグ可能チェックの修復スクリプトによってファイルが作成されます。

HelloWorld Checkを開発するには、次の手順を実行します。

1. プラグ可能チェックの設定 (137ページ) の手順を実行します。OCLI 1.0をインストールするサーバーが、このチュートリアルの開発サーバーになります。
2. HelloWorld Checkのサンプルコードは、APIコードの例を収録したZIPファイルに含まれています。SA_Platform_Developer_Guide_examples.zipファイルをHPセルフソルブ (http://support.openview.hp.com/selfsolve/document/KM00417670/binary/SA_10_Platform_Developer_Guide_examples.html) からダウンロードします。
3. 前のステップでダウンロードしたファイルを展開し、`pluggable_checks/helloworld`ディレクトリに次のファイルがあることを確認します。

```
config.xml
gethelloworld.py
sethelloworld.py
```

HelloWorld Checkは、これら3つのファイルから構成されます。config.xmlファイルは構成ファイルです。gethelloworld.pyは、監査を実行するPythonスクリプトです。sethelloworld.pyは、修復を実行するPythonスクリプトです。次の手順では、これらのファイルをZIPファイルにパッケージ化し、このZIPファイルをSAにインポートします。

4 開発サーバーで、展開されたhelloworldファイルを次の例のように作業ディレクトリにコピーします。

```
cd /home/jdoe/dev
mkdir helloworld
cd helloworld
cp 展開先/pluggable_checks/helloworld/* .
```

5 グローバルに一意のID (GUID) を取得します。各プラグ可能チェックにGUIDが必要です。有効なGUIDを得るには、次のいずれかの方法を使用します。

— 次のようなWebサイトにログオンします。

http://kruithof.xs4all.nl/uuid/uuidgen（**英語サイト**）

— 次のサイトから無料のWindowsツールguidgenをダウンロードします。

http://www.microsoft.com/downloads/details.aspx?FamilyID=94551F58-484F-4A8C-BB39-ADB270833AFC&displaylang=en（**英語サイト**）

GUIDをプログラムで作成する場合、コードはRFC4122 (http://www.ietf.org/rfc/rfc4122.txt) (英語サイト) に準拠する必要があります。

6 テキストエディターで、次のようにconfig.xmlファイルにGUIDを挿入します。

```
<checkGUID>6c7ed38c-d8d6-11db-8314-0800200c9a66</checkGUID>
```

このチュートリアルで変更するconfig.xmlの要素はこれだけです。

7 テキストエディターで、GUIDを変更したconfig.xmlを保存します。

テキストエディターは開いたままにしておきます。このチュートリアル全体を通じて、config.xmlのさまざまな要素を参照することにより、HelloWorld CheckのPythonスクリプトとSAクライアント表示フィールドに要素がどのように対応するかを調べます。

8 config.xmlファイルで、次の要素を確認します。これらはHelloWorld Checkの監査 (get) および修復 (set) スクリプトに対応します。

```
<!-- The name of the script that performs the check.-->
<checkGetScriptName>gethelloworld.py</checkGetScriptName>


<!-- The name of the script that remediates the audit.-->
<checkSetScriptName>sethelloworld.py</checkSetScriptName>


<!-- The exit code of the gethelloworld.py script will be checked.-->
<checkReturnType>EXITCODE</checkReturnType>


<!-- A string argument is passed to gethelloworld.py.-->
<checkGetArgumentType>STRING</checkGetArgumentType>


<!-- The default argument for gethelloworld.py is the name of the file the script
is checking for.-->
```

```
<checkGetArgumentDefaultValue>/var/tmp/helloworld
</checkGetArgumentDefaultValue>


<!-- If the helloworld file exists, the exit code of gethelloworld.py is 0.-->
<checkSuccessExitCodeValue>0</checkSuccessExitCodeValue>


<!-- If the helloworld file does not exist, the exit code of gethelloworld.py is
1.-->
<checkSuccessExitCodeValue>1</checkSuccessExitCodeValue>
```

9 gethelloworld.pyスクリプトを調べます。これはファイル /var/tmp/helloworldの存在をチェックすことで監査を実行します。このチュートリアルでは、このスクリプトを編集する必要はありません。このチュートリアルの後の方 (手順29 (143ページ)) で、SAクライアントで監査を実行するときに、このスクリプトが管理対象サーバー上で実行されます。

config.xmlのcheckGetArgumentDefaultValueの値が示すように、このスクリプトのデフォルト引数は文字列/var/tmp/helloworldです。スクリプトの終了コード (result) は、<checkSuccessExitCodes>に指定された値に対応します。

gethelloworld.pyスクリプトのソースコードを次に示します。

```
import sys
import os
import string

if __name__ == "__main__":

    if len(sys.argv) != 2:
        sys.stderr.write("No argument found!Please enter a
            file name!\n")
        sys.exit(220)

    filename = sys.argv[1]
    if os.path.isfile(filename) or os.path.isdir(filename):
        result = 0
    else:
        result = 1

    sys.stderr.write("Debugging:Found result %s\n"
       % result)
    sys.stdout.write("%s\n" % result)

    sys.exit(result)
```

10 次に、修復スクリプトsethelloworld.pyを調べます。これは /var/tmp/helloworldファイルを作成します。このスクリプトは、手順34 (143ページ) で監査を修復する場合に、管理対象サーバーで実行されます。このチュートリアルではスクリプトは変更しません。

sethelloworld.pyのソースコードを次に示します。

```
import sys
import os
import string
```

```
if __name__ == "__main__":

    if len(sys.argv) != 2:
        sys.stderr.write("No argument found!
           Please enter a file name!\n")
        sys.exit(220)

    filename = sys.argv[1]
    if os.path.isfile(filename) or os.path.isdir(filename):
        # Do nothing because the file already exists.
        pass
    else:
        try:
           fd = open(filename, "w")
           fd.write(" ")
           fd.close()
        except:
            sys.stderr.write("Could not open file %s for
               writing!\n" % filename)
            sys.exit(220)

    # Exit successfully with a 0 exit code.
    sys.stderr.write("Successfully created file\n")
    sys.exit(0)
```

11 HelloWorld Checkをパッケージ化します。

HelloWorldプラグ可能チェックをパッケージ化するには、次のように作業ディレクトリの内容を1つのZIPファイルにアーカイブします。

```
cd /home/jdoe/dev/helloworld
zip ../helloworld.zip *
```

12 ZIPファイルに2つのPythonスクリプトと`config.xml`ファイルが含まれることを確認するために、次の`unzip`コマンドを入力します。

```
unzip -t ../helloworld.zip
 testing: config.xml      OK
 testing: gethelloworld.py     OK
 testing: sethelloworld.py     OK
No errors detected in compressed data of ../helloworld.zip.
```

13 OCLI 1.0の`ouploadコマンド`で、プラグ可能チェックをSAにインポートします。

```
ouploadコマンド -C"Customer Independent" \
-t"Server Configuration Check" \
--forceoverwrite --old -O"SunOS 5.8" ../helloworld.zip
```

注: プラットフォームオプション (`-O`) は、UnixおよびLinuxチェックの場合はすべて`SunOS 5.8`です。Windowsチェックの場合、プラットフォームオプションは`Windows 2003`です。

`ouploadコマンド`が正常に実行されなかった場合、OCLI 1.0の正しいバージョンがインストールされており、PATH環境変数が正しく設定されており、`login`ファイルが環境に含まれていることを確認してください。これらの要件の詳細については、『SAユーザーガイド: Server Automation』の「OCLI 1.0」を参照してください。

14 SAクライアントを開きます。

この後の手順では、新しい監査を作成して、ouploadコマンドでインポートしたHelloWorld Checkに追加します。

15 [**ツール**] メニューで [**キャッシュの更新**] を選択します。

16 [ナビゲーション] ペインで、[**ライブラリ**] > [**タイプ別**] > [**監査と修復**] > [**監査**] > [**Unix**] を選択します。

17 [**アクション**] メニューで、[**新規**] を選択します。

18 [監査] ウィンドウの [プロパティ] ペインの [名前] フィールドに、「HelloWorld Audit」と入力します。

19 [ビュー] ペインで [**ルール**] > [**ファイルシステム**] を選択します。

図21に示すように、内容ペインでHelloWorld Checkが [監査に使用可能] の下に表示されます。

図 21　ファイルシステムに対するルール内のHelloWorld Check

20 config.xmlファイルで、次の要素を確認します。これらは図21に表示される情報に関連します。

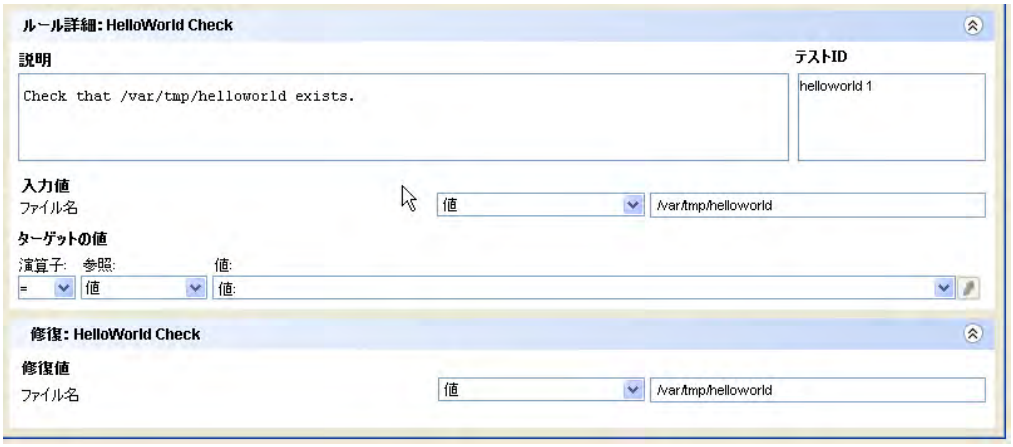
```
<!-- The check name is the rule name shown in the SAクライアント.-->
<checkName>HelloWorld Check</checkName>


<!-- The category corresponds to the rule hierarchy displayed by the SAクライアント
.-->
<checkCategory>File System|My Custom Checks</checkCategory>
```

21 SAクライアントの [監査] ウィンドウの [監査に使用可能] の下で、HelloWorld Checkを選択し、プラス記号をクリックします。

図22に示すように、内容ペインにHelloWorld Checkの詳細が表示されます。

図 22　HelloWorld Checkのルール詳細



22 config.xmlファイルで、次の要素を確認します。これらは図22のルール詳細に表示される情報に関連します。

```
<!-- The following value appears under Description in the Rule Details of the SAクライアント.-->
<checkDefaultDescription>
Check that /var/tmp/helloworld exists.
</checkDefaultDescription>


<!-- The following element corresponds to the Test ID in the SAクライアント.-->
<checkTestID>helloworld 1</checkTestID>


<!-- This label is under Input Values in the SAクライアント.-->
<checkGetArgumentDefaultLabel>File Name
</checkGetArgumentDefaultLabel>


<!-- The default argument to the gethelloworld.py script also appears under Input Values in the SAクライアント.-->
<checkGetArgumentDefaultValue>/var/tmp/helloworld
</checkGetArgumentDefaultValue>
```

23 SAクライアントのビューペインで [ターゲット] を選択します。

次のステップでは、HelloWorld Auditにターゲットサーバーを追加します。後の手順で、gethelloworld.pyスクリプトとsethelloworld.pyスクリプトがターゲットサーバーに対して実行されます。

24 内容ペインで [**追加**] をクリックします。

25 [サーバーの選択] ウィンドウで、サーバーにドリルダウンして [**OK**] をクリックします。

26 [監査] ウィンドウで [**ファイル**] > [**保存**] を選択します。

この時点で、HelloWorld AuditはHelloWorld Check (ルール) を含み、ターゲットサーバーと関連付けられています。

27 [監査] ウィンドウで、[**アクション**] メニューから [**監査の実行**] を選択します。

28 [監査の実行] タスクのウィンドウを順次実行します。

29 [監査の実行] ウィンドウで [**ジョブの開始**] をクリックします。

このアクションは、ターゲットサーバーに対してgethelloworld.pyスクリプトを実行するジョブを起動します。

30 ジョブが完了したら、[**結果の表示**] をクリックします。

31 [監査結果] ウィンドウの [ビュー] ペインで [ポリシールール (1)] を選択します。

32 [監査結果] ウィンドウの 内容ペインでHelloWorld Checkを選択します。

図23に示すように、[差異の詳細] ウィンドウが表示されます。

図 23　HelloWorld Checkの差異の詳細

33 config.xmlファイルで、次の要素を確認します。これらは図23の [差異の詳細] ウィンドウに表示される情報に関連します。

```
<!-- The following value appears as the Policy Value in the Difference Details
window.-->
<checkSuccessExitCodeDefaultDisplayName>
File exists</checkSuccessExitCodeDefaultDisplayName>


<!-- The next value appears as the Actual Value in the same window.-->
<checkSuccessExitCodeDefaultDisplayName>
File does not exist</checkSuccessExitCodeDefaultDisplayName>
```

34 ターゲットサーバー上に/var/tmp/helloworldファイルを作成するには、[差異] ウィンドウで [**修復**] をクリックします。

このアクションは、sethelloworld.pyスクリプトを実行します。詳細については、『SAユーザーガイド: 監査とコンプライアンス』を参照してください。

監査と修復の概要

Sarbanes-Oxley (SoX) 法、Information Technology Infrastructure Library (ITIL)、ISO20000のために、サーバー構成のコンプライアンスを維持することが緊急の課題になっています。SAの監査と修復機能は、コンプライアンスの問題に対処するための整理されたポリシーのセットを提供します。グラフィカルインタフェースにより、指定したサーバーに対して簡単に監査を選択して実行し、プロフェッショナル標準にどの程度適合するかを判定できます。

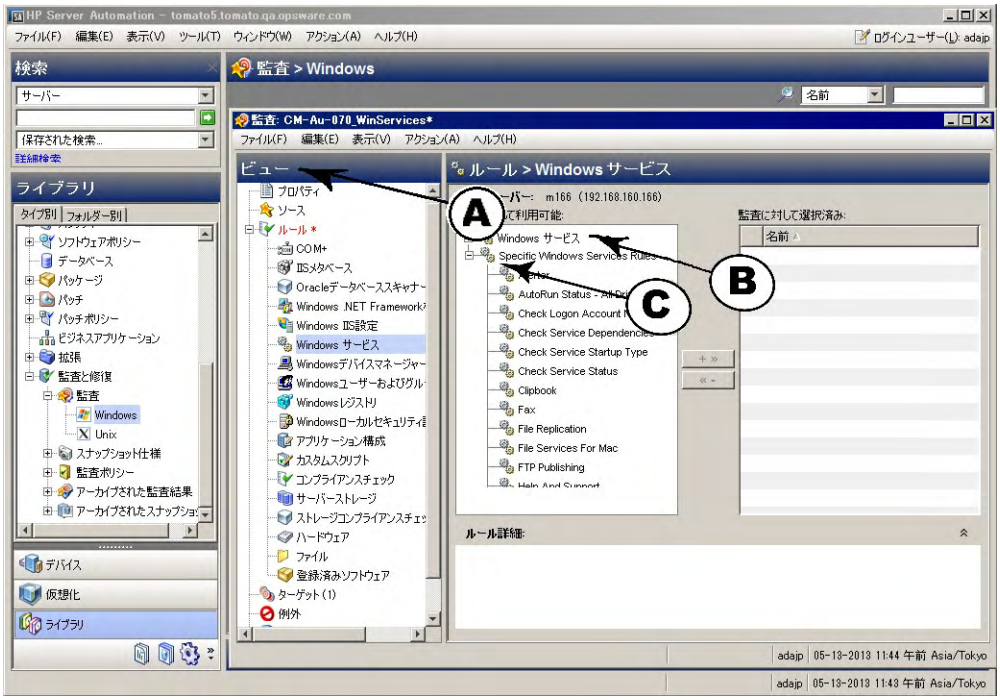
監査と修復は、システム管理を容易にする役割も果たします。たとえば、自社チームが開発したデータベースサーバーやミドルウェアアプリケーションなどのアプリケーションを実行するサーバーのクラスを監視するとします。アプリケーションを実行するサーバーを構成して監視する際に、構成の理想的な状態を記録したリストを作成します。このリストには、ファイル、ディレクトリ、ネットワーク共有のアクセス権などが含まれます。

これらの構成を定義した監査を作成し、アプリケーションをインストールした後でサーバーを監査できます。監査結果は、アプリケーションがインストールされ、基準に従って正しく構成されたかどうかを確認します。構成がコンプライアンス違反の場合、問題を解決するためのアドホック監査を作成できます。監査結果がエラーを示す場合、理想的な構成に合わせてサーバーを修復できます。構成変更が実務で動作するように、監査を実行するスケジュールを設定し、完了時に通知を受け取ることができます。

図24は監査を選択するためのウィンドウです。マークの説明を次に示します。

- **A**: [ビュー] パネルに表示されたカテゴリは、SAの変更不可能な機能または変更可能なプラグ可能チェックを持ちます。
- **B**: これはWindowsサービスを扱うSA機能を指します。
- **C**: これはWindowsサービスを対象とするプラグ可能チェックを示します。

図 24　Windowsサービスの監査ルール



各チェックは1つのルールを評価します。複数のチェックを1つのポリシーにまとめることができます。

SAの監査と修復機能には、多数の標準チェックが付属しています。ほとんどの監査は、必要なチェックを選択することで実行できます。選択できる監査は、開発者がさらに多くのチェックを設計し、コーディングしてテストし、HP Live Networkを通じてシステムに追加することで増え続けていきます。これらのチェックは、完成したポリシーとしてインポートできます。

ただし、各企業にはそれぞれ独自の課題やリソースがあるので、コンプライアンス判定のための監査に必要な基準が、SAの監査と修復フレームワーク内には存在しないことも考えられます。このために、独自のカスタムプラグ可能チェックを作成する方法が用意されています。

監査と修復機能は、個別のルールによって、SAで管理されているサーバーのコンプライアンス状態を評価します。この機能は、ルールに定義された必要な構成状態に一致しないサーバーを修復することもできます。これらのルールには、さまざまなサーバーパラメーター、レジストリ値、ファイルアクセス権、アプリケーション構成、ファイルの存在、COM+オブジェクトなどが含まれます。

Windows 環境では、Web サーバールールをアプリケーション構成に指定することもできます。これは、Microsoft Internet Information Services (IIS) のWebサーバー構成ファイルUrlScan.iniに基づきます。SAは、値の一部または全部を特定の構成ファイルと比較し、ファイルから必要な要素を選択して、これらの値または構成ファイルエントリが存在することを確認します。詳細については、『SAユーザーガイド: アプリケーション構成』を参照してください。

SAには、設計済みの監査ルールが多数付属しています。各ルールは、サーバーまたはサーバーグループの構成の必要な状態を定義します。一部のルールは値に基づいており、比較演算子 (<、>、==、!=、次の値を含む、など)、1つの値または値のセット、1つ以上のチェック (監査対象アイテムの状態を評価するために使用されるコードを記述したもの) が用意されています。比較データは、コンプライアンスまたは非コンプライアンスを決定します。チェックが修復をサポートする場合、ルールには修復値が含まれる場合もあります。

ルールは1つのチェックから構成されます。カスタム内容オブジェクトを使用して、プラグ可能チェックという形式で新しい機能を作成できます。また、関連するプラグ可能チェックを、使用に便利なように監査ポリシーにまとめることもできます。

プラグ可能チェックの作成

プラグ可能チェックとは、管理対象サーバーにダウンロードされ、監査と修復フレームワークによって実行されるコードです。チェックを使用して、監査と修復のネイティブのプロパティを拡張し、特殊化された機能を追加することができます。各プラグ可能チェックには、カスタマイズされたconfig.xmlファイルと、監査対象の機能とconfig.xmlファイルに指定された値とを比較する1つ以上のスクリプトが含まれます。プラグ可能チェックには、監査対象サーバーの指定された変数をconfig.xmlファイルに指定された値に設定するスクリプトが含まれる場合もあります。プラグ可能チェックのスクリプトの作成には、Python、Visual Basic Scripting (VBS)、BAT、シェルスクリプトが使用できます。プラグ可能チェックはZIPアーカイブにパッケージ化されます。

CISチェックのほとんどは、CISベンチマークから直接変換されたものです。
詳細については**http://www.cisecurity.org** (英語サイト) を参照してください。

ほとんどのタイプのチェックは、次のいずれかのカテゴリに分類されます。

- Windowsレジストリチェック
- Unixサービスチェック

- ユーザーチェック (パスワードまたはシャドウファイルの情報を使用する場合があります)

プラグ可能チェックのガイドライン

サーバーのメンテナンスを容易にするため、次のガイドラインに従ってください。

- 新しいプラグ可能チェックを作成する際には、名前に十分注意します。チェックの目的を示す名前を付け、スペースの代わりに下線を使用します。わかりやすい名前の例としては、`Users_Without_Password_Expiration`などが挙げられます。これは、サーバーが数百以上のチェックを処理するような場合に、目的のチェックを簡単に見つけられるようにするためです。
- 汎用的なチェックを作成します。これにより、数行のコードを変更するだけで、同じ実行タイプの別のチェックを容易に作成できます。たとえば、CIS2k3 Windows サービスチェックでは、ほとんどの場合 1 行のコードを変更するだけで、新しいサービスに対する新しいチェックを作成できます。
- 監査 (get) および修復 (set) スクリプトの名前を付ける際には、ディレクトリ名のスペースと下線を削除し、getまたはsetを必要に応じて先頭に付けます。たとえば、`getUsersWithoutPasswordExpiration.sh`は適切な監査ファイル名の例です。カスタムチェックを使用する予定なのが自分だけであっても、この規則は守ってください。
- エラーチェックに注意してください。スクリプトでエラーが発生すると、予期しない戻り値のために監査結果がコンプライアンス違反として報告される可能性があります。予期しないエラーや例外をトラップし、トラブルシューティングを容易にするため、それに関する情報をstdoutまたはstderrに出力します。
- チェックはできる限り単純な真または偽の2値のケースに変換します。
- 特定のベンチマークケースだけでなく、それと相補的なケースも常に処理するようにします。たとえば、「サービス X の無効化」プラグ可能チェックを作成する際には、ほとんどのコードを再使用して、「サービス X の有効化」も容易に作成できます。そうしておけば、後で逆の条件のテストが必要になった場合に役立ちます。
- できる限り、フレームワークによって定義された標準の終了コードを使用します。次に示すのがそうです。

  ```
  EXIT_FAILURE=220
  EXIT_ERR_USAGE=221
  EXIT_ERR_INVALID_OS=222
  ```

- ブール値型のチェックで無効または有効を返す場合は、0が無効、1が有効を表すようにします。
- 各プラグ可能チェックはZIPアーカイブにパッケージ化します。1つのファイルシステムディレクトリには、表23に示すファイルが含まれます。

表 23　プラグ可能チェックの内容

ファイル名	説明
`config.xml`	(必須) このプラグ可能チェックがどのように実行され、戻り、最終的にコンプライアンスまたは非コンプライアンスを報告するかを定義するXML構成ファイル。

表 23　(続き)プラグ可能チェックの内容

ファイル名	説明
get**名前**.{py | sh | BAT | vbs}	(必須) Python、VBS、BAT、シェルのいずれかで作成された監査スクリプト。監査対象のオブジェクトを評価し、config.xmlの定義に従ってテキストと終了コードを返します。
set**名前**.{py | sh | BAT | vbs}	(オプション) Python、VBS、BAT、シェルのいずれかで作成された修復スクリプト。監査スクリプトでチェックされた条件を修復します。
その他のコード、スクリプト、ライブラリ	(オプション) 監査スクリプトまたは修復スクリプトから使用される補助または補足用のスクリプト。

監査スクリプトと修復スクリプトのファイル名はgetおよびsetで始まる必要はありませんが、この規則に従えばメンテナンスが容易になります。

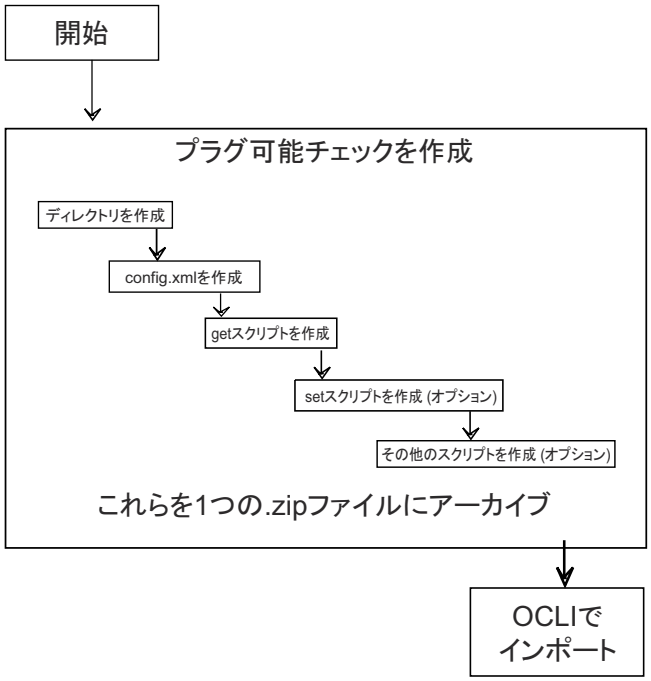
次の例は、プラグ可能チェックのディレクトリ構造を示します。

```
./check_name/
./check_name/config.xml
./check_name/getcheckname.py
./check_name/setcheckname.py
```

プラグ可能チェックの開発プロセス

図25には、コマンドライン環境で行われる開発プロセスの概要を示します。

図 25　開発プロセス



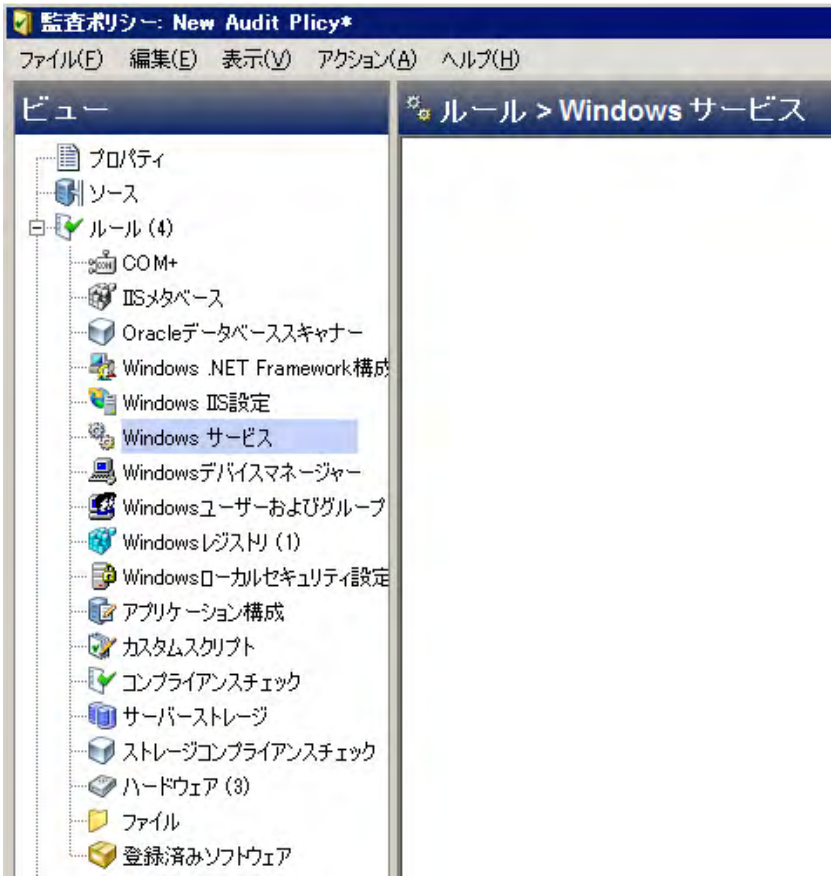
プラグ可能チェック構成 (config.xml)

config.xmlファイルはプラグ可能チェックの仕様ファイルであり、このチェックがSAクライアントに表示される方法、デフォルト値、比較の値タイプ、チェックのカテゴリを制御する要素が含まれています。たとえば、config.xmlファイルの次の要素は、SAクライアントでのプラグ可能チェックのルールカテゴリを決定します。

```
<checkCategory>Windows Services</checkCategory>
```

標準カテゴリは、図26に示すように、それぞれ固有のアイコンで表され、ハードウェア、ソフトウェア、オペレーティングシステム、ユーザーとグループ、ファイルシステムなどがあります。

図 26　ルール階層内のプラグ可能チェックカテゴリ



次のリストは、config.xmlファイルのテンプレートを示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE checkConfiguration SYSTEM "check.dtd">
<checkConfiguration version="1.0">
<checkName>$CHECKNAME</checkName>
<checkGUID>$CHECKGUID</checkGUID>
<checkDefaultDescription>$CHECKDESCRIPTION</checkDefaultDescription>
<checkRemediationDefaultDescription> $CHECKREMEDIATIONDESCRIPTION </
checkRemediationDefaultDescription>
<checkGetScriptName>$GETSCRIPTNAME</checkGetScriptName>
<checkGetScriptType>PY</checkGetScriptType><!-- Or SH for shell, BAT for Bat, VBS
for Visual Basic -->
<checkSetScriptName>$SETSCRIPTNAME</checkSetScriptName><!-- Optional -->
<checkSetScriptType>PY</checkSetScriptType><!-- Optional -->
<checkVersion>32b.0-1.0</checkVersion>
```

```
<checkReturnType>$RETURNTYPE</checkReturnType>   <!-- EXITCODE, STRING, or
NUMBER -->
<checkTestIDs>
<checkTestID>$CHECKTESTID</checkTestID> <!-- Optional -->
</checkTestIDs>
<checkPlatformTypes>
<checkPlatform>$PLATFORMTYPE</checkPlatform>  <!-- Currently Unix or Windows -->
</checkPlatformTypes>
<checkCategories>
<checkCategory>$CATEGORY</checkCategory>  <!-- Top-level GUI category -->
</checkCategories>
<checkGetArguments>  <!-- All arguments are optional -->
<checkGetArgument>
<checkGetArgumentType>$GETARGTYPE</checkGetArgumentType> <!-- STRING or NUMBER -->
     <checkGetArgumentDefaultLabel>$GETDEFAULTLABEL</checkGetArgumentDefaultLabel>
            <checkGetArgumentDefaultDescription>$GETDEFAULTDESCRIPTION</
checkGetArgumentDefaultDescription>
          <checkGetArgumentDefaultValue>$GETDEFAULTVALUE</
checkGetArgumentDefaultValue>
     </checkGetArgument>
</checkGetArguments>
<checkSetArguments>  <!-- Also optional -->
<checkSetArgument>
<checkSetArgumentType>$SETARGTYPE</checkSetArgumentType>
     <checkSetArgumentDefaultLabel>$SETDEFAULTLABEL</checkSetArgumentDefaultLabel>
            <checkSetArgumentDefaultDescription>$SETDEFAULTDESCRIPTION</
checkSetArgumentDefaultDescription>
     <checkSetArgumentDefaultValue>$SETDEFAULTVALUE</checkSetArgumentDefaultValue>
</checkSetArgument>
</checkSetArguments>
<checkSuccessExitCodes>  <!-- Only for EXITCODE type checks, generally at least two
entries -->
     <checkSuccessExitCode>
<checkSuccessExitCodeValue>$EXITCODEVALUE</checkSuccessExitCodeValue>
            <checkSuccessExitCodeDefaultDescription>$EXITCODEDESCRIPTION
     </checkSuccessExitCodeDefaultDescription>
            <checkSuccessExitCodeDefaultDisplayName>$EXITCODEDISPLAYNAME
     </checkSuccessExitCodeDefaultDisplayName>
  </checkSuccessExitCode>
</checkSuccessExitCodes>
</checkConfiguration>
```

詳細については、config.xmlファイルの文書型定義 (DTD) (153ページ) を参照してください。

監査 (get) スクリプト

監査スクリプト (getスクリプトとも呼ばれる) は、管理対象サーバーから値を取得するために作成します。スクリプトは、config.xmlファイルの指定に従って、オプションのパラメーター付きで実行されます。スクリプトでEXITCODEチェックを実行する場合、スクリプトの結果がconfig.xmlファイルに指定された終了コードと比較されます。STRINGおよびNUMBER戻り値型チェックの場合、結果はSTDOUTへの出力と比較されます。

監査スクリプトには、定義済みのリターンコードのセットがあります。チェックの config.xml ファイルで追加のリターンコードを定義することもできます。

監査スクリプトは、情報メッセージを表示することができます。メッセージは、監査スクリプトのエラーのトラブルシューティングの際に有用です。次のPython監査スクリプトの例を見てください。

```
import sys
import os
import string

if __name__ == "__main__":

  # If there are get arguments they will be loaded into sys.argv

  # Enter the desired check code here
  # Example:
  #   Looking for file "/usr/bin/ssh"

  if os.path.isfile("/usr/bin/ssh"):
     result = 1
  else:
     result = 0

  # Case A:
  #  If number/string check, the results are grabbed from    #  stdout.
  #   All debugging statements must be sent to stderr so as not
  #   to be picked up.

  sys.stderr.write("Debugging:Found result %s\n" % result)

  sys.stdout.write(result)

  # Case B:
  # If exitcode check, the results are returned by the argument
  #   passed to sys.exit().The exitcodes must match the
  #   ExitCodeValues defined in the config.xml file.

sys.exit(result)
```

修復 (set) スクリプト

修復スクリプト (setスクリプトとも呼ばれる) は、監査スクリプトが終了時に成功を返すように管理対象サーバーを変更するために作成します。スクリプトは、チェックの config.xml ファイルの指定に従って、オプションのパラメーター付きで実行されます。

setスクリプトはオプションであり、対応するgetスクリプトとよく似た性質を持つ場合も、まったく異なる (そしてもっと長い) 場合もあります。

シェルの観点からは、スクリプト自体には、使用されるリターンコードを別として、特別なことは何もありません。ほとんどのチェックは、何らかのデバッグ出力または情報メッセージを表示します。これは通常はユーザーには見えませんが、スクリプトでエラーが発生した場合には、メッセージがトラブルシューティングの役に立つことがあります。

標準的慣行として、setスクリプトは常に少なくとも1つのパラメーターを取るようにします。また、既存のチェックにsetスクリプトを追加した場合、SAクライアントにうまく表示されるようにconfig.xmlファイルを修正するようにしてください。

修復スクリプトが成功した場合は、終了コード0を返すようにします。それ以外の終了コードは、修復操作の失敗を示します。

次のPython setスクリプトの例を見てください。

```
import sys
import os
import string
if __name__ == "__main__":

   # If there are set arguments they will be loaded into
   # sys.argv
   # Enter the desired set code here.Stdout may be used for
   # debugging.
   # Uses exitcode 0 for success, and all other values for
   # failure.
   # enter condition where set script if successful. for this
   # example, use 'if 1'

   if 1:
      sys.exit(0)

   else:
      sys.exit(-1)
```

プラグ可能チェックのその他のコード

プラグ可能チェックには、getまたはsetスクリプト以外のコードが含まれる場合もあります。ライブラリ、実行可能ファイル、他のスクリプトをチェックに追加して、setまたはgetスクリプトから実行時に使用することができます。

ZIPに追加のコードを含めることもできます。

プラグ可能チェックのZIP化

config.xmlファイル、監査 (get) スクリプト、オプションの修復 (set) スクリプトを作成したら、これらのファイルを含むZIPアーカイブを作成します。次のシェル履歴は、UNIX環境での作成プロセスを示します。

```
# ls
 check_name
# cd check_name
# zip ../checkname.zip *
 adding: config.xml
 adding: getcheckname.py
 adding: setcheckname.py
# unzip -t ../checkname.zip
 testing: config.xml      OK
 testing: getcheckname.py     OK
 testing: setcheckname.py     OK
No errors detected in compressed data of ../checkname.zip.
```

プラグ可能チェックのインポート

プラグ可能チェックをSAコアまたはメッシュにインポートするには、『SAコンテンツユーティリティガイド』に記述されているOCLI 1.0ユーティリティを使用します。次のシェル履歴は、Linuxのインポートプロセスの例を示します。

```
# cp checkname.zip /var/tmp/checks
# cd /var/tmp/checks
# cp opsware_32.a.692.0-upload/disk001/packages/Linux/3AS/ocli-32a.2.0.5-linux-3AS .
# chmod 755 ocli-32a.2.0.5-linux-3AS
# ./ocli-32a.2.0.5-linux-3AS
# ../ocli/login.sh
# export PATH=/opt/opsware/bin:$PATH
# oupload -C"Customer Independent" -t"Server Configuration Check" --forceoverwrite --
old -O"SunOS 5.8" プラグ可能チェック名.zip
```

ouploadコマンドでは、"SunOS 5.8" を使用して、チェックがSAクライアントのUnix一般のカテゴリに分類されることを指定します。Windowsカテゴリのチェックを指定するには、"Windows 2003" を使用します。

監査ポリシーの作成

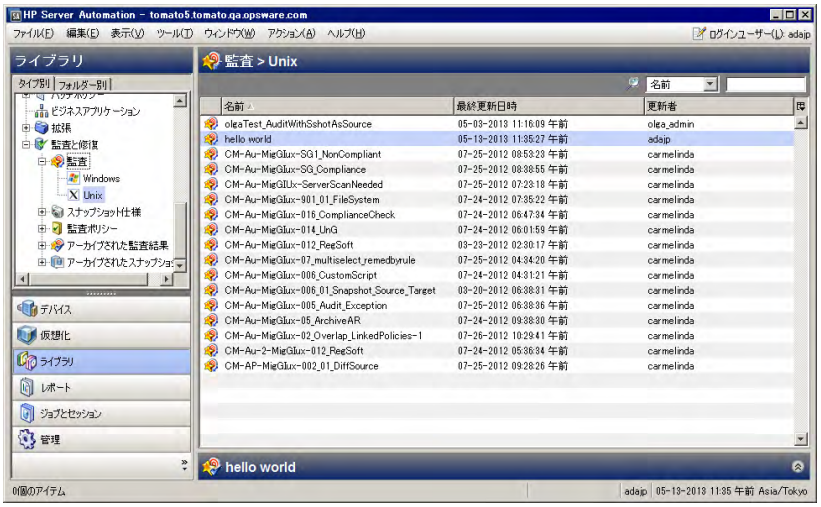
監査ポリシーの作成手順を下の図27に示します。

図 27　監査ポリシーの作成手順

監査ポリシーの作成

監査ポリシーは、ルールから構成されます。各ルールは1つ以上のチェックから構成され、ユーザーが作成したプラグ可能チェックを含むことができます。監査ポリシーとルールの表示、作成、編集は、SAクライアントで行われます。図28に、モデルシステムで使用可能な監査ルールのリストを示します。

図 28　監査ルールのリスト



監査ポリシーの作成に関する詳細については、『SAユーザーガイド: 監査とコンプライアンス』を参照してください。

監査ポリシーのエクスポート

新しい監査ポリシーを別のSAコアに移動するには、現在のコアからポリシーをエクスポートして別のコアにインポートします。このためには、DCML Exchange Tool (DET) コマンドラインユーティリティを使用します。このツールは、新しくインストールしたSAコアに既存コアのポリシーなどのコンテンツを入れるために使用します。この手順の詳細については、『SAコンテンツユーティリティガイド』を参照してください。

config.xmlファイルの文書型定義 (DTD)

このファイルは、SAクライアントの表示名と説明、デフォルト値、チェックコードから返された値に対して実行する比較、これらの値を表示するSAクライアントのカテゴリなどを規定します。

デフォルトのconfig.xmlファイルにあるcheckGetArgumentsとcheckSetArgumentsの2つの要素は、実行時にスクリプトにデータ値を渡すために使用されます。プログラマブルチェックに引数が不要な場合は、これらの要素をconfig.xmlファイルから削除します。

次に示すconfig.xmlのDTDは、SAによって動的に生成されます。

```
<!ELEMENT checkConfiguration (checkName, checkGUID, checkDefaultDescription,
checkRemediationDefaultDescription?, checkGetScriptName?, checkGetScriptType?,
checkSetScriptName?, checkSetScriptType?, checkVersion,
checkAllowRemediationOnFailure?, checkReturnType, checkTestIDs?, checkPlatformTypes,
checkExclusivePlatforms?, checkExcludePlatforms?, checkCategories, checkGetArguments?,
checkSetArguments?, checkComparisonDefaults?, checkCompareValidValues?,
checkSuccessExitCodes?)>
<!ATTLIST checkConfiguration version CDATA #REQUIRED>
<!ELEMENT checkName (#PCDATA)>
<!ELEMENT checkGUID (#PCDATA)>
<!ELEMENT checkDefaultDescription (#PCDATA)>
<!ELEMENT checkRemediationDefaultDescription (#PCDATA)>
<!ELEMENT checkGetScriptName (#PCDATA)>
```

```
<!ELEMENT checkGetScriptType (#PCDATA)>
<!ELEMENT checkSetScriptName (#PCDATA)>
<!ELEMENT checkSetScriptType (#PCDATA)>
<!ELEMENT checkVersion (#PCDATA)>
<!ELEMENT checkAllowRemediationOnFailure (#PCDATA)>
<!ELEMENT checkReturnType (#PCDATA)>
<!ELEMENT checkTestIDs (checkTestID+)>
<!ELEMENT checkTestID (#PCDATA)>
<!ELEMENT checkPlatformTypes (checkPlatform+)>
<!ELEMENT checkPlatform (#PCDATA)>
<!ELEMENT checkExclusivePlatforms (checkExclusivePlatform+)>
<!ELEMENT checkExclusivePlatform (#PCDATA)>
<!ELEMENT checkExcludePlatforms (checkExcludePlatform+)>
<!ELEMENT checkExcludePlatform (#PCDATA)>
<!ELEMENT checkCategories (checkCategory+)>
<!ELEMENT checkCategory (#PCDATA)>
<!ELEMENT checkGetArguments (checkGetArgument+)>
<!ELEMENT checkGetArgument (checkGetArgumentType, checkGetArgumentDefaultLabel,
checkGetArgumentDefaultDescription, checkGetArgumentDefaultValue?,
checkGetArgumentValidValues?)>
<!ELEMENT checkGetArgumentType (#PCDATA)>
<!ELEMENT checkGetArgumentDefaultLabel (#PCDATA)>
<!ELEMENT checkGetArgumentDefaultDescription (#PCDATA)>
<!ELEMENT checkGetArgumentDefaultValue (#PCDATA)>
<!ELEMENT checkGetArgumentValidValues (checkGetArgumentValidValue+)>
<!ELEMENT checkGetArgumentValidValue (checkGetArgumentValidValueItem,
checkGetArgumentValidValueDisplayName)>
<!ELEMENT checkGetArgumentValidValueItem (#PCDATA)>
<!ELEMENT checkGetArgumentValidValueDisplayName (#PCDATA)>
<!ELEMENT checkSetArguments (checkSetArgument+)>
<!ELEMENT checkSetArgument (checkSetArgumentType, checkSetArgumentDefaultLabel,
checkSetArgumentDefaultDescription, checkSetArgumentDefaultValue?,
checkSetArgumentValidValues?)>
<!ATTLIST checkSetArgument populateFromRule CDATA #IMPLIED>
<!ELEMENT checkSetArgumentType (#PCDATA)>
<!ELEMENT checkSetArgumentDefaultLabel (#PCDATA)>
<!ELEMENT checkSetArgumentDefaultDescription (#PCDATA)>
<!ELEMENT checkSetArgumentDefaultValue (#PCDATA)>
<!ELEMENT checkSetArgumentValidValues (checkSetArgumentValidValue+)>
<!ELEMENT checkSetArgumentValidValue (checkSetArgumentValidValueItem,
checkSetArgumentValidValueDisplayName)>
<!ELEMENT checkSetArgumentValidValueItem (#PCDATA)>
<!ELEMENT checkSetArgumentValidValueDisplayName (#PCDATA)>
<!ELEMENT checkComparisonDefaults (checkComparisonDefaultOperator?,
checkComparisonDefaultValues)>
<!ELEMENT checkComparisonDefaultOperator (#PCDATA)>
<!ATTLIST checkComparisonDefaultOperator not CDATA #IMPLIED>
<!ATTLIST checkComparisonDefaultOperator caseInsensitive CDATA #IMPLIED>
<!ELEMENT checkComparisonDefaultValues (checkComparisonDefaultValue+)>
<!ELEMENT checkComparisonDefaultValue (checkComparisonDefaultValueItem,
checkComparisonDefaultValueDisplayName)>
<!ELEMENT checkComparisonDefaultValueItem (#PCDATA)>
<!ELEMENT checkComparisonDefaultValueDisplayName (#PCDATA)>
<!ELEMENT checkCompareValidValues (checkCompareValidValue+)>
```

```
<!ELEMENT checkCompareValidValue (checkCompareValidValueItem,
checkCompareValidValueDisplayName)>
<!ELEMENT checkCompareValidValueItem (#PCDATA)>
<!ELEMENT checkCompareValidValueDisplayName (#PCDATA)>
<!ELEMENT checkSuccessExitCodes (checkSuccessExitCode+)>
<!ELEMENT checkSuccessExitCode (checkSuccessExitCodeValue,
checkSuccessExitCodeDefaultDescription, checkSuccessExitCodeDefaultDisplayName)>
<!ELEMENT checkSuccessExitCodeValue (#PCDATA)>
<!ELEMENT checkSuccessExitCodeDefaultDescription (#PCDATA)>
<!ELEMENT checkSuccessExitCodeDefaultDisplayName (#PCDATA)>
```

次の表は、config.xmlのDTDの要素を示します。

表 24　DTD要素と属性

要素	属性
checkConfiguration version	1.0に設定。監査と修復フレームワークで必要な場合のみ変更します。
checkName	SAクライアントに表示されるチェック/ルールの英語名。
checkGUID	標準のGUID、例: 9500A4AE-EE9E-4383-87F2-BAD7DDC26C59 guidgen Windowsユーティリティで生成するか、Webサイトからダウンロードするか、その他の方法で生成できます。 GUIDは一意である必要があります。そうでないと、プラグ可能チェックをコアにアップロードする際にエラーが発生します。一意のGUIDを持つチェックをアップロードした後では、GUIDを変更することはできません。変更した場合、オリジナルを削除しない限り、再アップロード時に「データベース一意性制約エラー」が発生します。チェックはGUIDによって一意に識別されますが、アップロード時には (ZIP ファイルの) 名前のみで識別されます。
checkDefaultDescription	SA クライアントの説明ボックスに表示されます。改行とHTMLは反映されます。HTMLの場合、HTMLタグは<と>を使用して変換する必要があります。
checkRemediationDefaultDescription	SAクライアントでチェック/ルールの [修復] セクションに表示されます。
checkGetScriptName	getスクリプトのファイル名。 例: getUsersWithoutPasswordExpiration.sh
checkGetScriptType	コードのタイプによって、実行するインタープリターが決まります。getスクリプトとsetスクリプトのタイプとしては、SH、VBS、PY、BATが使用できます。
checkSetScriptName	修復スクリプトのファイル名。

表 24　DTD要素と属性 (続き)

要素	属性
checkSetScriptType	コードのタイプによって、実行するインタープリターが決まります。set (修復スクリプトのタイプとしては、SH、VBS、PY、BAが使用できます。
checkVersion	これはSA とフレームワークのビルド番号に基づきます。例: 32b.0-1.0
checkAllowRemediationOnFailure	スクリプトによっては、get段階でエラーが発生しても、その条件を修復スクリプトで修正できることがあります。この場合、スクリプトがエラーになっても修復を実行することができます。たとえば、レジストリキーが存在されていないことが定義されていなくても、setコードで作成して設定することができます。
checkReturnType	使用可能な値は、EXITCODE、STRING、NUMBERです。 EXITCODE - Wscript.Quit()、exit、returnなどによる標準のスクリプトの戻り。 NUMBER - 監査と修復フレームワークが stdout の出力を捕捉して、数値型として解釈します。 STRING - 監査と修復フレームワークがstdoutの出力を捕捉して、文字列型として解釈します。
checkTestIDs	テストIDのリスト。
checkTestID	CIS、MSFT、NSA、またはその他のポリシーの標準命名法を表示するために用いられます。例: CIS-RHEL 8.4。これは自由形式のフィールドであり、SA クライアントに表示されるので、TON内容と対応するように一貫した名前を付ける必要があります。
checkPlatformTypes	チェックに対する有効なプラットフォームタイプのリスト。
checkPlatform	WINDOWS | UNIX (または両方を個別の要素として)
checkExclusivePlatforms	排他的プラットフォームのリスト。監査と修復は、現在はデフォルトでWindows とUnixの区分を採用していますが、実世界の標準や、オペレーティングシステムの間の制限や違いによっては、この区分では対応できない可能性もあります。監査と修復を、spinから取得されたプラットフォームIDによって指定される任意のプラットフォームに制限することができます。 このパラメーターは、『SA Supported Platforms』ドキュメント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。
checkExclusivePlatform	個別のプラットフォームID。
checkExcludePlatforms	除外されるプラットフォームのリスト。PlatformTypeがUNIXの場合、UNIXセット (すべてのLinux＋すべてのUnix) から除外するプラットフォームIDを指定できます。
checkExcludePlatform	個別のプラットフォームID

表 24　DTD要素と属性 (続き)

要素	属性
checkCategory	チェックが表示されるSAクライアントカテゴリ。現時点では、チェックは1つのカテゴリのみに表示できます。カテゴリが存在しない場合、アップロード時に作成されます。可能な場合は、既存のチェックに対する次の標準カテゴリを使用します。 イベントロギング ファイルシステム オペレーティングシステム オペレーティングシステム|ドメインコントローラー (サブカテゴリ) オペレーティングシステム|ネットワーク (サブカテゴリ) レジストリ サービス ユーザーとグループ
checkGetArgument (checkGetArgumentType, checkGetArgumentDefaultLabel, checkGetArgumentDefaultDescription, checkGetArgumentDefaultValue?, checkGetArgumentValidValues?)	getスクリプトのパラメーターを指定します。
checkGetArgumentType	NUMBER | STRING
checkGetArgumentDefaultLabel	SAクライアントでの入力ボックスまたはドロップダウンの隣のタグ。
checkGetArgumentDefaultDescription	マウスでポイントすると表示される詳細な説明のテキスト。
checkGetArgumentDefaultValue	このgetパラメーターのデフォルト値。
checkGetArgumentValidValue (checkGetArgumentValidValueItem, checkGetArgumentValidValueDisplayName	checkGetArgumentValidValueItem (#PCDATA) checkGetArgumentValidValueDisplayName (#PCDATA)
checkGetArgumentValidValues (checkGetArgumentValidValue+)	(オプション) パラメーターを0/無効、1/有効などに制限する場合に便利です。

表 24　DTD要素と属性 (続き)

要素	属性
checkSetArguments (checkSetArgument+)	checkSetArgument (checkSetArgumentType, checkSetArgumentDefaultLabel, checkSetArgumentDefaultDescription, checkSetArgumentDefaultValue?, checkSetArgumentValidValues?) setArgument要素はGetArgumentsと同じですが、修復/setスクリプト (存在する場合) に対して使用されます。 例外: checkSetArgument populateFromRule - set パラメーターのデフォルト値がconfig.xmlに指定されている場合に、デフォルト値をルールデータから取るかどうか。通常は常にtrueに設定します。
checkSetArgumentType	NUMBER | STRING
checkSetArgumentDefaultLabel	SA クライアントでの入力ボックスまたはドロップダウンの隣のタグ。
checkSetArgumentDefaultDescription	マウスでポイントすると表示される詳細な説明のテキスト。
checkSetArgumentDefaultValue	このsetパラメーターのデフォルト値。
checkSetArgumentValidValues (checkSetArgumentValidValue+)	
checkSetArgumentValidValue (checkSetArgumentValidValue Item, checkSetArgumentValidValue DisplayName)>	checkSetArgumentValidValueItem (#PCDATA)> checkSetArgumentValidValueDisplayName (#PCDATA)> checkSetArgumentValidValueItem (#PCDATA)> checkSetArgumentValidValueDisplayName (#PCDATA)>
checkSetArgumentValidValue Item	(オプション) パラメーターを0/無効、1/有効などに制限する場合に便利です。
checkSetArgumentValidValueDisplayName	
<!ELEMENT checkComparisonDefaults (checkComparisonDefaultOperator?, checkComparisonDefaultValues)>	checkComparisonDefaultOperator not — negation of operator specified, TRUE | FALSE checkComparisonDefaultOperator caseInsensitive - STRING タイプのみで有効。
<!ELEMENT checkComparisonDefaultOperator (#PCDATA)>	比較演算子のデフォルト値のリスト。TON ビルドフレームワーク外部のフィールドまたは開発に便利。
checkComparisonDefaultValues (checkComparisonDefaultValue+)	checkComparisonDefaultValue (checkComparisonDefaultValueItem, checkComparisonDefaultValueDisplayName).

表 24　DTD要素と属性 (続き)

要素	属性
checkComparisonDefaultValueIte	デフォルトの値。コードに渡されます。
checkComparisonDefaultValueDisplayName	SAクライアントに表示される値の表示名。
checkCompareValidValues (checkCompareValidValue+)> checkCompareValidValue (checkCompareValidValueItem, checkCompareValidValueDisplayName)> checkCompareValidValueItem (#PCDATA)> checkCompareValidValueDisplayName (#PCDATA)>	
checkSuccessExitCodes (checkSuccessExitCode+) checkSuccessExitCode (checkSuccessExitCodeValue, checkSuccessExitCodeDefaultDescription, checkSuccessExitCodeDefaultDisplayName)>	checkReturnTypeがEXITCODEの場合、スクリプトが正しく動作した場合の有効な値を定義する必要があります。これは通常、コンプライアンスと非コンプライアンスの期待される値を含みます。ここに指定された値以外が返された場合、スクリプトのエラーと見なされ、SAクライアントとレポートに異なる方法で表示されます。
checkSuccessExitCodeValue	スクリプト完了の値。例: 0 (通常は「無効」)。
checkSuccessExitCodeDefaultDescription	DisplayName/Valueに関する、マウスでポイントすると表示されるテキスト。
checkSuccessExitCodeDefaultDisplayName	この値に関してユーザーに表示される値またはテキスト。 例: Disabled。

付録 A　検索フィルターの構文

フィルターの文法

検索フィルターは、findServerRefsなどのメソッドのパラメーターです。検索フィルターの式を使うと、SAオブジェクト (サーバーやフォルダーなど) への参照を、オブジェクト属性の値に基づいて取得することができます。検索フィルターの形式的構文は次のとおりです。

```
<フィルタ>                    := (<式結合>)+

<式結合>                      ::= <式リストオープン> <結合>
    (<式>)+ <式リストクローズ>

<式>                          ::= 式オープン> 属性>
    <一般区切り記号> <演算子>< 一般区切り記号>
    <値リスト 式クローズ>

<属性>                        ::= <リソースフィールド>
<voメンバー >                 ::= <テキスト>
<リソースフィールド>          ::= <テキスト>
<値リスト> .                  ::= (<二重引用符> <テキスト> <二重引用符>)* | (<数値>)*

<テキスト>                    ::= [a-z] [A-Z] [0-9]
<数値>                        ::= [0-9] [.]

<結合>                        ::= <OR結合> |
    <AND結合>
<OR結合>                      ::= '|'
<AND結合>                     ::= '&'
<式リストオープン>            ::= '('
<式リストクローズ>            ::= ')'
<式オープン>                  ::= '(' | '{'
<式クローズ>                  ::= '(' | '}'
<一般区切り記号>              ::= <空白>
<空白>                        ::= ' '
<二重引用符>                  ::= '"'
<エスケープ文字>              ::= '\'

<演算子>                      ::= <等しい> |...| <含むまたは上>
```

前の行に使用できる演算子:

```
<等しい>                      ::= '='     | 'EQUAL_TO'
<等しくない>                  ::= '!='    | '<>' | 'NOT_EQUAL_TO'
<含まれる>                    ::= '='     | 'IN'
<含まれない>                  ::= '!='    | '<>' | 'NOT_IN'
```

```
<大きい>              ::= '>'     | 'GREATER_THAN'
<小さい>              ::= '<'     | 'LESS_THAN'
<以上>                ::= '>='    | 'GREATER_THAN_OR_EQUAL'
<以下>                ::= '<='    | 'LESS_THAN_OR_EQUAL'
<先頭>                ::= '=*'    | 'BEGINS_WITH'
<末尾>                ::= '*='    | 'ENDS_WITH'
<含む>                ::= '*=*'   | 'CONTAINS'
<含まない>            ::= '*<>*'  | 'NOT_CONTAINS'
<含まれるまたは下>    ::= 'IN_OR_BELOW'
<含まれるまたは上>    ::= 'IN_OR_ABOVE'
<間>                  ::= 'BETWEEN'
<間でない>            ::= 'NOT_BETWEEN'
<先頭でない>          ::= 'NOT_BEGINS_WITH'
<末尾でない>          ::= 'NOT_ENDS_WITH'
<今日>                ::= 'IS_TODAY'
<今日でない>          ::= 'IS_NOT_TODAY'
<過去何日以内>        ::= 'WITHIN_LAST_DAYS'
<過去何か月以内>      ::= 'WITHIN_LAST_MONTHS'
<今後何日以内>        ::= 'WITHIN_NEXT_DAYS'
<今後何か月以内>      ::= 'WITHIN_NEXT_MONTHS'
<過去何日以内でない>  ::= 'NOT_WITHIN_LAST_DAYS'
<過去何か月以内でない> ::= 'NOT_WITHIN_LAST_MONTHS'
<今後何日以内でない>  ::= 'NOT_WITHIN_NEXT_DAYS'
<今後何か月以内でない> ::= 'NOT_WITHIN_NEXT_MONTHS'
<含むまたは下>        ::= 'CONTAINS_OR_BELOW'
<含むまたは上>        ::= 'CONTAINS_OR_ABOVE'
```

使用に関する注

1つの式結合内部では、同じ結合タイプを使用する必要があります。

- 有効: `((x = y) & (a = y) & (x = a))`
- 無効: `((x = y) & (a = y) | (x = a))`

テキスト値は二重引用符で囲む必要がありますが、数値はその必要はありません。値自体に二重引用符が含まれる場合は、バックスラッシュでエスケープする必要があります。

- 有効な数値: `123.456`
- 有効なテキスト: `"abc"`
- 無効なテキスト: `abc`
- 有効なテキスト: `"ab\"c"`
- 無効なテキスト: `"ab"c"`
- 無効なテキスト: `ab"c`

別の式のグループと結合される式のグループは、括弧で囲む必要があります。

- 有効なグループ化: `((x = y) & (a = b)) | (n = r)`
- 無効なグループ化: `(x = y) & (a = b) | (n = r)`

付録 B　Apache HTTPサーバーおよびPHPのリビルド

この付録では、SA で Apache HTTP サーバーと PHP をリビルドして置き換える方法を説明します。SA には Apache HTTP サーバーと PHP が付属しているので、この付録が必要なのは、別のバージョンの Apache HTTP サーバーを使用するか、他のライブラリやモジュールをコンパイルして PHP に追加する必要がある場合に限られます。

SAでは、Web自動化プラットフォーム拡張 (APX) 用にApache HTTPサーバーとPHPが使用されます。詳細については、自動化プラットフォーム拡張 (APX) の作成 (61ページ) を参照してください。

APX HTTP環境の拡張

ここでは、Apache HTTPサーバーとPHPのリビルドによってAPX HTTP環境を拡張する方法を示します。

ここに示す作業は、コアアップグレードがすべて終了した後で行う必要があります。

マルチマスターメッシュを使用している場合、これらの作業はすべてのコアの各スライスに対して実行する必要があります。スライスコンポーネントバンドルの詳細については、『SA 管理ガイド』を参照してください。

PHPのリビルド

次の作業を実行してPHPをリビルドします。

1. PHPのソースを`http://www.php.net/` (英語サイト) からダウンロードします。
2. サーバー上のapxproxyがインストールされているディレクトリにソースを置きます。これは通常 `/opt/opsware/apxproxy`.
3. 次のコマンドを入力します。ダウンロードしたPHPのバージョンが異なる場合はバージョン番号を書き換えてください。

```
mkdir /build ; cp php-4.4.8.tar.gz /build; cd /build
gzip -dc php-4.4.8.tar.gz | tar xvf -
cd php-4.4.8
./configure --prefix=/opt/opsware/apxphp
--with-pear=/opt/opsware/apxphp/lib/pear
--with-config-file-path=/opt/opsware/apxphp/lib
--with-apxs2=/opt/opsware/apxhttpd/bin/apxs <any other options you>
make clean
make
```

4. `libphp4.so`の古いコピーをバックアップします。

```
cp /opt/opsware/apxhttpd/modules/libphp4.so /opt/opsware/apxhttpd/modules/
libphp4.so.backup
```

5 新しいlibphp4.soファイルをapxhhtpsディレクトリにコピーします。

```
cp libs/libphp4.so /opt/opsware/apxhttpd/modules/libphp4.so
```

6 参照ライブラリ全体がtool.listに存在することを確認します。

```
ldd ./libs/libphp4.so
```

出力の各エントリに対して、ファイルが
/etc/opt/opsware/ogfs/tool.listに存在することを確認します。

存在しないエントリがあれば、それを追加します。

7 apxphpフォルダーをバックアップします。

```
mv /opt/opsware/apxphp /opt/opsware/apxphp.orig
```

8 PHPをインストールします。

```
make install
```

9 OGFS を再ロードして再リンクし、/etc/opt/opsware/ogfs/tools.list に追加したものがすべてOGFSに表示されることを確認します。

```
/opt/opsware/ogfs/tools/rewink && /opt/opsware/ogfs/
tools/reload
```

10 apxproxyを再起動します。

```
/etc/opt/opsware/startup/apxproxy restart
```

Apacheのリビルド

次の作業を実行してApache HTTPサーバーをリビルドします。

1 Apache HTTP サーバーのソースコードをhttp://httpd.apache.org/ (英語サイト) からダウンロードします。

2 サーバー上のスライスコンポーネントバンドルがあるディレクトリにソースを置きます。スライスコンポーネントバンドルの詳細については、『SA 管理ガイド』を参照してください。

3 次のコマンドを入力します。ダウンロードしたhttpdのバージョンが異なる場合はバージョン番号を書き換えてください。

```
mkdir /build; cp httpd-2.2.8.tar.gz /build; cd /build
```

```
gzip -dc httpd-2.2.8.tar.gz | tar xf -
```

```
cd httpd-2.2.8
```

```
./configure --prefix=/opt/opsware/apxhttpd <その他必要なオプション>
```

SAでは現在次のオプションが使用されています。

```
-enable-mods-shared="actions alias auth_basic auth_digest authn_file authz_user
cgi deflate dir dumpio env expires headers ident logio log_config mime negotiation
rewrite userdir vhost_alias imagemap status"
```

```
--disable-dav
```

```
--with-port=8021
```

```
--with-expat=builtin
```

```
--without-pgsql
```

(SunOSのみ) 次のコマンドを入力します。

```
perl -pi -e 's/#define HAVE_GETADDRINFO 1/#undef HAVE_GETADDRINFO/g' ./srclib/apr/
include/arch/unix/apr_private.h

make
```

4 apxhttpディレクトリのバックアップを作成します。

```
mv /opt/opsware/apxhttpd /opt/opsware/apxhttpd.orig
```

5 Apacheをインストールします。

```
make install
```

6 新しいファイルをOGFSに再ロードして再リンクします。

```
/opt/opsware/ogfs/tools/rewink && /opt/opsware/ogfs/tools/reload
```

7 モジュールディレクトリのHTTPDおよび.soファイルは、外部ライブラリを参照している可能性があります。これらのライブラリはOGFSから見える (またはリンクされている) 必要があります。

OGFSにログインし、/opt/opsware/apxhttpd/bin/httpdと、/opt/opsware/apxhttpd/modulesにある.soファイルに対してLDDを実行して、表示されるすべてのファイルがOGFSに存在することを確認します。存在しない場合、ファイルを/etc/opt/opsware/ogfs/tool.list (OGFS外部) に追加してから、手順6を再実行して、すべてのファイルが/opt/opsware/apxhttpd/bin/httpdで使用できるようにします。

8 その後、PHPをリビルドする必要があります。PHPのリビルド (163ページ) を参照してください。

索引

B

BAT, 147

C

CIS, 145, 156

D

DisplayName, 149, 157, 158, 159

DTD, 153

G

GUID, 138, 148, 153, 155

H

HP Live Network, 145

I

Internet Information Services (IIS), 145

O

OCLI, 152

S

Sarbanes-Oxley (SoX) 法, 144

SAクライアント, 141, 152, 153, 155, 156, 157, 158, 159

stderr, 150

Stdout, 151

stdout, 146, 149, 156

SunOS, 152

U

Unixサービス, 145

V

VBS, 147, 155

Visual Basic, 145

W

Windowsレジストリ, 145

え

エラーチェック, 146

か

監査, 152

く

グローバルに一意のID番号 (GUID), 138

こ

コア, 155

コアまたは, 152

コンプライアンス, 145, 146

さ

サービス, 144, 157

し

シェル, 148, 150, 151, 152

終了コード, 146, 149

は

パスワード, 146

パラメーター , 145, 149, 150, 157

ふ

プラットフォーム, 149, 156

フレームワーク, 145, 155, 156, 158

へ

ベンチマーク, 146

め

メッシュに, 152

も

文字列, 149, 150, 151, 156, 157, 158